IT Request Form Documentation

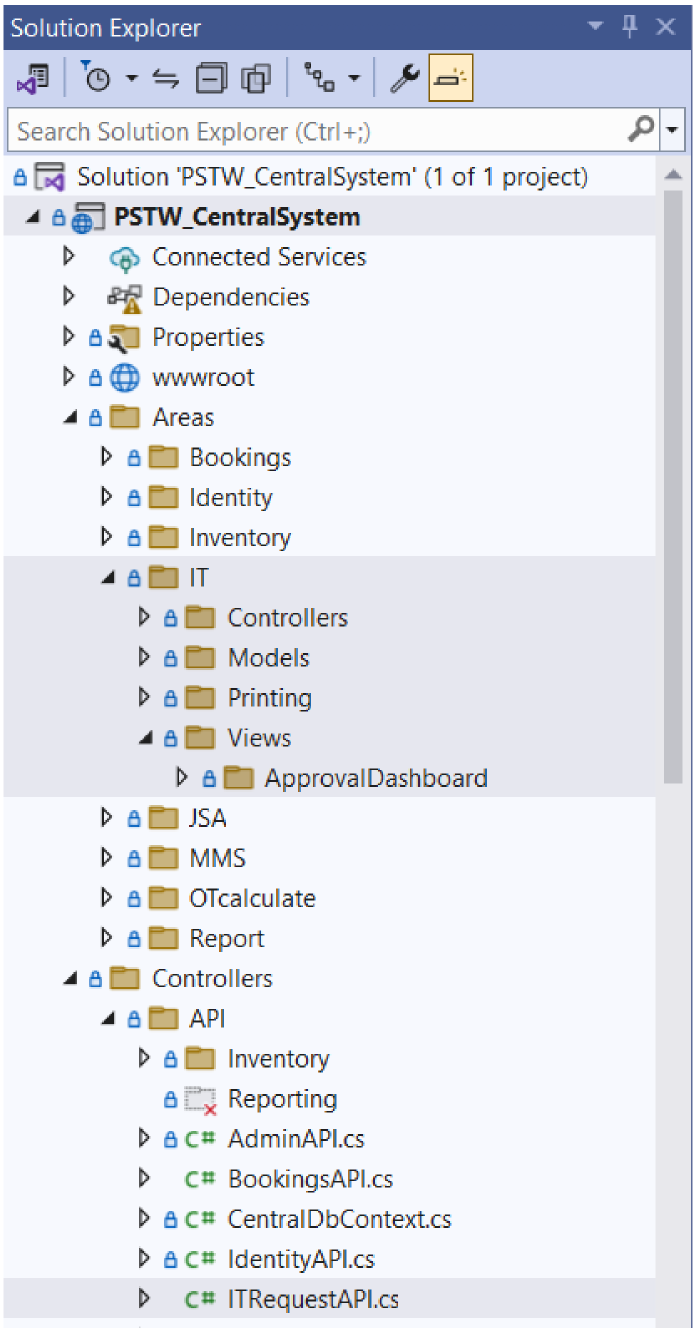
Prepared by: Harris Bin Muslisham

Table of Contents

FOLDER STRUCTURE

In the **PSTW_CentralSystem** project, the following new folder structure has been implemented under the **Areas** directory to accommodate the new functionality:

- **IT** Module: This module contains the **Controllers**, **Models**, **Printing** and **Views** directories, designed to encapsulate the logic and presentation for the overtime system.
- **API Directory** (Controllers/API): Under the Controllers folder, within the API sub-directory, the **ITRequestAPI.cs** file has been added. This API is crucial for supporting data interaction and reporting functionalities specifically related to the IT Request Form system.

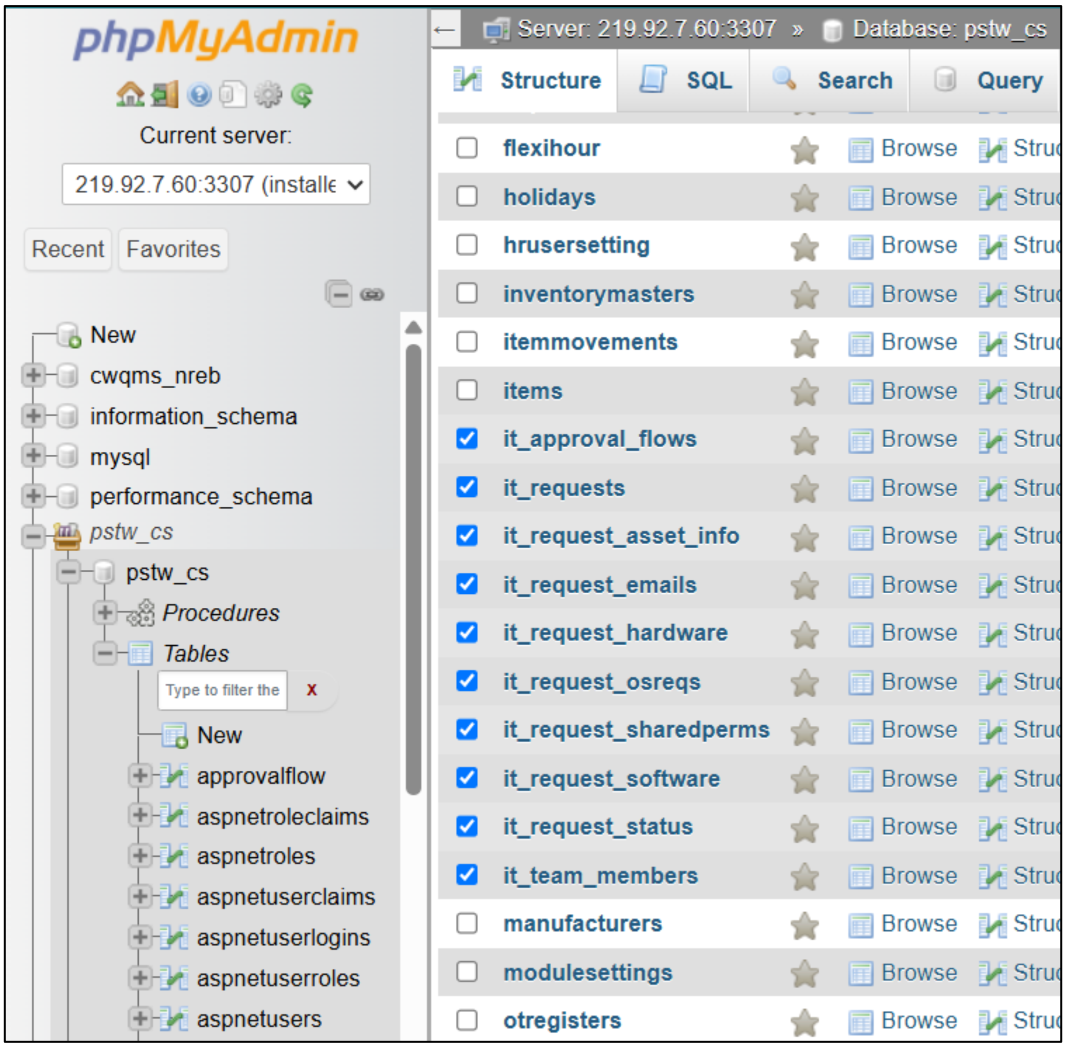


```
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'PSTW_CentralSystem' (1 of 1 project)
  PSTW_CentralSystem
    Connected Services
    Dependencies
    Properties
    wwwroot
    Areas
      Bookings
      Identity
      Inventory
      IT
        Controllers
        Models
        Printing
        Views
          ApprovalDashboard
      JSA
      MMS
      OTcalculate
      Report
    Controllers
      API
        Inventory
        Reporting
        AdminAPI.cs
        BookingsAPI.cs
        CentralDbContext.cs
        IdentityAPI.cs
        ITRequestAPI.cs
```

DATABASE TABLES

The **PSTW_CS** database has been extended with the introduction of the following new tables to support the system's enhanced features:

The tables added are **it_approval_flows, it_requests, it_request_asset_info, it_request_emails, it_request_hardware, it_request_osreqs, it_request_sharedperms, it_request_software, it_request_status, it_team_members, request_form_managers.**


phpMyAdmin

Current server: 219.92.7.60:3307 (installe

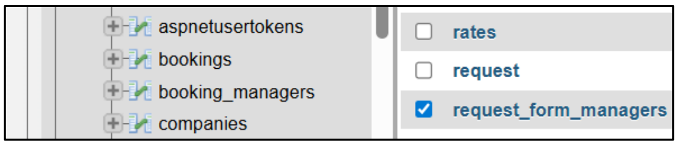
Recent | Favorites

- New
- cwqms_nreb
- information_schema
- mysql
- performance_schema
- *pstw_cs*
  - pstw_cs
    - *Procedures*
    - *Tables*
      - Type to filter the
      - New
      - approvalflow
      - aspnetroleclaims
      - aspnetroles
      - aspnetuserclaims
      - aspnetuserlogins
      - aspnetuserroles
      - aspnetusers

Server: 219.92.7.60:3307 » Database: pstw_cs

Structure | SQL | Search | Query

☐	flexihour	Browse	Struc
☐	holidays	Browse	Struc
☐	hrusersetting	Browse	Struc
☐	inventorymasters	Browse	Struc
☐	itemmovements	Browse	Struc
☐	items	Browse	Struc
✓	it_approval_flows	Browse	Struc
✓	it_requests	Browse	Struc
✓	it_request_asset_info	Browse	Struc
✓	it_request_emails	Browse	Struc
✓	it_request_hardware	Browse	Struc
✓	it_request_osreqs	Browse	Struc
✓	it_request_sharedperms	Browse	Struc
✓	it_request_software	Browse	Struc
✓	it_request_status	Browse	Struc
✓	it_team_members	Browse	Struc
☐	manufacturers	Browse	Struc
☐	modulesettings	Browse	Struc
☐	otregisters	Browse	Struc

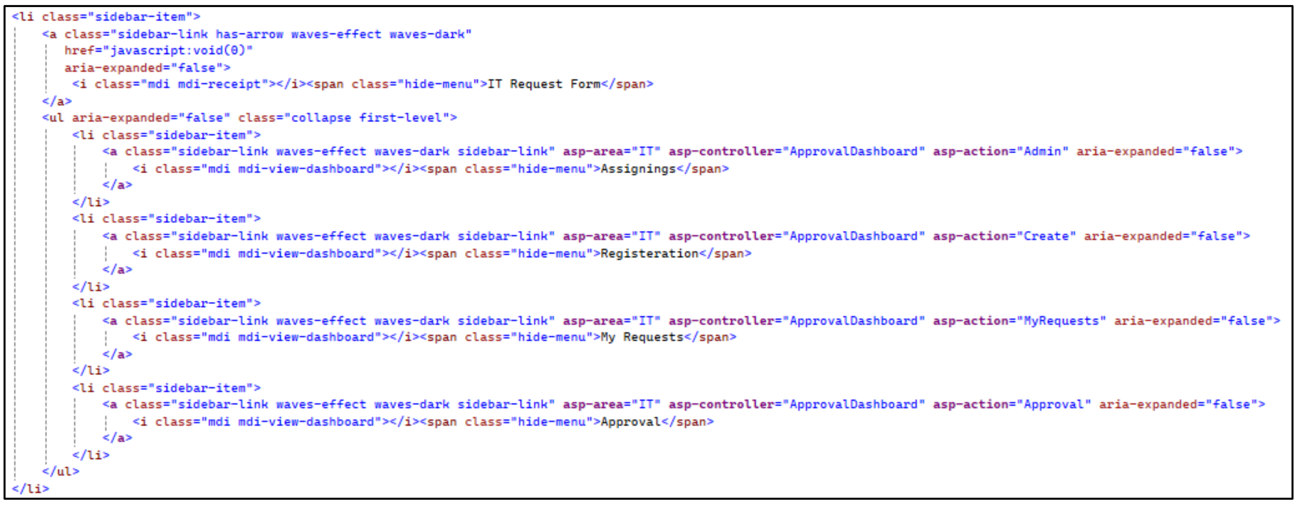

- aspnetusertokens
- bookings
- booking_managers
- companies

☐	rates
☐	request
✓	request_form_managers

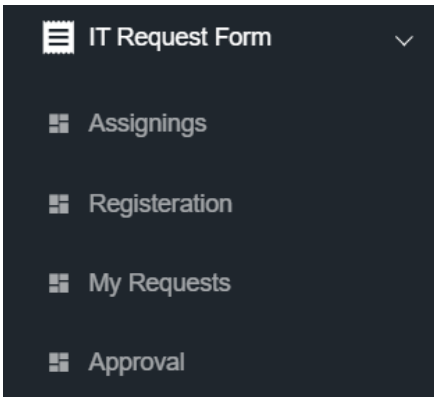
SIDEABR NAVIGATION (_Layout.cshtml)

The provided code snippet from **_Layout.cshtml** illustrates the integration of the new IT Request Form System's navigation into the application's main sidebar. The following key menu sections have been implemented:

- **IT Request Form** Menu Section: This section facilitates user-specific overtime functionalities. It includes a parent menu item, "User Overtime," which expands to reveal sub-links for:
    - "**Assignings**": Allows chosen users to create new approval flows as well as assigns who are the approvers. They are also responsible of choosing the IT team members
    - "**Registration**": Displays the first part of the IT Request Form for users to fill.
    - "**My Requests**": Shows the current status of their request along with the section b statuses of their request.
    - "**Approval**": Shows the current approval status of requests. These links are directing them to the relevant views within the IT Request Form module developed for this system.



```html
<li class="sidebar-item">
    <a class="sidebar-link has-arrow waves-effect waves-dark"
       href="javascript:void(0)"
       aria-expanded="false">
        <i class="mdi mdi-receipt"></i><span class="hide-menu">IT Request Form</span>
    </a>
    <ul aria-expanded="false" class="collapse first-level">
        <li class="sidebar-item">
            <a class="sidebar-link waves-effect waves-dark sidebar-link" asp-area="IT" asp-controller="ApprovalDashboard" asp-action="Admin" aria-expanded="false">
                <i class="mdi mdi-view-dashboard"></i><span class="hide-menu">Assignings</span>
            </a>
        </li>
        <li class="sidebar-item">
            <a class="sidebar-link waves-effect waves-dark sidebar-link" asp-area="IT" asp-controller="ApprovalDashboard" asp-action="Create" aria-expanded="false">
                <i class="mdi mdi-view-dashboard"></i><span class="hide-menu">Registeration</span>
            </a>
        </li>
        <li class="sidebar-item">
            <a class="sidebar-link waves-effect waves-dark sidebar-link" asp-area="IT" asp-controller="ApprovalDashboard" asp-action="MyRequests" aria-expanded="false">
                <i class="mdi mdi-view-dashboard"></i><span class="hide-menu">My Requests</span>
            </a>
        </li>
        <li class="sidebar-item">
            <a class="sidebar-link waves-effect waves-dark sidebar-link" asp-area="IT" asp-controller="ApprovalDashboard" asp-action="Approval" aria-expanded="false">
                <i class="mdi mdi-view-dashboard"></i><span class="hide-menu">Approval</span>
            </a>
        </li>
    </ul>
</li>
```

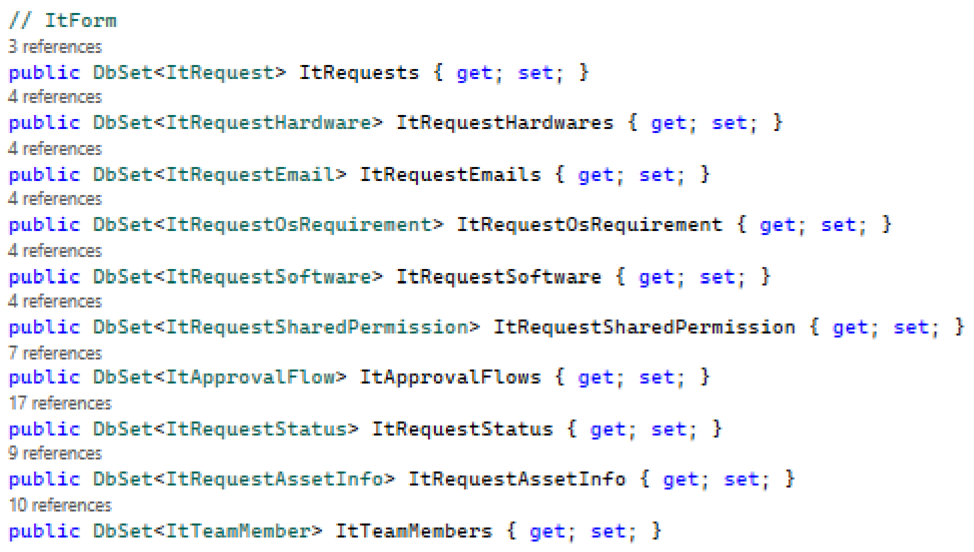


DATABASE CONTEXT CONFIGURATION(CentralSystemContext.cs)

The **CentralSystemContext.cs** file serves as the main gateway for the Entity Framework Core to interact with the application database.

The following DbSet properties have been explicitly added to this context to support new features and modules, particularly those related to the IT Request Form System.

This Db Set is important because it allows the application to easily query and store data for each of these models. It acts as a bridge, connecting our application data models directly to the **PSTW_CS** database tables.



```
// ItForm
3 references
public DbSet<ItRequest> ItRequests { get; set; }
4 references
public DbSet<ItRequestHardware> ItRequestHardwares { get; set; }
4 references
public DbSet<ItRequestEmail> ItRequestEmails { get; set; }
4 references
public DbSet<ItRequestOsRequirement> ItRequestOsRequirement { get; set; }
4 references
public DbSet<ItRequestSoftware> ItRequestSoftware { get; set; }
4 references
public DbSet<ItRequestSharedPermission> ItRequestSharedPermission { get; set; }
7 references
public DbSet<ItApprovalFlow> ItApprovalFlows { get; set; }
17 references
public DbSet<ItRequestStatus> ItRequestStatus { get; set; }
9 references
public DbSet<ItRequestAssetInfo> ItRequestAssetInfo { get; set; }
10 references
public DbSet<ItTeamMember> ItTeamMembers { get; set; }
```

CONTROLLERS

- **ApprovalDashboardController.cs**

The controller lives under the **IT area** and is **authorization-protected**.

- **Attributes**
  - ✓ [Area("IT")] — routes are prefixed with /IT/...
  - ✓ [Authorize] — only authenticated users can access any action in this controller

- **Approval( )** Function:

Purpose: This function is to show the main **Approval** page where approvers manage incoming IT requests.

How it works: When the approver opens this page, they will see two types of requests. First is pending meaning those requests has been sent by the users to be approved and second is completed meaning the requests has either been approved or rejected either way it will show up in the complete tab.

- **Create( )** Function:

Purpose: This function is to show the main **Registration** page where users are displayed a blank and new request form to be filled out.

How it works: When the user opens this page, they will see fields that must be filled out by them to create a new request form. Once filled out, options such as save as draft and send now are displayed for the users to click and choose.

- **Edit( )** Function:

Purpose: This function is to show users and allow them to revise back on what they had registered and change if necessary.

How it works: When the user saves the registered request as draft, they will immediately be brought to this Edit page where they are able to make any changes to it as they please. A 24-

hour time limit will be there in order to signify that they must edit or send the request as soon as possible. If the time limit has surpassed, the request shall be locked from editing.

- **MyRequests( )** Function:

Purpose: This function is to show the main **My Requests** page where users are able to see their requests statuses.

How it works: When the user opens this page, they will see displays of their requests' status based on the action the approvers have chosen, they are also able to see the section b status of each request that has been approved overall and what actions are needed to be done in order to complete the request form overall.

- **Admin( )** Function:

Purpose: This function is to show the main **Assignings** page where chosen users are displayed with two main tables.

How it works: When the user opens this page, they will see two types of tables, one is for the approval flow table where the user is able to create a new flow and assign new members to the flow. The other table is to assign the IT team members based on all the users registered to the PSTW Inventory System.

- **RequestReview ( )** Function:

Purpose: This function is to show the sub **xxx** page where requestors and approvers are displayed the information filled out during registration and the approval trail of the request.

How it works: When the user opens this page, they will be displayed of all the information that they had kicked in during the registration process. The approval trail which signifies if an approver had either approved or rejected said request will also be known as well as the timing on when they did it. The request shown is based on its statuId (Id of the request).

- **SectionB( )** Function:

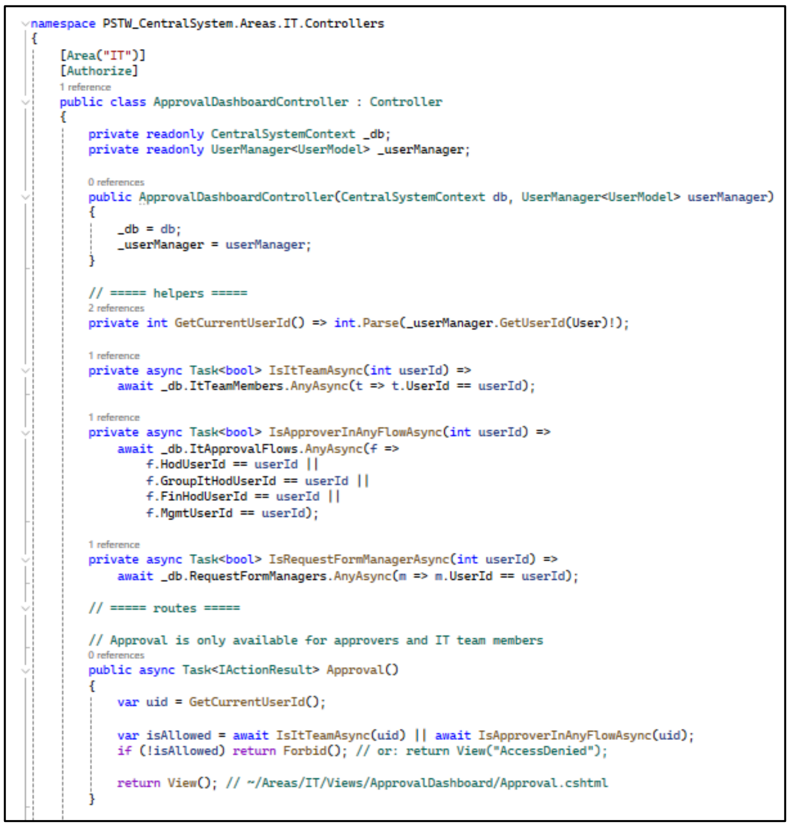
Purpose: This function is to show the main **SectionB** page where it team members displayed fields to be filled out.

How it works: When the it team member opens this page, they will see fields that must be filled out by them to complete the overall approved request form. Once filled out, options such as save as draft and send now are displayed for the users to click and choose.

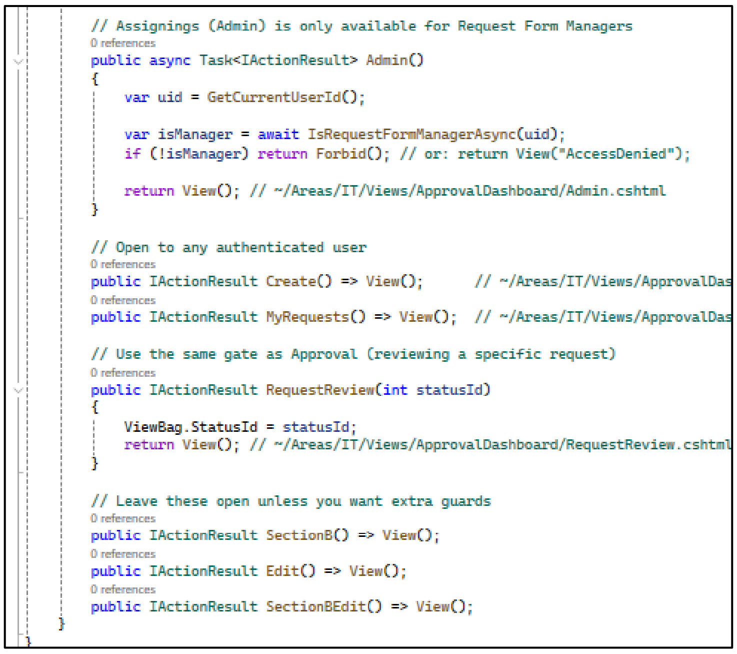
- **SectionBEdit( )** Function:

Purpose: This function is to show the IT team members the main **SectionBEdit** page and allow them to revise back on what they had filled out and change if necessary.

How it works: When the user saves the filled out section b as a draft, they will immediately be brought to this SectionBEdit page where they are able to make any changes to it as they please. Once they have done so, they are to send the section b to be accepted by the requestor and themselves.



```
namespace PSTW_CentralSystem.Areas.IT.Controllers
{
    [Area("IT")]
    [Authorize]
    1 reference
    public class ApprovalDashboardController : Controller
    {
        private readonly CentralSystemContext _db;
        private readonly UserManager<UserModel> _userManager;

        0 references
        public ApprovalDashboardController(CentralSystemContext db, UserManager<UserModel> userManager)
        {
            _db = db;
            _userManager = userManager;
        }

        // ===== helpers =====
        2 references
        private int GetCurrentUserId() => int.Parse(_userManager.GetUserId(User)!);

        1 reference
        private async Task<bool> IsItTeamAsync(int userId) =>
            await _db.ItTeamMembers.AnyAsync(t => t.UserId == userId);

        1 reference
        private async Task<bool> IsApproverInAnyFlowAsync(int userId) =>
            await _db.ItApprovalFlows.AnyAsync(f =>
                f.HodUserId == userId ||
                f.GroupItHodUserId == userId ||
                f.FinHodUserId == userId ||
                f.MgmtUserId == userId);

        1 reference
        private async Task<bool> IsRequestFormManagerAsync(int userId) =>
            await _db.RequestFormManagers.AnyAsync(m => m.UserId == userId);

        // ===== routes =====

        // Approval is only available for approvers and IT team members
        0 references
        public async Task<IActionResult> Approval()
        {
            var uid = GetCurrentUserId();

            var isAllowed = await IsItTeamAsync(uid) || await IsApproverInAnyFlowAsync(uid);
            if (!isAllowed) return Forbid(); // or: return View("AccessDenied");

            return View(); // ~/Areas/IT/Views/ApprovalDashboard/Approval.cshtml
        }
```



```
    // Assignings (Admin) is only available for Request Form Managers
    0 references
    public async Task<IActionResult> Admin()
    {
        var uid = GetCurrentUserId();

        var isManager = await IsRequestFormManagerAsync(uid);
        if (!isManager) return Forbid(); // or: return View("AccessDenied");

        return View(); // ~/Areas/IT/Views/ApprovalDashboard/Admin.cshtml
    }

    // Open to any authenticated user
    0 references
    public IActionResult Create() => View();      // ~/Areas/IT/Views/ApprovalDas
    0 references
    public IActionResult MyRequests() => View();  // ~/Areas/IT/Views/ApprovalDas

    // Use the same gate as Approval (reviewing a specific request)
    0 references
    public IActionResult RequestReview(int statusId)
    {
        ViewBag.StatusId = statusId;
        return View(); // ~/Areas/IT/Views/ApprovalDashboard/RequestReview.cshtml
    }

    // Leave these open unless you want extra guards
    0 references
    public IActionResult SectionB() => View();
    0 references
    public IActionResult Edit() => View();
    0 references
    public IActionResult SectionBEdit() => View();
}
}
```

MODELS

- **ItApprovalFlow.cs**

This model represents a **predefined approval route** for an IT request. Each record describes a named flow and exactly **who** (by user ID) must approve at each stage, so the system can consistently assign the right approvers without guessing per request.

- **ItApprovalFlowId (Primary Key):**

  The unique identifier for a flow. This is what other entities (like an IT request or status row) will store to say "use this flow."

- **FlowName (Human label):**

  A friendly name so admins and users can recognize the flow in dropdowns or settings (e.g., "Default - Corporate," "Plant Ops - Finance First"). The 200-char limit prevents overly long labels and keeps indexes efficient.

- **HodUserId (Head of Department):**

The **UserId** of the HoD who must approve first (or at the department stage) for requests using this flow. Because it's a non-nullable int, this stage is **required** because there must be a valid user assigned

- **GroupItHodUserId (Group IT HoD):**

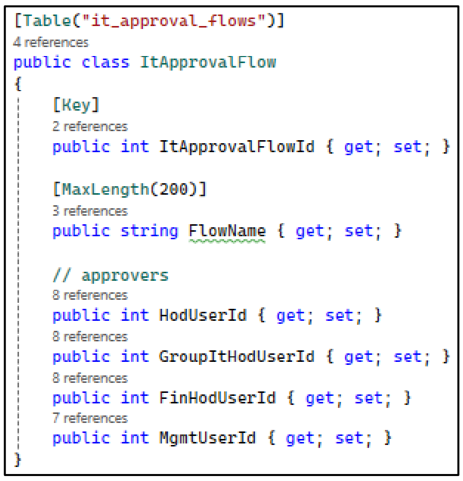
  The **UserId** for the Group IT Head of Department stage. Also required. The workflow engine will route the request here after HoD approval.

- **FinHodUserId (Finance HoD):**

  The **UserId** for the Finance approval stage. Required. Typically used to verify budget/financial impact before moving on.

- **MgmtUserId (Management):**

  The **UserId** for the final Management sign-off. Required. This usually represents the ultimate authority for high-impact IT requests.



```
[Table("it_approval_flows")]
4 references
public class ItApprovalFlow
{
    [Key]
    2 references
    public int ItApprovalFlowId { get; set; }

    [MaxLength(200)]
    3 references
    public string FlowName { get; set; }

    // approvers
    8 references
    public int HodUserId { get; set; }
    8 references
    public int GroupItHodUserId { get; set; }
    8 references
    public int FinHodUserId { get; set; }
    7 references
    public int MgmtUserId { get; set; }
}
```

- **ItRequest.cs**

This model is the **core record** of an IT request. It captures who's asking, when they need it, what they're asking for (hardware/software/email/etc.), and a "snapshot" of key profile details at the moment of submission so history stays accurate even if the user later changes departments or roles.

- Identity & ownership
  - ✓ **ItRequestId** ([Key] int): Primary key. What everything else (statuses, PDFs, approvals) will reference.

- ✓ **UserId** (int): The requester's user ID (FK to your AspNetUsers/UserModel). This ties the request back to the actual account that created it.

- o Snapshot fields (frozen at submission)
  - ✓ **StaffName** ([Required][MaxLength(200)] string): Display name printed everywhere (review pages, PDFs). Required.
  - ✓ **CompanyName / DepartmentName / Designation / Location** (string? up to 200): Optional labels so the request remains readable even if the org chart changes next month.
  - ✓ **EmploymentStatus** (string? up to 50): "Permanent / Contract / Temp / New Staff" (or your set). Handy for routing rules and eligibility checks.
  - ✓ **ContractEndDate** (DateTime?): Only relevant for contract/temp staff. Lets IT see if the need spans beyond the contract.
  - ✓ **PhoneExt** (string? up to 20): Quick contact detail for IT to clarify things fast.

- o Dates that drive behavior
  - ✓ **RequiredDate** (DateTime): When the user needs the item/service. Use this for prioritization and SLA visuals.
  - ✓ **SubmitDate** (DateTime): When the form was actually submitted (created or last "Send Now"). Useful for aging/turnaround metrics.
  - ✓ **FirstSubmittedAt** (DateTime?): The first time the request ever got submitted which stays the same even if resubmitted later. Great for lifecycle analytics.
  - ✓ **EditableUntil** (DateTime?): Dynamic edit window end (usually FirstSubmittedAt + N hours). Your UI can check this to show/hide "Edit" or force read-only.
  - ✓ **IsLockedForEdit** (bool): A hard stop switch. Even if EditableUntil hasn't passed, this can force a lock (e.g., when approvals have begun or after cancellation).

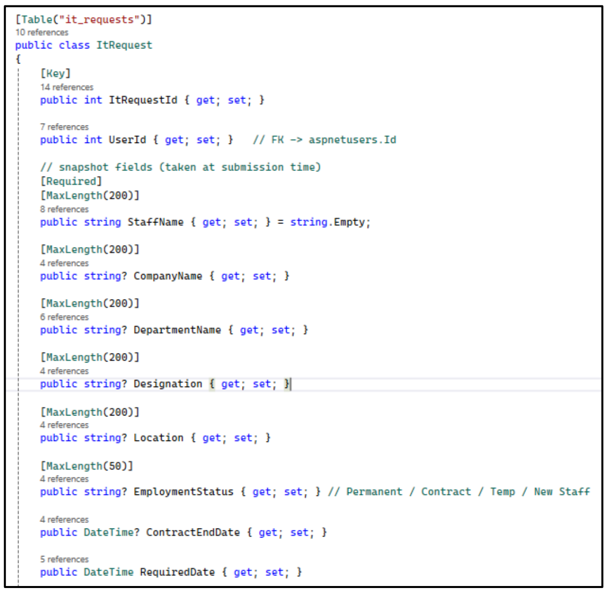
- o Navigation (what users request)
  - ✓ **Hardware**:ICollection<ItRequestHardware>
    Laptop/desktop/monitor/etc., quantities, specs, and justifications.
  - ✓ **Emails**:ICollection<ItRequestEmail>
    New mailbox, distribution lists, shared mailboxes, permissions.
  - ✓ **OsRequirements**:ICollection<ItRequestOsRequirement>
    Specific OS, versions, or build constraints.

- ✓ **Software**:ICollection<ItRequestSoftware>

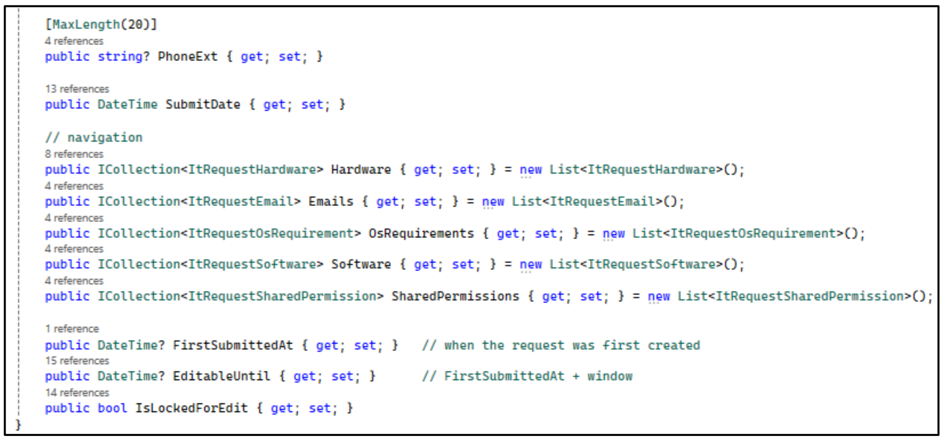
  Titles, license types, versions, and access scope.

- ✓ **SharedPermissions**:ICollection<ItRequestSharedPermission>

  Shared folders/drives/app roles for users who needs read/write and where.



```
[Table("it_requests")]
10 references
public class ItRequest
{
    [Key]
    14 references
    public int ItRequestId { get; set; }

    7 references
    public int UserId { get; set; }   // FK -> aspnetusers.Id

    // snapshot fields (taken at submission time)
    [Required]
    [MaxLength(200)]
    8 references
    public string StaffName { get; set; } = string.Empty;

    [MaxLength(200)]
    4 references
    public string? CompanyName { get; set; }

    [MaxLength(200)]
    6 references
    public string? DepartmentName { get; set; }

    [MaxLength(200)]
    4 references
    public string? Designation { get; set; }

    [MaxLength(200)]
    4 references
    public string? Location { get; set; }

    [MaxLength(50)]
    4 references
    public string? EmploymentStatus { get; set; } // Permanent / Contract / Temp / New Staff

    4 references
    public DateTime? ContractEndDate { get; set; }

    5 references
    public DateTime RequiredDate { get; set; }
```



```
    [MaxLength(20)]
    4 references
    public string? PhoneExt { get; set; }

    13 references
    public DateTime SubmitDate { get; set; }

    // navigation
    8 references
    public ICollection<ItRequestHardware> Hardware { get; set; } = new List<ItRequestHardware>();
    4 references
    public ICollection<ItRequestEmail> Emails { get; set; } = new List<ItRequestEmail>();
    4 references
    public ICollection<ItRequestOsRequirement> OsRequirements { get; set; } = new List<ItRequestOsRequirement>();
    4 references
    public ICollection<ItRequestSoftware> Software { get; set; } = new List<ItRequestSoftware>();
    4 references
    public ICollection<ItRequestSharedPermission> SharedPermissions { get; set; } = new List<ItRequestSharedPermission>();

    1 reference
    public DateTime? FirstSubmittedAt { get; set; }   // when the request was first created
    15 references
    public DateTime? EditableUntil { get; set; }      // FirstSubmittedAt + window
    14 references
    public bool IsLockedForEdit { get; set; }
}
```

- **ItRequestAssetInfo.cs**

Data is stored in it_request_asset_info. Keeping it in its own table lets you control edit/lock behavior for Section B without touching the main request.

- Identity & linkage
  - ✓ **Id (int):** Primary key for this asset info row.
  - ✓ **ItRequestId (int):** Points back to the parent IT request (ItRequest.ItRequestId). All the asset/account details below belong to that request.

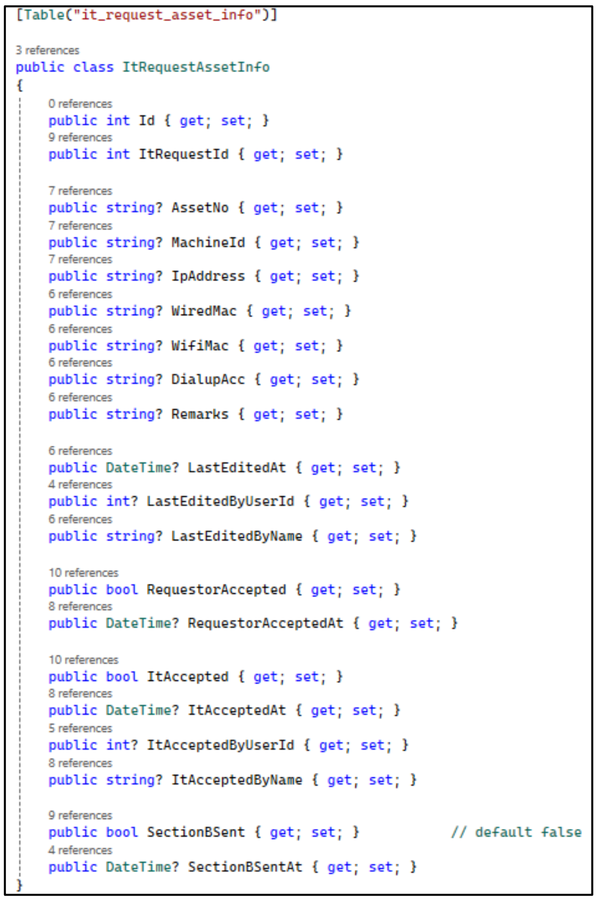
- Asset / network identifiers
  - ✓ **AssetNo** (string?): Company asset tag/asset register number (e.g., "AS-02458").
  - ✓ **MachineId** (string?): Local machine/computer name (e.g., hostname used in AD or on the label).
  - ✓ **IpAddress** (string?): Intended/assigned IP address.
  - ✓ **WiredMac** (string?): MAC address for the Ethernet adapter.
  - ✓ **WifiMac** (string?): MAC address for the Wi-Fi adapter.
  - ✓ **DialupAcc** (string?): Any dial-up/VPN/access account identifier if your environment still uses one.
  - ✓ **Remarks** (string?): Free-text notes are common uses: "User sits in 3F-24," "Requires static IP for legacy app," "Swap with AS-01977."

- Edit trail
  - ✓ **LastEditedAt** (DateTime?): Last time Section B was modified—use this to show "Saved 5 minutes ago" and for audits.
  - ✓ **LastEditedByUserId** (int?) & **LastEditedByName** (string?): Who made that last edit. Useful in review screens and PDFs when tracking changes.

- Acceptance by the requester
  - ✓ **RequestorAccepted** (bool): Has the requestor confirmed the details/assets are correct?
  - ✓ **RequestorAcceptedAt** (DateTime?): Timestamp of the requestor's acceptance. Typical flow: IT drafts Section B > sends for confirmation > requester checks and accepts.

- Acceptance by IT
  - ✓ **ItAccepted** (bool): IT's own acceptance that the asset info is correct and work is ready/complete on their side.
  - ✓ **ItAcceptedAt** (DateTime?): When IT accepted.
  - ✓ **ItAcceptedByUserId** (int?) & **ItAcceptedByName** (string?): Which IT staff finalized it. Helpful for accountability and follow-ups.

- Submission state of Section B
  - ✓ **SectionBSent** (bool): Whether Section B has been **sent** (as opposed to just saved as a draft). Defaults to false.
  - ✓ **SectionBSentAt** (DateTime?): When it was sent. This is your anchor for "awaiting acceptance" logic.



```
[Table("it_request_asset_info")]

3 references
public class ItRequestAssetInfo
{
    0 references
    public int Id { get; set; }
    9 references
    public int ItRequestId { get; set; }

    7 references
    public string? AssetNo { get; set; }
    7 references
    public string? MachineId { get; set; }
    7 references
    public string? IpAddress { get; set; }
    6 references
    public string? WiredMac { get; set; }
    6 references
    public string? WifiMac { get; set; }
    6 references
    public string? DialupAcc { get; set; }
    6 references
    public string? Remarks { get; set; }

    6 references
    public DateTime? LastEditedAt { get; set; }
    4 references
    public int? LastEditedByUserId { get; set; }
    6 references
    public string? LastEditedByName { get; set; }

    10 references
    public bool RequestorAccepted { get; set; }
    8 references
    public DateTime? RequestorAcceptedAt { get; set; }

    10 references
    public bool ItAccepted { get; set; }
    8 references
    public DateTime? ItAcceptedAt { get; set; }
    5 references
    public int? ItAcceptedByUserId { get; set; }
    8 references
    public string? ItAcceptedByName { get; set; }

    9 references
    public bool SectionBSent { get; set; }          // default false
    4 references
    public DateTime? SectionBSentAt { get; set; }
}
```

- **ItRequestEmail.cs**

This model captures the email-related line items inside an IT Request. Use it when someone needs a new mailbox, a replacement address, or an additional alias/shared mailbox as part of their submission. Each row is one email request linked back to its parent ItRequest.

- **Id** ([Key] int): Primary key for this email line.

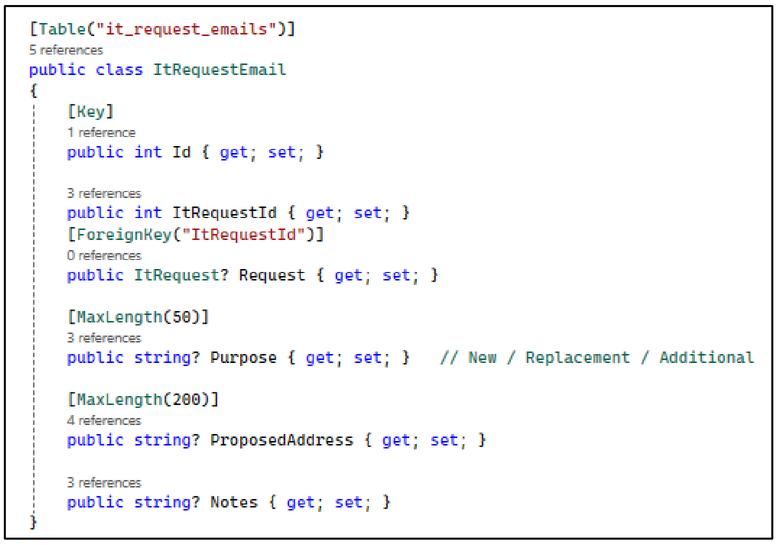
- **ItRequestId** (int): Foreign key to the parent request (ItRequest.ItRequestId).

- **Request** (ItRequest? with [ForeignKey("ItRequestId")]): EF navigation property. Lets you traverse from an email line to its owning request when you need header context (requestor name, department, dates) in lists or PDFs.

- **Purpose** (string?, max 50): Why this email is being requested. Common values:
  - ✓ **New** – create a brand-new mailbox
  - ✓ **Replacement** – change from old address to new (e.g., naming standard updates, marriage/name change)
  - ✓ **Additional** – extra mailbox, alias, or shared mailbox access Keeping it short (50 chars) helps standardize and filter easily.

- **ProposedAddress** (string?, max 200): The target email address or mailbox name the user has in mind (e.g., nur.aisyah@company.com, or finance.ap@company.com). This is what IT will validate against naming rules and availability.

- **Notes** (string?): Free-text details. Typical uses: preferred display name, mailbox type (shared vs user), required access list (“needs Ali and Mei as members”), forwarding rules, or special distribution group requirements.



```
[Table("it_request_emails")]
5 references
public class ItRequestEmail
{
    [Key]
    1 reference
    public int Id { get; set; }

    3 references
    public int ItRequestId { get; set; }
    [ForeignKey("ItRequestId")]
    0 references
    public ItRequest? Request { get; set; }

    [MaxLength(50)]
    3 references
    public string? Purpose { get; set; }   // New / Replacement / Additional

    [MaxLength(200)]
    4 references
    public string? ProposedAddress { get; set; }

    3 references
    public string? Notes { get; set; }
}
```

- **ItRequestHardware.cs**

This model represents a **hardware line item** inside an IT Request. Each row describes one requested piece of hardware (or type), why it's needed, and the business reason behind it is so approvers can make a decision quickly and IT can provision the right thing.

- **Id** ([Key] int): Primary key for this hardware line.

- **ItRequestId** (int): Foreign key back to the parent ItRequest.

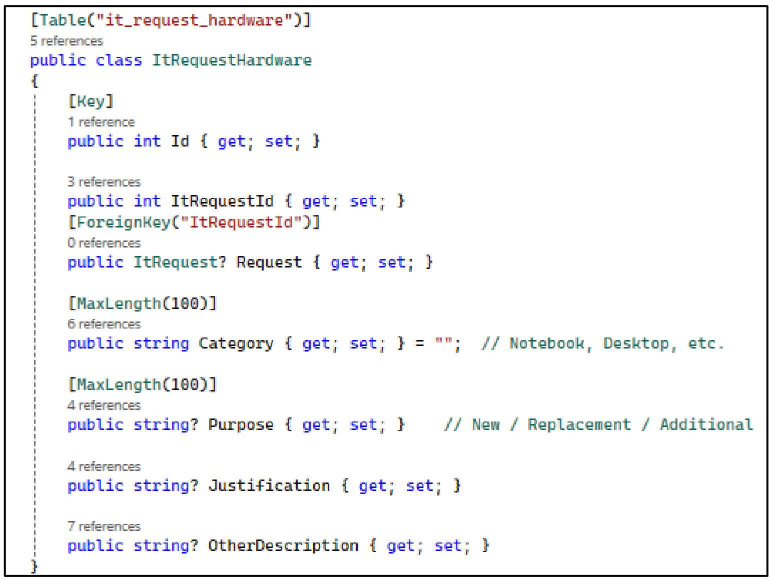
- **Request** (ItRequest? with [ForeignKey("ItRequestId")]): Navigation property for pulling header info (requestor, dates) when listing or printing.

- **Category** (string up to 100, required, default ""):
What the item is. Examples: "Notebook", "Desktop", "Monitor", "Printer", "Docking Station". This drives procurement rules, standard models, and approval routing. Since it's non-nullable, make sure the UI never leaves it as an empty string to validate on save.

- **Purpose** (string? up to 100):
Why the hardware is requested. Typical values: **New**, **Replacement**, **Additional**. Helpful for policy checks (e.g., replacements might require returning the old asset; additional units might need a stronger justification).

- **Justification** (string?):
  The business reason. This is what approvers read to understand *why* this is needed (e.g., “New hire starts 15 Jan,” “Existing laptop cannot run CAD,” “Team expanding by 3 FTE,” “Existing unit beyond economic repair”).
  In many organizations this is what makes or breaks the approval to encourage specific, outcome-focused explanations.

- **OtherDescription** (string?):
  Extra detail, especially when **Category = "Other"** or when you need to capture specs (e.g., “32GB RAM, 1TB SSD, dedicated GPU,” “Ergonomic keyboard for wrist issues,” “Color laser with duplex”).
  This keeps your Category tidy while allowing flexibility when the standard catalog doesn’t fit.

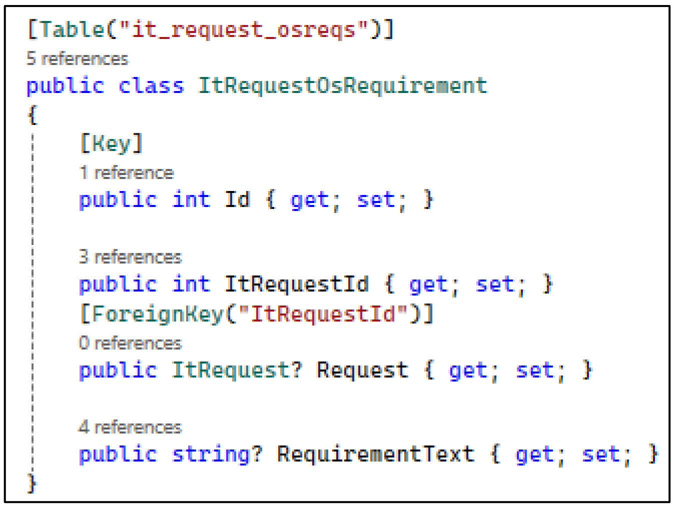


```
[Table("it_request_hardware")]
5 references
public class ItRequestHardware
{
    [Key]
    1 reference
    public int Id { get; set; }

    3 references
    public int ItRequestId { get; set; }
    [ForeignKey("ItRequestId")]
    0 references
    public ItRequest? Request { get; set; }

    [MaxLength(100)]
    6 references
    public string Category { get; set; } = "";  // Notebook, Desktop, etc.

    [MaxLength(100)]
    4 references
    public string? Purpose { get; set; }    // New / Replacement / Additional

    4 references
    public string? Justification { get; set; }

    7 references
    public string? OtherDescription { get; set; }
}
```

- **ItRequestOSRequirements.cs**

This model captures the **operating-system–related requirements** for an IT Request. Think of it as the place where a requester (or IT) spells out OS constraints that matter for deployment versions, builds, security settings, or special configurations that standard images don't cover. One request can have multiple OS requirement lines, each stored as a separate row.

- **Id** ([Key] int): Primary key for the requirement line.
- **ItRequestId** (int): Foreign key back to the parent ItRequest.
- **Request** (ItRequest? with [ForeignKey("ItRequestId")]): Navigation property when you need the request header (staff, department, dates) next to the OS details (e.g., in review screens or PDFs).
- **RequirementText** (string?): The actual OS requirement in human language. Keep it clear and specific, this is what provisioning/engineering will act on.



```
[Table("it_request_osreqs")]
5 references
public class ItRequestOsRequirement
{
    [Key]
    1 reference
    public int Id { get; set; }

    3 references
    public int ItRequestId { get; set; }
    [ForeignKey("ItRequestId")]
    0 references
    public ItRequest? Request { get; set; }

    4 references
    public string? RequirementText { get; set; }
}
```

- **ItRequestSharedPermissions.cs**

This model records **shared resource access** requested inside an IT Request—think shared folders/drives, project shares, or app workspaces. Each row says "which share" the user needs, so approvers can judge risk and IT can provision the correct ACLs.

  - **Id** ([Key] int): Primary key for this permission line.

  - **ItRequestId** (int): FK to the parent ItRequest.

  - **Request** (ItRequest? with [ForeignKey("ItRequestId")]): Navigation back to the request header (useful for PDFs and approval screens).

  - **ShareName** (string? up to 100):
    The target shared resource. Examples:

    ✓ \\fileserver01\Finance\Reports

    ✓ TeamSites\Operations

    ✓ Apps\PowerBI\SalesWorkspace
    Keep names consistent with how IT tracks shares to avoid provisioning mistakes.

- **CanRead** (bool):

  Read/browse/download. The baseline permission most users need.

- **CanWrite** (bool):

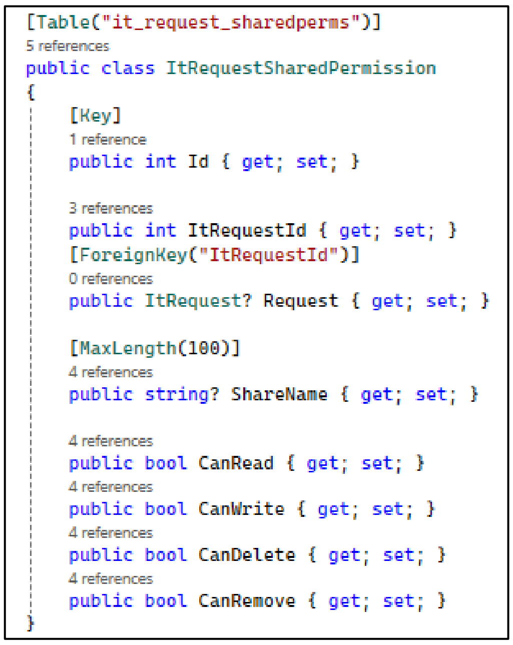
  Create or modify items (files/folders/records). Implies responsibility—approvers usually want a clear reason when this is requested.

- **CanDelete** (bool):

  Delete items from the share. Higher risk; often limited to power users or content owners.

- **CanRemove** (bool):

  Ability to remove **others'** access or membership (i.e., permission management / ownership actions). Treat this as **owner/maintainer** capability—granted sparingly.

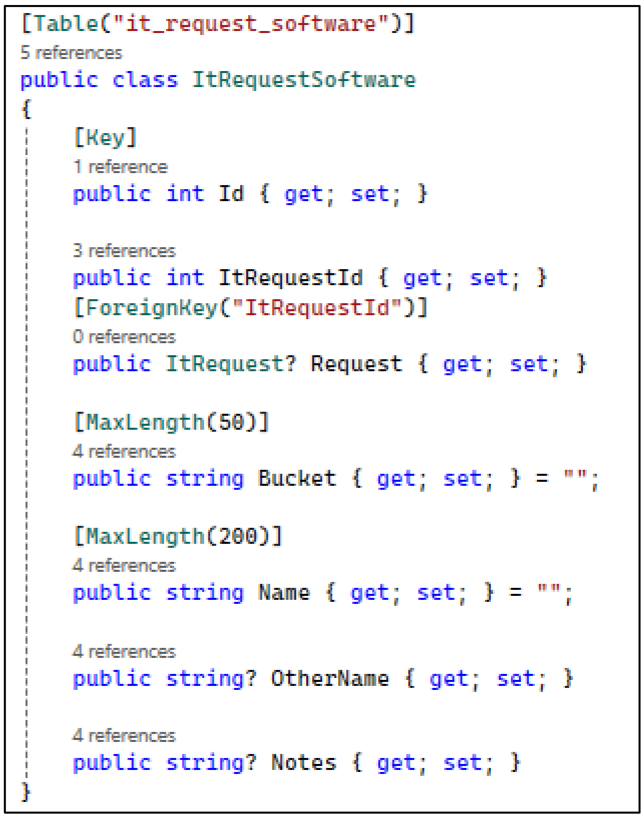


```
[Table("it_request_sharedperms")]
5 references
public class ItRequestSharedPermission
{
    [Key]
    1 reference
    public int Id { get; set; }

    3 references
    public int ItRequestId { get; set; }
    [ForeignKey("ItRequestId")]
    0 references
    public ItRequest? Request { get; set; }

    [MaxLength(100)]
    4 references
    public string? ShareName { get; set; }

    4 references
    public bool CanRead { get; set; }
    4 references
    public bool CanWrite { get; set; }
    4 references
    public bool CanDelete { get; set; }
    4 references
    public bool CanRemove { get; set; }
}
```

- **ItRequestSoftware.cs**

This model captures a **software line item** inside an IT Request. Each row represents one piece of software (or license) the requester needs, grouped into a simple "bucket" so approvers and IT know whether it's a standard app, a utility, or something bespoke.

- **Id** ([Key] int): Primary key for this software line.
- **ItRequestId** (int): FK to the parent ItRequest.
- **Request** (ItRequest? with [ForeignKey("ItRequestId")]): Navigation back to the request header (useful for PDFs/lists).
- **Bucket** (string up to 50, required, default ""):
  A quick classification that tells IT how to treat the item
- **Name** (string up to 200, required, default ""):
  The software's official name, ideally matching your internal catalog (e.g., "Microsoft Visio Standard 2021", "AutoCAD 2025"). Clear naming avoids provisioning mistakes.
- **OtherName** (string?):
  A free text fallback when **Name** doesn't exist in your catalog or when the user needs to specify a variant (edition, module, plugin). Example: "Plugin: Revit MEP Add-in", "Local language pack – Japanese".
- **Notes** (string?):
  Context that approvers care about: intended use, required version, seat type (user/device), number of seats, license owner/cost centre, any compliance constraints ("requires admin rights", "offline activation"). This is often what determines approval speed.



```
[Table("it_request_software")]
5 references
public class ItRequestSoftware
{
    [Key]
    1 reference
    public int Id { get; set; }

    3 references
    public int ItRequestId { get; set; }
    [ForeignKey("ItRequestId")]
    0 references
    public ItRequest? Request { get; set; }

    [MaxLength(50)]
    4 references
    public string Bucket { get; set; } = "";

    [MaxLength(200)]
    4 references
    public string Name { get; set; } = "";

    4 references
    public string? OtherName { get; set; }

    4 references
    public string? Notes { get; set; }
}
```

- **ItRequestStatus**

This model is the **live tracker** for an IT request's approval journey. It links a request to a chosen approval flow and records where each stage stands (HoD > Group IT HoD > Finance HoD > Management), with timestamps for when each stage acted. It also keeps a single **OverallStatus** you can display on boards and in reports.

- Identity & linkage
  - ✓ **StatusId** ([Key] int): Primary key for this status row.
  - ✓ **ItRequestId** (int + [ForeignKey] → ItRequest): Which request this status belongs to.
  - ✓ **ItApprovalFlowId** (int + [ForeignKey] → ItApprovalFlow): Which approver route is being used (who must approve at each stage).

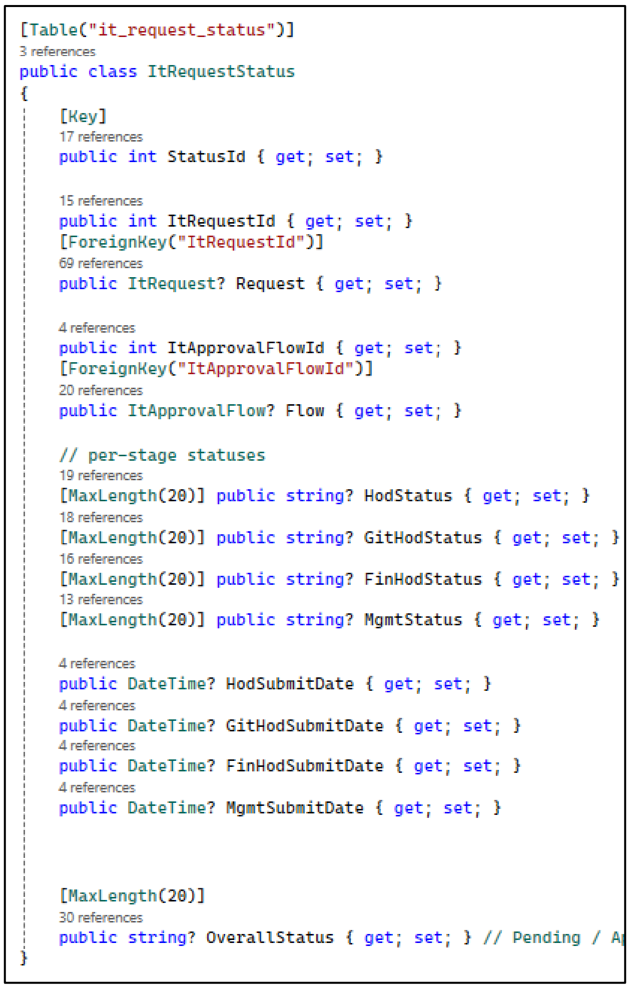
- Per-stage statuses

  Each stage has a short text status (up to 20 chars) and an optional timestamp for when it was submitted/acted on.
  - ✓ **HodStatus / HodSubmitDate**
  - ✓ **GitHodStatus / GitHodSubmitDate** (Group IT HoD)

- ✓ **FinHodStatus / FinHodSubmitDate**
- ✓ **MgmtStatus / MgmtSubmitDate**

- o Typical values for each *Status:
  - ✓ "Pending" – action required, approver hasn't decided yet
  - ✓ "Approved" – stage passed
  - ✓ "Rejected" – stage failed (usually stops the flow)
  - ✓ null – stage not yet opened/visible (e.g., earlier stages still pending)
- o The paired *SubmitDate captures when the approver **actually made a decision** (approve/reject). You can also set it when they forward/submit to the next stage if that matches your process.

- o Overall status
  - ✓ **OverallStatus** (string?, max 20): A single label you can show everywhere (e.g., "Pending", "Approved", "Rejected").



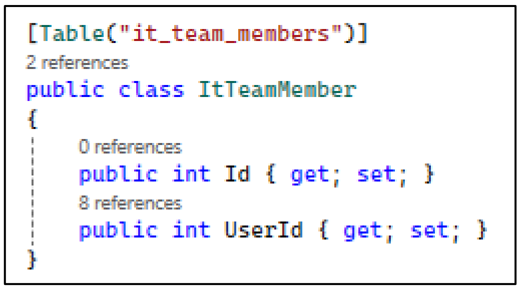
```
[Table("it_request_status")]
3 references
public class ItRequestStatus
{
    [Key]
    17 references
    public int StatusId { get; set; }

    15 references
    public int ItRequestId { get; set; }
    [ForeignKey("ItRequestId")]
    69 references
    public ItRequest? Request { get; set; }

    4 references
    public int ItApprovalFlowId { get; set; }
    [ForeignKey("ItApprovalFlowId")]
    20 references
    public ItApprovalFlow? Flow { get; set; }

    // per-stage statuses
    19 references
    [MaxLength(20)] public string? HodStatus { get; set; }
    18 references
    [MaxLength(20)] public string? GitHodStatus { get; set; }
    16 references
    [MaxLength(20)] public string? FinHodStatus { get; set; }
    13 references
    [MaxLength(20)] public string? MgmtStatus { get; set; }

    4 references
    public DateTime? HodSubmitDate { get; set; }
    4 references
    public DateTime? GitHodSubmitDate { get; set; }
    4 references
    public DateTime? FinHodSubmitDate { get; set; }
    4 references
    public DateTime? MgmtSubmitDate { get; set; }



    [MaxLength(20)]
    30 references
    public string? OverallStatus { get; set; } // Pending / A
}
```

- **ItTeamMember.cs**

This model is the **roster** of who counts as part of the IT team inside your system. It’s intentionally simple: one row ties a platform **UserId** to the IT team list, which you can then use for permissions, routing, and dashboards (e.g., “only IT team can accept Section B,” “assign tickets to IT members,” etc.).

  - **Id** (int): Primary key for this membership row. Internal identifier.

  - **UserId** (int): The user’s ID from your users table (e.g., AspNetUsers/UserModel). If a user’s ID appears here, they’re considered part of the IT team for app logic.



```
[Table("it_team_members")]
2 references
public class ItTeamMember
{
    0 references
    public int Id { get; set; }
    8 references
    public int UserId { get; set; }
}
```

- **ItRequestReportModel.cs**

This is the **view-model for the PDF**. It flattens everything the document needs like header/meta, Section A (requestor snapshot), Section B (IT asset details), line-item lists, acceptance info, approval timeline into one tidy object so QuestPDF template can render without doing extra joins or heavy logic.

- Header / meta (top banner of the PDF)
  - ✓ DocumentNo = "GITRF_01" helps version control across departments.
  - ✓ RevNo = "05" shows the controlled document revision.
  - ✓ DocPageNo = "1 of 1" is a placeholder you can override if you paginate.
  - ✓ EffectiveDate defaults to today; you can stamp policy-in-force date if needed.

- Section A - Requestor snapshot
  - ✓ **StaffName, CompanyName, DepartmentName, Designation, Location**: Who, where, and role context—printed plainly in the header table.
  - ✓ **EmploymentStatus, ContractEndDate**: “Permanent / Contract / Temp / New Staff”. If contract, the end date matters for hardware issuance decisions.
  - ✓ **RequiredDate**: When the user needs it—use to highlight urgency.
  - ✓ **PhoneExt**: Handy for follow-ups.

- Captured lists
  - ✓ **Hardware** (List<string>): Human-readable items like “Notebook (Additional)”, “Monitor 27””.
  - ✓ **Justification** (string?): The **single** justification text to be shown
  - ✓ **Emails** (List<string>): e.g., “New mailbox: finance.ap@company.com”
  - ✓ **OsRequirements** (List<string>): “Win11 23H2; BitLocker; Domain CORP”.

- ✓ **Software** (List<(Bucket, Name, Other, Notes)>): Each row can render as: Bucket/Name/Other/Notes.

- o Hardware purposes
  - ✓ **HwPurposeNewRecruitment / HwPurposeReplacement / HwPurposeAdditional** (bool)
    These power the left-side "Purpose" checkboxes the PDF shows. They summarize why hardware is requested without re-reading every line item.

- o Hardware selections
  - ✓ **HwDesktopAllIn, HwNotebookAllIn, HwDesktopOnly, HwNotebookOnly, HwNotebookBattery, HwPowerAdapter, HwMouse, HwExternalHdd, HwOtherText**
    These drive the right-side catalog-style checkboxes (and one free-text field). You can tick whichever items apply to this request overall and show "Other: …" when HwOtherText has value.

- o Section B - IT asset details
  - ✓ **AssetNo, MachineId, IpAddress, WiredMac, WifiMac, DialupAcc, Remarks**
    The identifiers IT cares about when provisioning/validating a device or account. Remarks is where you show concise install notes or location hints.

- o Acceptance
  - ✓ **RequestorName, RequestorAcceptedAt**: "I have received/verified the above."
  - ✓ **ItCompletedBy, ItAcceptedAt**: Who in IT finalized the work and when. These pair nicely with signature lines in the PDF.
- o Status
  - ✓ ItRequestId, StatusId: Glue back to your database rows; useful for QR codes or "Ref: #12345".
  - ✓ **OverallStatus**: Single label for the entire document ("Pending / Approved / Rejected / Complete").
  - ✓ **SubmitDate**: Original submission timestamp to show aging/turnaround context.

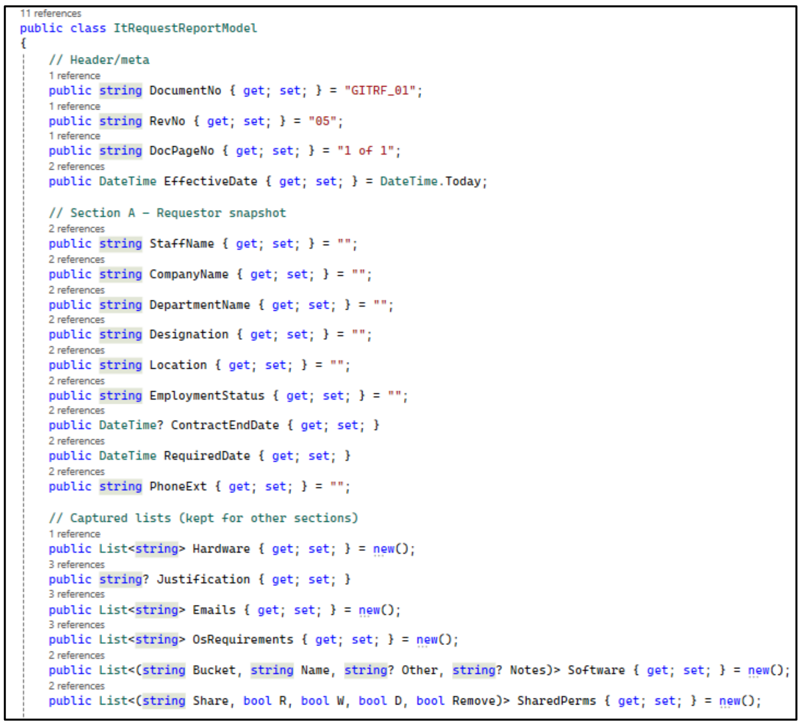
- o Approval flow dates

- ✓ **HodSubmitDate / GitHodSubmitDate / FinHodSubmitDate / MgmtSubmitDate**

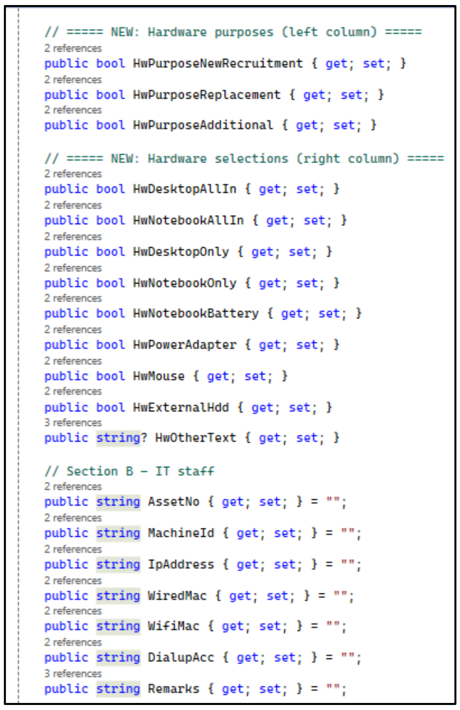
  These timestamps let you print a small “approval trail” table (Stage | Date). If a stage is skipped, leave blank.

- o Approvers
  - ✓ **HodApprovedBy / GitHodApprovedBy / FinHodApprovedBy / MgmtApprovedBy**

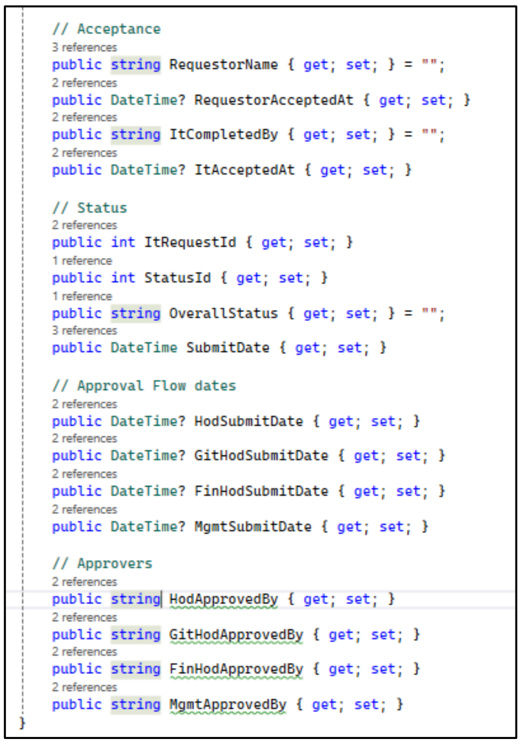
    Human-readable names you can print under each stage (e.g., “Approved by: Nurul A.”). Populate these in the controller when you build the report model to avoid EF lookups inside the PDF renderer.



```
11 references
public class ItRequestReportModel
{
    // Header/meta
    1 reference
    public string DocumentNo { get; set; } = "GITRF_01";
    1 reference
    public string RevNo { get; set; } = "05";
    1 reference
    public string DocPageNo { get; set; } = "1 of 1";
    2 references
    public DateTime EffectiveDate { get; set; } = DateTime.Today;

    // Section A – Requestor snapshot
    2 references
    public string StaffName { get; set; } = "";
    2 references
    public string CompanyName { get; set; } = "";
    2 references
    public string DepartmentName { get; set; } = "";
    2 references
    public string Designation { get; set; } = "";
    2 references
    public string Location { get; set; } = "";
    2 references
    public string EmploymentStatus { get; set; } = "";
    2 references
    public DateTime? ContractEndDate { get; set; }
    2 references
    public DateTime RequiredDate { get; set; }
    2 references
    public string PhoneExt { get; set; } = "";

    // Captured lists (kept for other sections)
    1 reference
    public List<string> Hardware { get; set; } = new();
    3 references
    public string? Justification { get; set; }
    3 references
    public List<string> Emails { get; set; } = new();
    3 references
    public List<string> OsRequirements { get; set; } = new();
    2 references
    public List<(string Bucket, string Name, string? Other, string? Notes)> Software { get; set; } = new();
    2 references
    public List<(string Share, bool R, bool W, bool D, bool Remove)> SharedPerms { get; set; } = new();
```



```
    // ===== NEW: Hardware purposes (left column) =====
    2 references
    public bool HwPurposeNewRecruitment { get; set; }
    2 references
    public bool HwPurposeReplacement { get; set; }
    2 references
    public bool HwPurposeAdditional { get; set; }

    // ===== NEW: Hardware selections (right column) =====
    2 references
    public bool HwDesktopAllIn { get; set; }
    2 references
    public bool HwNotebookAllIn { get; set; }
    2 references
    public bool HwDesktopOnly { get; set; }
    2 references
    public bool HwNotebookOnly { get; set; }
    2 references
    public bool HwNotebookBattery { get; set; }
    2 references
    public bool HwPowerAdapter { get; set; }
    2 references
    public bool HwMouse { get; set; }
    2 references
    public bool HwExternalHdd { get; set; }
    3 references
    public string? HwOtherText { get; set; }

    // Section B - IT staff
    2 references
    public string AssetNo { get; set; } = "";
    2 references
    public string MachineId { get; set; } = "";
    2 references
    public string IpAddress { get; set; } = "";
    2 references
    public string WiredMac { get; set; } = "";
    2 references
    public string WifiMac { get; set; } = "";
    2 references
    public string DialupAcc { get; set; } = "";
    3 references
    public string Remarks { get; set; } = "";
```

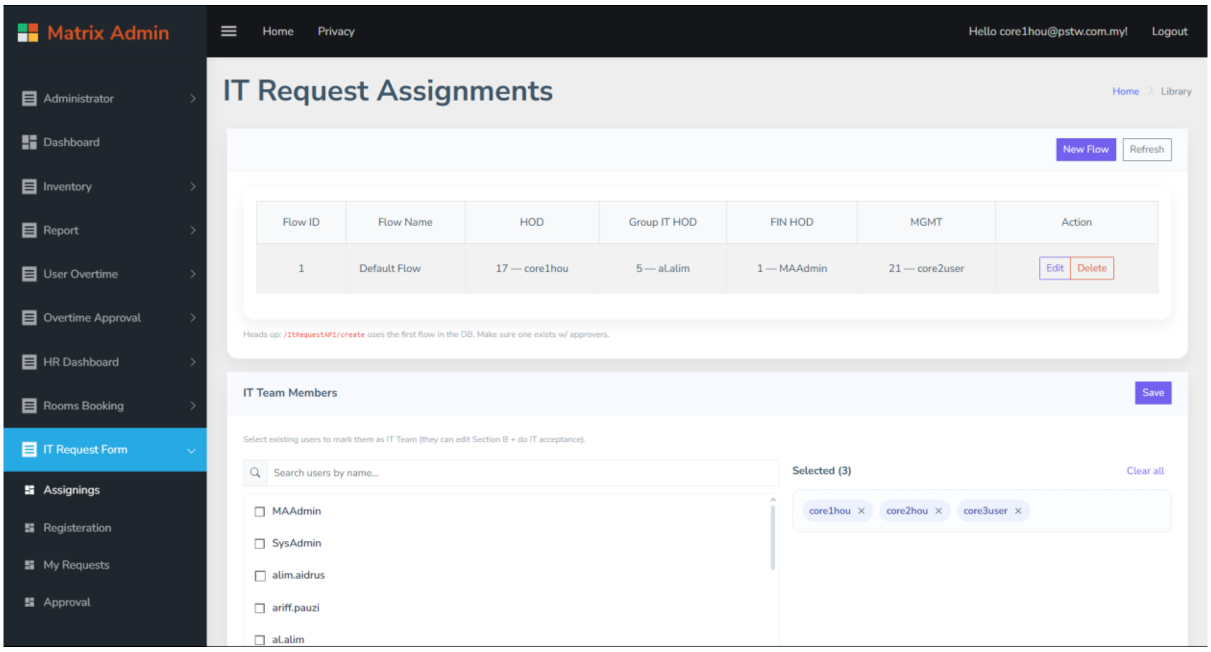


```
    // Acceptance
    3 references
    public string RequestorName { get; set; } = "";
    2 references
    public DateTime? RequestorAcceptedAt { get; set; }
    2 references
    public string ItCompletedBy { get; set; } = "";
    2 references
    public DateTime? ItAcceptedAt { get; set; }

    // Status
    2 references
    public int ItRequestId { get; set; }
    1 reference
    public int StatusId { get; set; }
    1 reference
    public string OverallStatus { get; set; } = "";
    3 references
    public DateTime SubmitDate { get; set; }

    // Approval Flow dates
    2 references
    public DateTime? HodSubmitDate { get; set; }
    2 references
    public DateTime? GitHodSubmitDate { get; set; }
    2 references
    public DateTime? FinHodSubmitDate { get; set; }
    2 references
    public DateTime? MgmtSubmitDate { get; set; }

    // Approvers
    2 references
    public string HodApprovedBy { get; set; }
    2 references
    public string GitHodApprovedBy { get; set; }
    2 references
    public string FinHodApprovedBy { get; set; }
    2 references
    public string MgmtApprovedBy { get; set; }
}
```

VIEWS & IT REQUEST API

- **ApprovalDashboard**

  - Admin.cshtml



IT Request Assignments

Flow ID	Flow Name	HOD	Group IT HOD	FIN HOD	MGMT	Action
1	Default Flow	17 — core1hou	5 — al.alim	1 — MAAdmin	21 — core2user	Edit Delete

Heads up: `/ItRequestAPI/create` uses the first flow in the DB. Make sure one exists w/ approvers.

IT Team Members

Select existing users to mark them as IT Team (they can edit Section B + do IT acceptance).

- ☐ MAAdmin
- ☐ SysAdmin
- ☐ alim.aidrus
- ☐ ariff.pauzi
- ☐ al.alim

Selected (3)

core1hou × core2hou × core3user ×

This file (Admin.cshtml) is the admin page where you manage approval flows and mark users as part of the IT Team.

✓ **<div id="flowApp">…</div>**

- **Purpose:** This is the root container for the Vue app. Everything inside will be reactive and controlled by Vue.

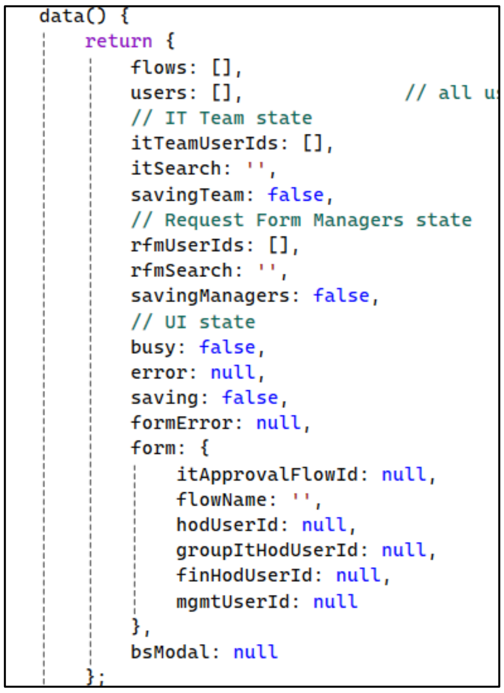
✓ **const flowApp = Vue.createApp({...})**

- **Purpose:** This creates the Vue instance and wires up state, computed helpers, methods, and lifecycle so the page can load, edit, and save data.

✓ **data()**

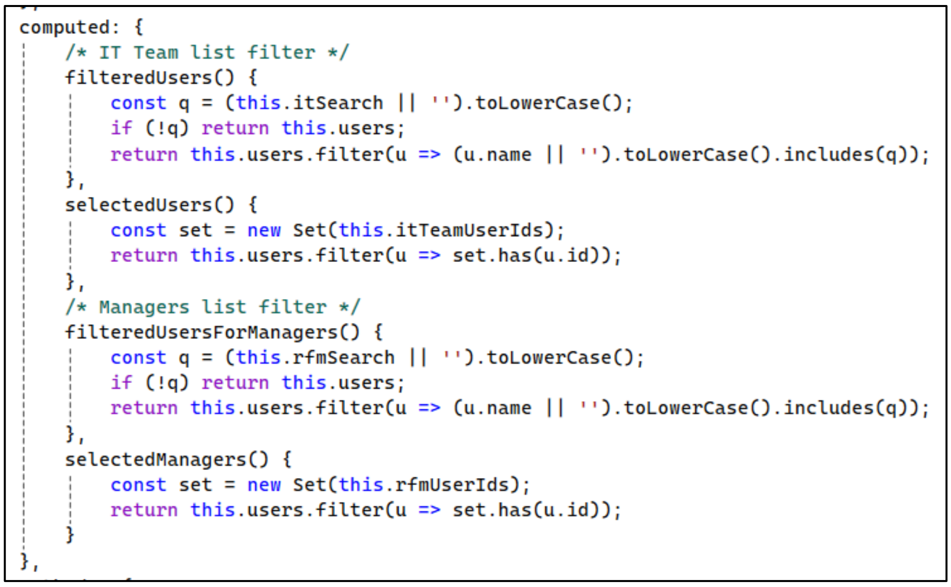
- **flows:** Holds the list of approval flows pulled from the API. Each item is normalized for consistent casing, so the UI does not care whether the API returns Pascal Case or camelCase.
- **users:** The flat list of users used to populate dropdowns and the IT Team selector. Kept simple with only the fields we need for display and lookups.

- **itTeamUserIds:** An array of selected user IDs that represent the current IT Team. This is what gets sent back to the server when you hit Save.
- **itSearch**: The live search term for filtering users in the left panel. Keeps the list manageable as the directory grows.
- **rfmUserIds:** An array of selected user IDs that represent the current request form manager. This is what gets sent back to the server when you hit Save.
- **rfmSearch**: The live search term for filtering users in the left panel. Keeps the list manageable as the directory grows.
- **busy, error**: Loading and error flags for the main flows table. These control the "Loading…" and error alert so the user knows what is happening.
- **saving, savingTeam, savingManagers**: Separate save flags for the flow modal and IT Team action. This prevents double submissions and gives proper button feedback.
- **formError**: Validation or API error message specific to the flow modal. Shown above the inputs to guide the admin to fix it quickly.
- **form**: The working copy of a flow while creating or editing. It mirrors the fields in the database and is reset or cloned depending on context.
- **bsModal:** A reference to the Bootstrap modal instance. Stored so we can reliably open and close the modal from Vue methods.



```
data() {
    return {
        flows: [],
        users: [],            // all u
        // IT Team state
        itTeamUserIds: [],
        itSearch: '',
        savingTeam: false,
        // Request Form Managers state
        rfmUserIds: [],
        rfmSearch: '',
        savingManagers: false,
        // UI state
        busy: false,
        error: null,
        saving: false,
        formError: null,
        form: {
            itApprovalFlowId: null,
            flowName: '',
            hodUserId: null,
            groupItHodUserId: null,
            finHodUserId: null,
            mgmtUserId: null
        },
        bsModal: null
    };
```

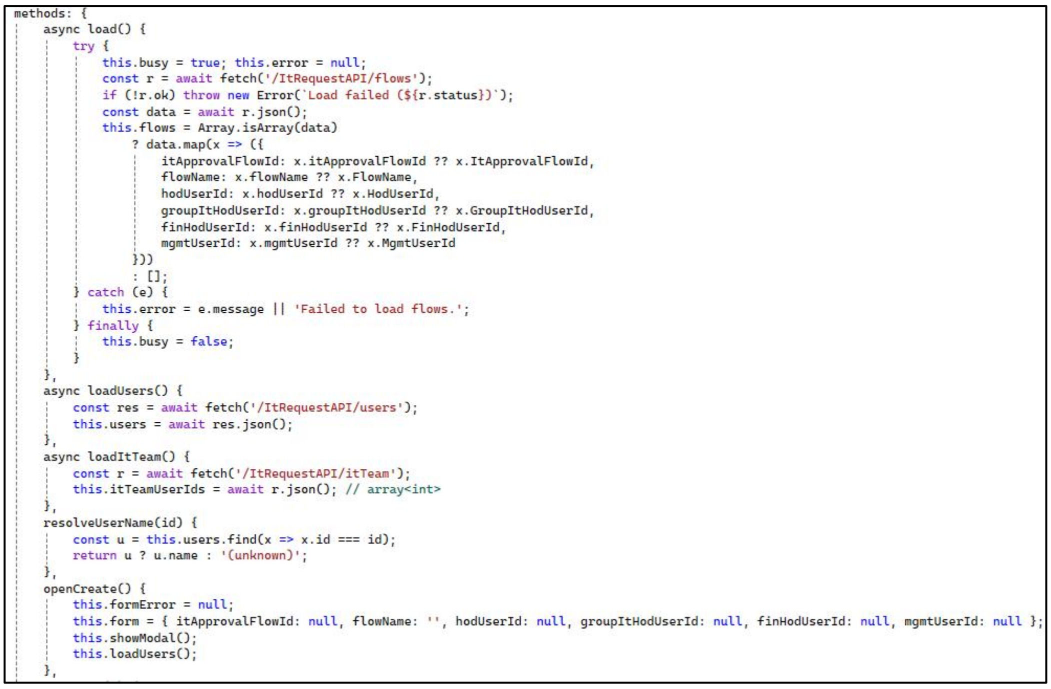
- ✓ **computed**
  - **filteredUsers, filteredUsers:** Returns the users list filtered by the current search term. This avoids recomputing the same filter inside the template and keeps the view snappy even with many users.
  - **selectedUsers, selectedManagers**: Projects itTeamUserIds and rfmUserIds back into full user objects so the right panel can show "chips" with names and remove buttons. Using a Set makes the lookup fast and clean.



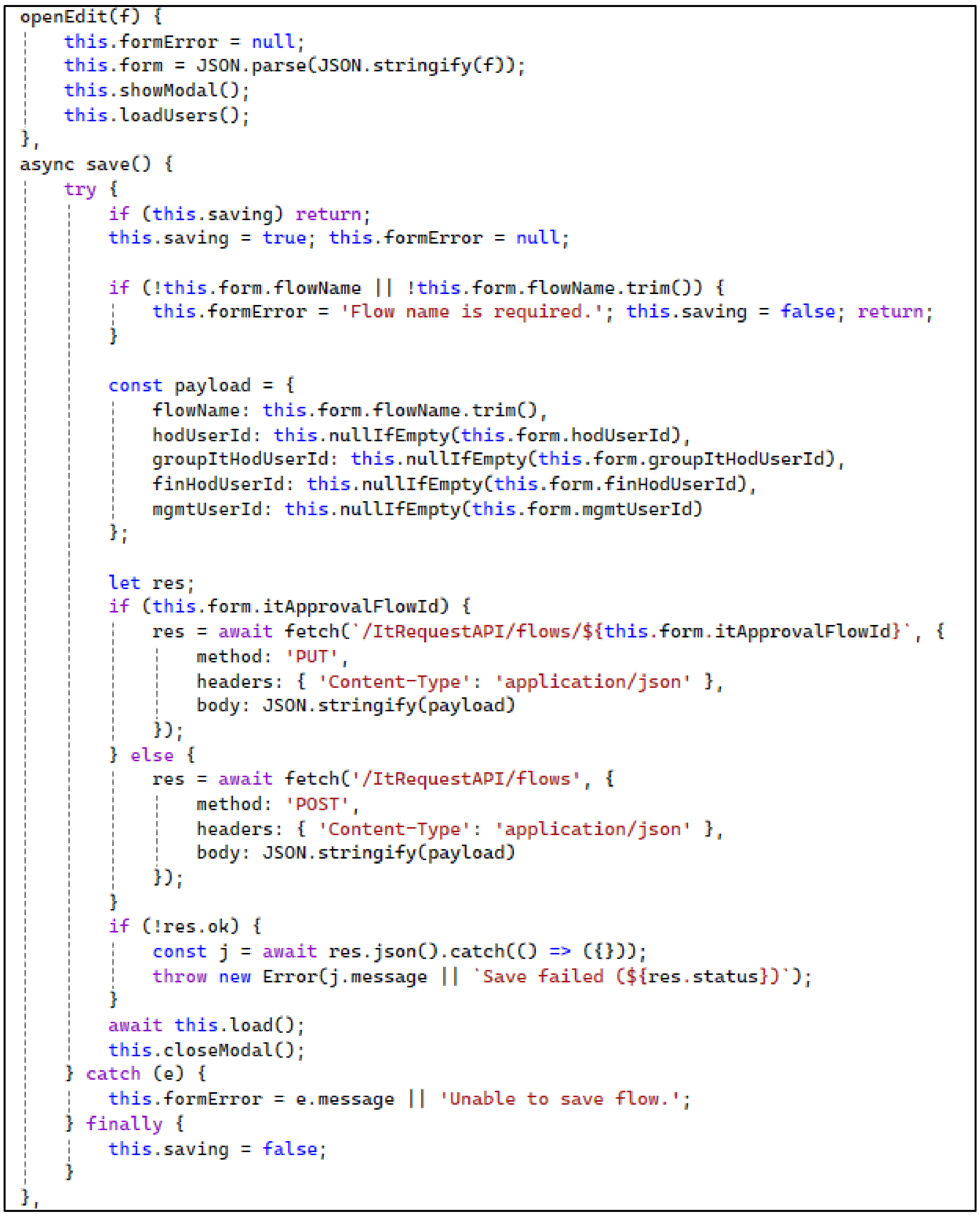
```
computed: {
    /* IT Team list filter */
    filteredUsers() {
        const q = (this.itSearch || '').toLowerCase();
        if (!q) return this.users;
        return this.users.filter(u => (u.name || '').toLowerCase().includes(q));
    },
    selectedUsers() {
        const set = new Set(this.itTeamUserIds);
        return this.users.filter(u => set.has(u.id));
    },
    /* Managers list filter */
    filteredUsersForManagers() {
        const q = (this.rfmSearch || '').toLowerCase();
        if (!q) return this.users;
        return this.users.filter(u => (u.name || '').toLowerCase().includes(q));
    },
    selectedManagers() {
        const set = new Set(this.rfmUserIds);
        return this.users.filter(u => set.has(u.id));
    }
},
```

- ✓ **methods**
  - **load():** Fetches the flows from /ItRequestAPI/flows, normalizes property casing, and updates flows. It toggles busy and captures any network or server errors so the table always reflects a clear state.
  - **loadUsers()**: Gets the user directory from /ItRequestAPI/users. This powers the dropdowns in the flow modal and the IT Team list, so it is called on mount and when the modal opens.
  - **loadItTeam()**: Pulls the current IT Team from /ItRequestAPI/itTeam as an array of IDs. The UI then highlights those users and fills the Selected section on the right.
  - **resolveUserName(id)**: Maps a user ID to a display name using the already loaded users array. If the user list has not arrived yet, it returns a safe placeholder so the table does not break.

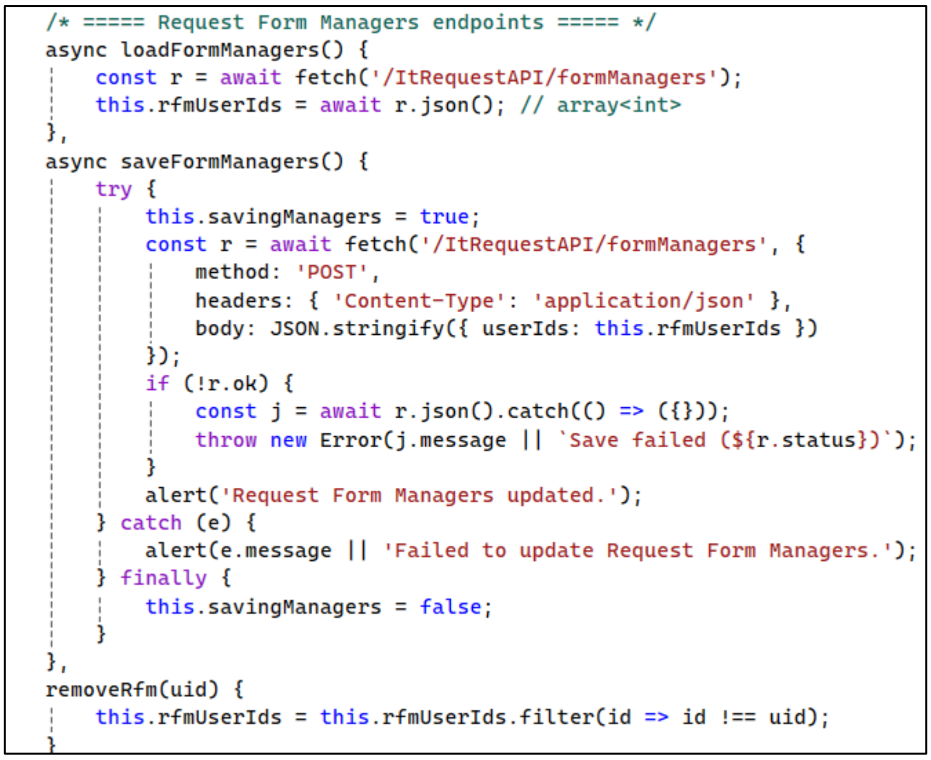
- **openCreate()**: Resets form to a clean state, clears formError, opens the modal, and ensures users are available for the approver dropdowns. This keeps the create path predictable and quick.
- **openEdit(f)**: Deep clones the selected flow into form, clears formError, opens the modal, and ensures users are loaded. Cloning avoids mutating the table row before saving.
- **save()**: Validates the form, builds a minimal payload with nulls for empty picks, then calls POST /ItRequestAPI/flows for create or PUT /ItRequestAPI/flows/{id} for update. On success it reloads the list and closes the modal. On failure it shows a friendly message in formError.
- **del(f)**: Confirms with the admin, then calls DELETE /ItRequestAPI/flows/{id}. If the API returns an error, it surfaces the message so the admin knows why deletion failed.
- **nullIfEmpty(v)**: Utility that converts empty values to null and coerces numeric strings. Keeps payloads clean and avoids accidental empty strings in integer fields.
- **showModal() / closeModal()**: Create and control the Bootstrap modal instance. These helpers make the modal behavior consistent no matter how it is invoked.
- **removeIt(uid)**: Removes a user from the selected IT Team by filtering the ID list. This powers the small “x” on each selected chip.
- **saveItTeam()**: Sends { userIds: itTeamUserIds } to POST /ItRequestAPI/itTeam. Shows a simple success or error alert and protects against double clicks with savingTeam.



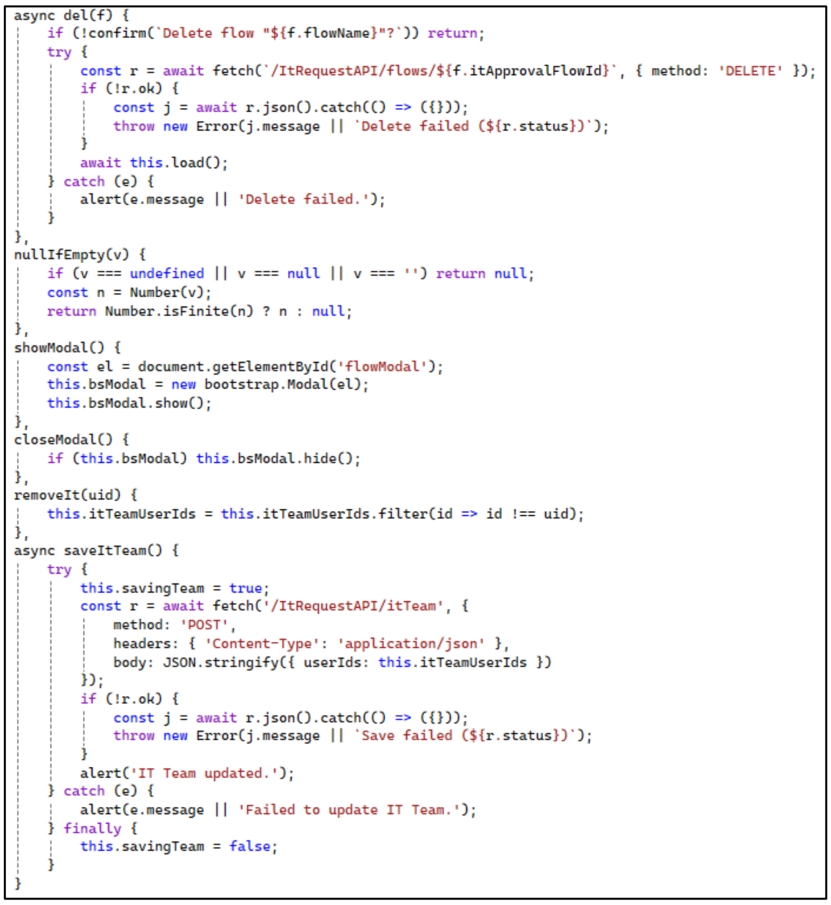
```
methods: {
    async load() {
        try {
            this.busy = true; this.error = null;
            const r = await fetch('/ItRequestAPI/flows');
            if (!r.ok) throw new Error(`Load failed (${r.status})`);
            const data = await r.json();
            this.flows = Array.isArray(data)
                ? data.map(x => ({
                    itApprovalFlowId: x.itApprovalFlowId ?? x.ItApprovalFlowId,
                    flowName: x.flowName ?? x.FlowName,
                    hodUserId: x.hodUserId ?? x.HodUserId,
                    groupItHodUserId: x.groupItHodUserId ?? x.GroupItHodUserId,
                    finHodUserId: x.finHodUserId ?? x.FinHodUserId,
                    mgmtUserId: x.mgmtUserId ?? x.MgmtUserId
                }))
                : [];
        } catch (e) {
            this.error = e.message || 'Failed to load flows.';
        } finally {
            this.busy = false;
        }
    },
    async loadUsers() {
        const res = await fetch('/ItRequestAPI/users');
        this.users = await res.json();
    },
    async loadItTeam() {
        const r = await fetch('/ItRequestAPI/itTeam');
        this.itTeamUserIds = await r.json(); // array<int>
    },
    resolveUserName(id) {
        const u = this.users.find(x => x.id === id);
        return u ? u.name : '(unknown)';
    },
    openCreate() {
        this.formError = null;
        this.form = { itApprovalFlowId: null, flowName: '', hodUserId: null, groupItHodUserId: null, finHodUserId: null, mgmtUserId: null };
        this.showModal();
        this.loadUsers();
    },
```



```
openEdit(f) {
    this.formError = null;
    this.form = JSON.parse(JSON.stringify(f));
    this.showModal();
    this.loadUsers();
},
async save() {
    try {
        if (this.saving) return;
        this.saving = true; this.formError = null;

        if (!this.form.flowName || !this.form.flowName.trim()) {
            this.formError = 'Flow name is required.'; this.saving = false; return;
        }

        const payload = {
            flowName: this.form.flowName.trim(),
            hodUserId: this.nullIfEmpty(this.form.hodUserId),
            groupItHodUserId: this.nullIfEmpty(this.form.groupItHodUserId),
            finHodUserId: this.nullIfEmpty(this.form.finHodUserId),
            mgmtUserId: this.nullIfEmpty(this.form.mgmtUserId)
        };

        let res;
        if (this.form.itApprovalFlowId) {
            res = await fetch(`/ItRequestAPI/flows/${this.form.itApprovalFlowId}`, {
                method: 'PUT',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify(payload)
            });
        } else {
            res = await fetch('/ItRequestAPI/flows', {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify(payload)
            });
        }
        if (!res.ok) {
            const j = await res.json().catch(() => ({}));
            throw new Error(j.message || `Save failed (${res.status})`);
        }
        await this.load();
        this.closeModal();
    } catch (e) {
        this.formError = e.message || 'Unable to save flow.';
    } finally {
        this.saving = false;
    }
},
```



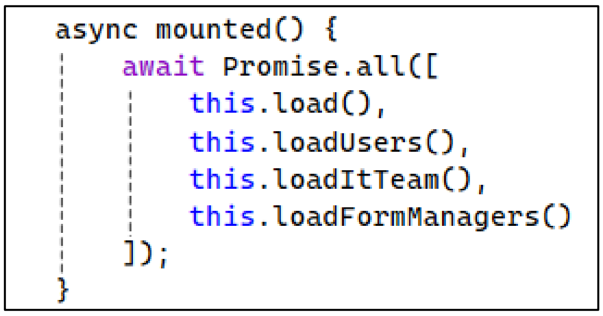
```
/* ===== Request Form Managers endpoints ===== */
async loadFormManagers() {
    const r = await fetch('/ItRequestAPI/formManagers');
    this.rfmUserIds = await r.json(); // array<int>
},
async saveFormManagers() {
    try {
        this.savingManagers = true;
        const r = await fetch('/ItRequestAPI/formManagers', {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({ userIds: this.rfmUserIds })
        });
        if (!r.ok) {
            const j = await r.json().catch(() => ({}));
            throw new Error(j.message || `Save failed (${r.status})`);
        }
        alert('Request Form Managers updated.');
    } catch (e) {
        alert(e.message || 'Failed to update Request Form Managers.');
    } finally {
        this.savingManagers = false;
    }
},
removeRfm(uid) {
    this.rfmUserIds = this.rfmUserIds.filter(id => id !== uid);
}
```



```
async del(f) {
    if (!confirm(`Delete flow "${f.flowName}"?`)) return;
    try {
        const r = await fetch(`/ItRequestAPI/flows/${f.itApprovalFlowId}`, { method: 'DELETE' });
        if (!r.ok) {
            const j = await r.json().catch(() => ({}));
            throw new Error(j.message || `Delete failed (${r.status})`);
        }
        await this.load();
    } catch (e) {
        alert(e.message || 'Delete failed.');
    }
},
nullIfEmpty(v) {
    if (v === undefined || v === null || v === '') return null;
    const n = Number(v);
    return Number.isFinite(n) ? n : null;
},
showModal() {
    const el = document.getElementById('flowModal');
    this.bsModal = new bootstrap.Modal(el);
    this.bsModal.show();
},
closeModal() {
    if (this.bsModal) this.bsModal.hide();
},
removeIt(uid) {
    this.itTeamUserIds = this.itTeamUserIds.filter(id => id !== uid);
},
async saveItTeam() {
    try {
        this.savingTeam = true;
        const r = await fetch('/ItRequestAPI/itTeam', {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({ userIds: this.itTeamUserIds })
        });
        if (!r.ok) {
            const j = await r.json().catch(() => ({}));
            throw new Error(j.message || `Save failed (${r.status})`);
        }
        alert('IT Team updated.');
    } catch (e) {
        alert(e.message || 'Failed to update IT Team.');
    } finally {
        this.savingTeam = false;
    }
}
```

✓ **mounted**

- **mounted()**: On first render it runs load, loadUsers, loadFormManagers, and loadItTeam in parallel. This fills the flows table, the user dropdowns, and the IT Team selection right away so the page is usable without extra clicks.



```
async mounted() {
    await Promise.all([
        this.load(),
        this.loadUsers(),
        this.loadItTeam(),
        this.loadFormManagers()
    ]);
}
```

- ITRequestAPI (Admin.cshtml)

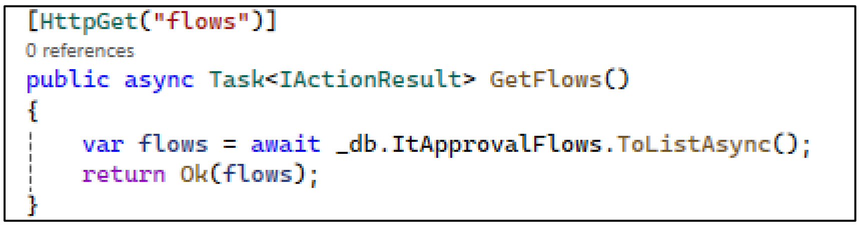
[HttpGet("flows")] (Used by: Admin.cshtml)

- **Purpose:**
  Load every approval flow so the Admin page can list, preview, and pick one to edit or delete. This is the source for your table/grid and any "Edit Flow" modal.
- **Key functionalities:**
  - Reads all rows from ItApprovalFlows without filters. This lets the UI show an up-to-date list every time the page loads or after a change.
  - Returns the raw entity list so you can display FlowName plus the assigned approver IDs. The UI can later resolve those user IDs to names using the users endpoint.

[HttpPost("flows")] (Used by: Admin.cshtml, Create Flow)

- **Purpose:**
  Create a brand new approval flow record. This is typically called from a "New Flow" dialog after the admin enters a name and (optionally) picks approvers.
- **Key functionalities:**

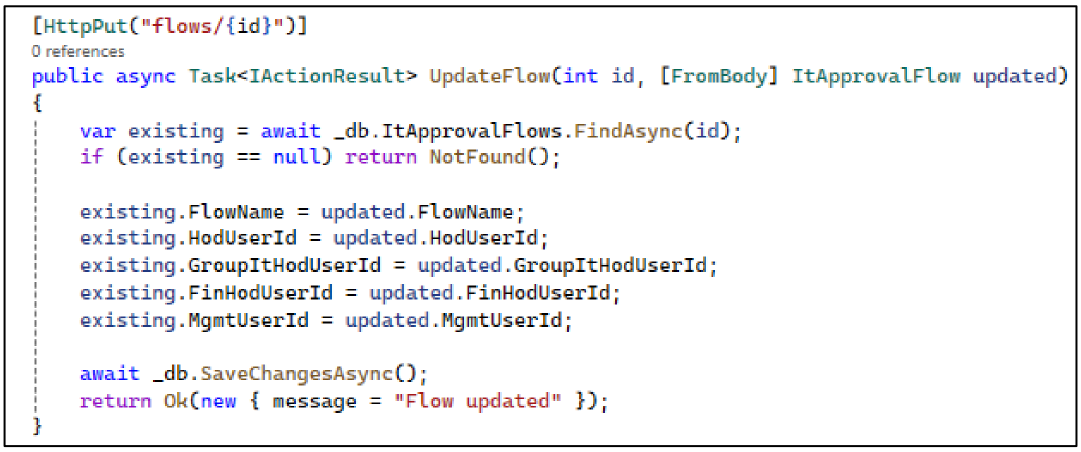
  - Validates that FlowName is present before saving. This avoids placeholder or empty flows that would confuse approvers later.
  - Inserts a new ItApprovalFlow row and persists it immediately. The response returns a simple success message so you can refresh the grid and close the dialog.



```
[HttpGet("flows")]
0 references
public async Task<IActionResult> GetFlows()
{
    var flows = await _db.ItApprovalFlows.ToListAsync();
    return Ok(flows);
}
```

[HttpPut("flows/{id}")] (Used by: Admin.cshtml, Edit Flow)

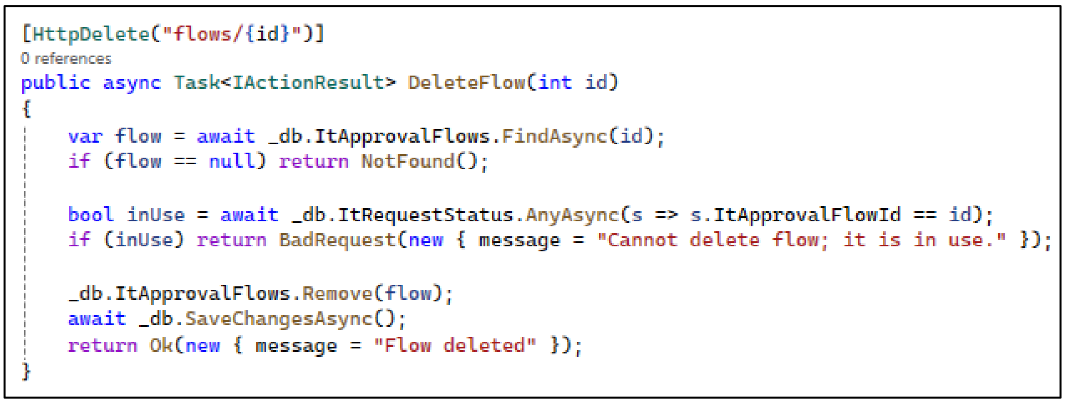
- **Purpose:**
  Update an existing approval flow's name and which users are assigned to each approval role. This is what your "Save changes" button in the flow editor should call.
- **Key functionalities:**
  - Looks up the target flow by id and fails fast if it doesn't exist. That prevents the UI from silently "saving" to a non-existent record.
  - Copies over FlowName and the four role assignments (HodUserId, GroupItHodUserId, FinHodUserId, MgmtUserId) then saves. This guarantees the new approver lineup applies to all new requests that use this flow.



```
[HttpPut("flows/{id}")]
0 references
public async Task<IActionResult> UpdateFlow(int id, [FromBody] ItApprovalFlow updated)
{
    var existing = await _db.ItApprovalFlows.FindAsync(id);
    if (existing == null) return NotFound();

    existing.FlowName = updated.FlowName;
    existing.HodUserId = updated.HodUserId;
    existing.GroupItHodUserId = updated.GroupItHodUserId;
    existing.FinHodUserId = updated.FinHodUserId;
    existing.MgmtUserId = updated.MgmtUserId;

    await _db.SaveChangesAsync();
    return Ok(new { message = "Flow updated" });
}
```

[HttpDelete("flows/{id}")] (Used by: Admin.cshtml, Delete Flow)

- **Purpose:**
  Remove an approval flow that's no longer needed. Useful when cleaning up test flows or deprecated configurations.
- **Key functionalities:**
  - Fetches the flow by id and returns NotFound if it's already gone. That keeps the UI consistent if two admins act at the same time.
  - Checks if any ItRequestStatus rows point to this flow; if yes, deletion is blocked with a clear message. This protects history and prevents orphaned status records.

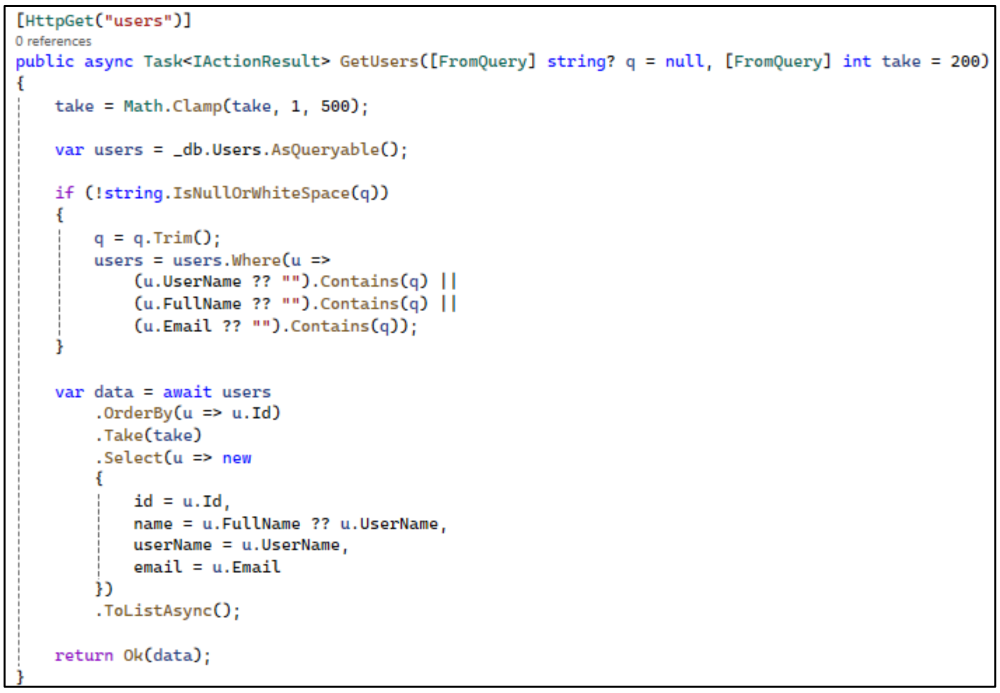


```
[HttpDelete("flows/{id}")]
0 references
public async Task<IActionResult> DeleteFlow(int id)
{
    var flow = await _db.ItApprovalFlows.FindAsync(id);
    if (flow == null) return NotFound();

    bool inUse = await _db.ItRequestStatus.AnyAsync(s => s.ItApprovalFlowId == id);
    if (inUse) return BadRequest(new { message = "Cannot delete flow; it is in use." });

    _db.ItApprovalFlows.Remove(flow);
    await _db.SaveChangesAsync();
    return Ok(new { message = "Flow deleted" });
}
```

[HttpGet("users")] (Used by: Admin.cshtml, approver pickers and IT team pickers)

- **Purpose:**
  Provide a lightweight user lookup for dropdowns and autocomplete. Admins use this to assign approvers and to populate the IT Team list with real names.
- **Key functionalities:**
  - Supports an optional q filter that searches UserName, FullName, and Email. That means you can type a name fragment or email prefix and still get relevant matches.
  - Limits the result set with take (1–500, default 200) and orders by Id. This keeps the UI snappy and avoids sending giant payloads to the browser.

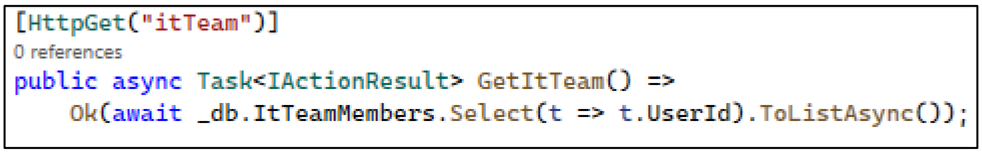
- Projects to a compact shape { id, name, userName, email }. The name field prefers FullName and falls back to UserName so your dropdowns look clean even when profiles are incomplete.



```
[HttpGet("users")]
0 references
public async Task<IActionResult> GetUsers([FromQuery] string? q = null, [FromQuery] int take = 200)
{
    take = Math.Clamp(take, 1, 500);

    var users = _db.Users.AsQueryable();

    if (!string.IsNullOrWhiteSpace(q))
    {
        q = q.Trim();
        users = users.Where(u =>
            (u.UserName ?? "").Contains(q) ||
            (u.FullName ?? "").Contains(q) ||
            (u.Email ?? "").Contains(q));
    }

    var data = await users
        .OrderBy(u => u.Id)
        .Take(take)
        .Select(u => new
        {
            id = u.Id,
            name = u.FullName ?? u.UserName,
            userName = u.UserName,
            email = u.Email
        })
        .ToListAsync();

    return Ok(data);
}
```

[HttpGet("itTeam")] (Used by: Admin.cshtml, load IT team)

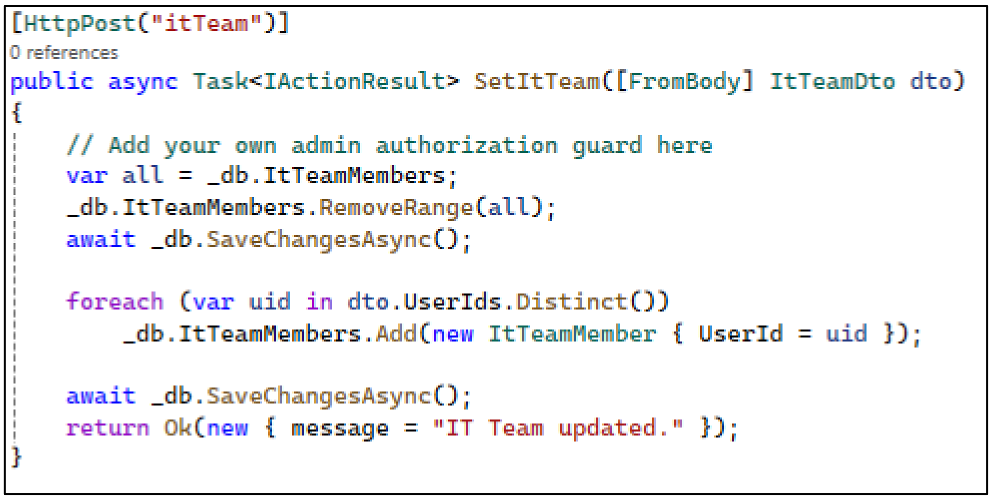
- **Purpose:**
  Return the current IT team membership so the page can show who's on the roster. The Admin page can then pre-select these users in a multi-select control.
- **Key functionalities:**
  - Reads all ItTeamMembers and returns just the UserId list. This keeps the endpoint fast and simple.
  - The UI can hydrate display names by calling users with a search or by mapping from a cached user list. That way you don't duplicate name logic here.



```
[HttpGet("itTeam")]
0 references
public async Task<IActionResult> GetItTeam() =>
    Ok(await _db.ItTeamMembers.Select(t => t.UserId).ToListAsync());
```

[HttpPost("itTeam")] (Used by: Admin.cshtml, save IT team)

- **Purpose:**
  Replace the IT team with a new set of user ids in one go. This is useful for bulk changes after reorganizations.
- **Key functionalities:**
  - Clears all existing ItTeamMembers rows first, commits, then inserts the distinct ids from the payload. Doing it in two steps ensures there's no partial overlap or duplicate rows.
  - Returns a simple confirmation so the UI can show a toast and refresh the list. You should add an admin-only authorization guard here to prevent accidental edits by regular users.

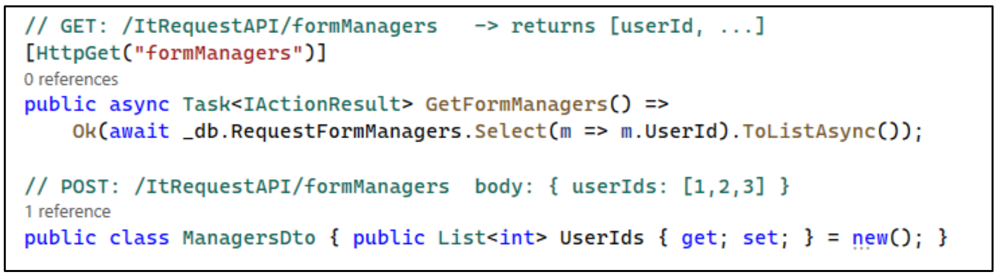


```
[HttpPost("itTeam")]
0 references
public async Task<IActionResult> SetItTeam([FromBody] ItTeamDto dto)
{
    // Add your own admin authorization guard here
    var all = _db.ItTeamMembers;
    _db.ItTeamMembers.RemoveRange(all);
    await _db.SaveChangesAsync();

    foreach (var uid in dto.UserIds.Distinct())
        _db.ItTeamMembers.Add(new ItTeamMember { UserId = uid });

    await _db.SaveChangesAsync();
    return Ok(new { message = "IT Team updated." });
}
```

[HttpGet("formManagers")] (Used by: Admin.cshtml, load RequestFormManagers)

- **Purpose:**
  Return the current IT team membership so the page can show who's on the roster. The Admin page can then pre-select these users in a multi-select control.
- **Key functionalities:**
  - Reads all RequestFormManagers and returns just the UserId list. This keeps the endpoint fast and simple.

- The UI can hydrate display names by calling users with a search or by mapping from a cached user list. That way you don't duplicate name logic here.

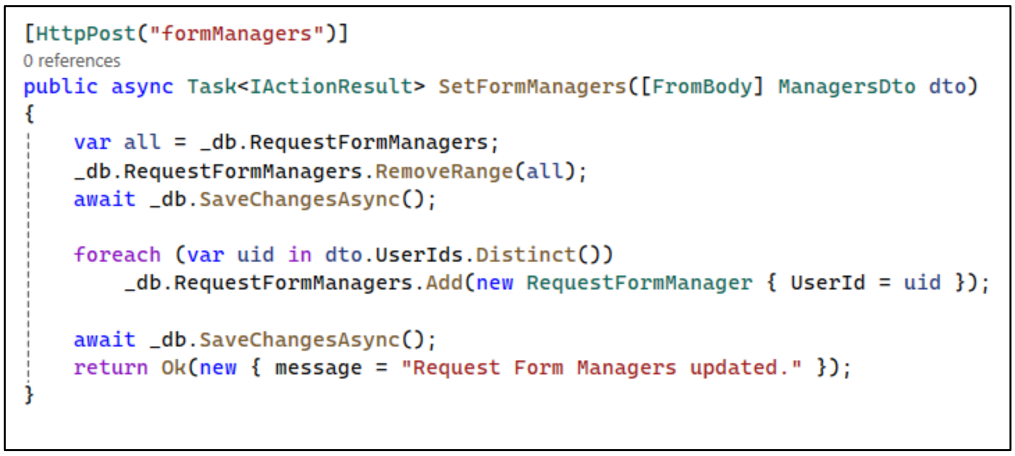


```
// GET: /ItRequestAPI/formManagers   -> returns [userId, ...]
[HttpGet("formManagers")]
0 references
public async Task<IActionResult> GetFormManagers() =>
    Ok(await _db.RequestFormManagers.Select(m => m.UserId).ToListAsync());

// POST: /ItRequestAPI/formManagers  body: { userIds: [1,2,3] }
1 reference
public class ManagersDto { public List<int> UserIds { get; set; } = new(); }
```

[HttpPost("itTeam")] (Used by: Admin.cshtml, save Request Form Manager)

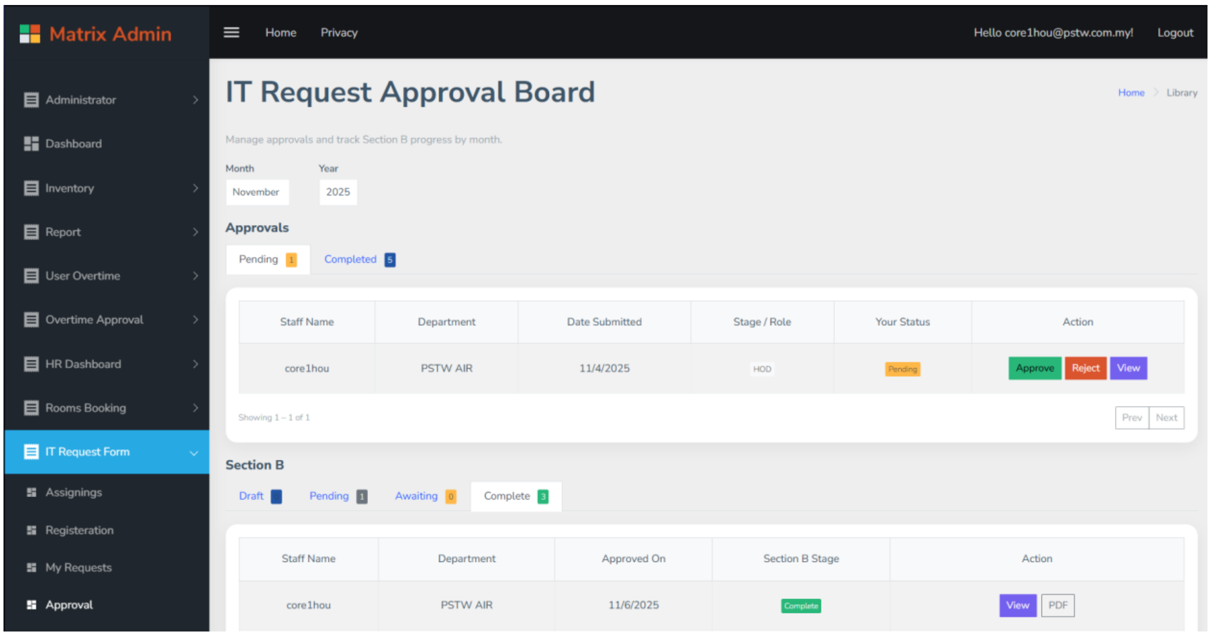
- **Purpose:**
  Replace the Request Form Manager with a new set of user ids in one go. This is useful for bulk changes after reorganizations.

- **Key functionalities:**
  - Clears all existing Request Form Managers rows first, commits, then inserts the distinct ids from the payload. Doing it in two steps ensures there's no partial overlap or duplicate rows.
  - Returns a simple confirmation so the UI can show a toast and refresh the list. You should add an admin-only authorization guard here to prevent accidental edits by regular users.



```
[HttpPost("formManagers")]
0 references
public async Task<IActionResult> SetFormManagers([FromBody] ManagersDto dto)
{
    var all = _db.RequestFormManagers;
    _db.RequestFormManagers.RemoveRange(all);
    await _db.SaveChangesAsync();

    foreach (var uid in dto.UserIds.Distinct())
        _db.RequestFormManagers.Add(new RequestFormManager { UserId = uid });

    await _db.SaveChangesAsync();
    return Ok(new { message = "Request Form Managers updated." });
}
```

- Approval.cshtml



IT Request Approval Board

Manage approvals and track Section B progress by month.

Month: November
Year: 2025

Approvals

Pending 1 | Completed 5

Staff Name	Department	Date Submitted	Stage / Role	Your Status	Action
core1hou	PSTW AIR	11/4/2025	HOD	Pending	Approve Reject View

Showing 1 – 1 of 1 Prev Next

Section B

Draft | Pending 1 | Awaiting 0 | Complete 3

Staff Name	Department	Approved On	Section B Stage	Action
core1hou	PSTW AIR	11/6/2025	Complete	View PDF

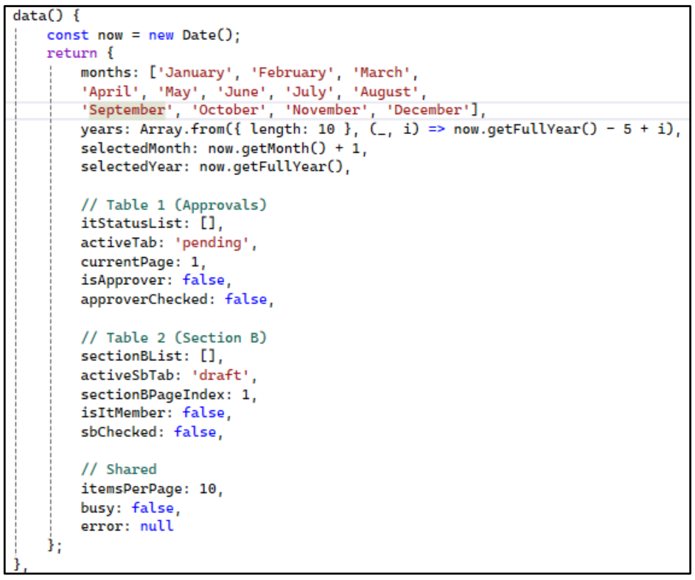
This file (Approval.cshtml) represents the web page where approvers (like HoD, IT HoD, Finance, Management) can view and manage pending overtime requests.

- ✓ <div id="app">...</div>
  - **Purpose**: This div acts as the **main container** for your Vue.js application. All the HTML elements and components inside this div will be managed by Vue.js. The id="app" matches the app.mount('#app') line in the script, linking the Vue instance to this part of the page.

- ✓ const app = Vue.createApp({...})
  - **Purpose**: This line **initializes your Vue.js application**. It creates a new Vue application instance and defines its core properties: data, computed, mounted and methods.

- ✓ data() function
  - **months, years:** Arrays for dropdown options.
  - **selectedMonth, selectedYear**: Current selected month/year, initialized from sessionStorage (to remember user's last selection) or current date.
  - **itStatusList**: An array that will hold the list of IT request statuses fetched from the API.

- **activeTab**: Controls which tab ("Pending Actions" or "Completed Actions") is currently active.
- **currentPage, itemsPerPage**: Used for pagination (how many items to show per page).
- **isApprover:** gatekeeping flag (UI/queries adapt if the user can approve).
- **approverChecked:** ensures we only run the “am I an approver?” check once before drawing privileged UI.
- **sectionBList:** the fetched dataset for Section B rows.
- **activeSbTab:** which Section B tab is active ('draft' by default).
- **sectionBPageIndex:** page index for Section B pagination.
- **isItMember**: capability flag for IT-only actions.
- **sbChecked**: guard so the IT-membership check doesn’t re-run unnecessarily.
- **busy:** global loading lock to prevent double-clicks/race conditions.
- **error:** a single place to capture and render API/logic errors.

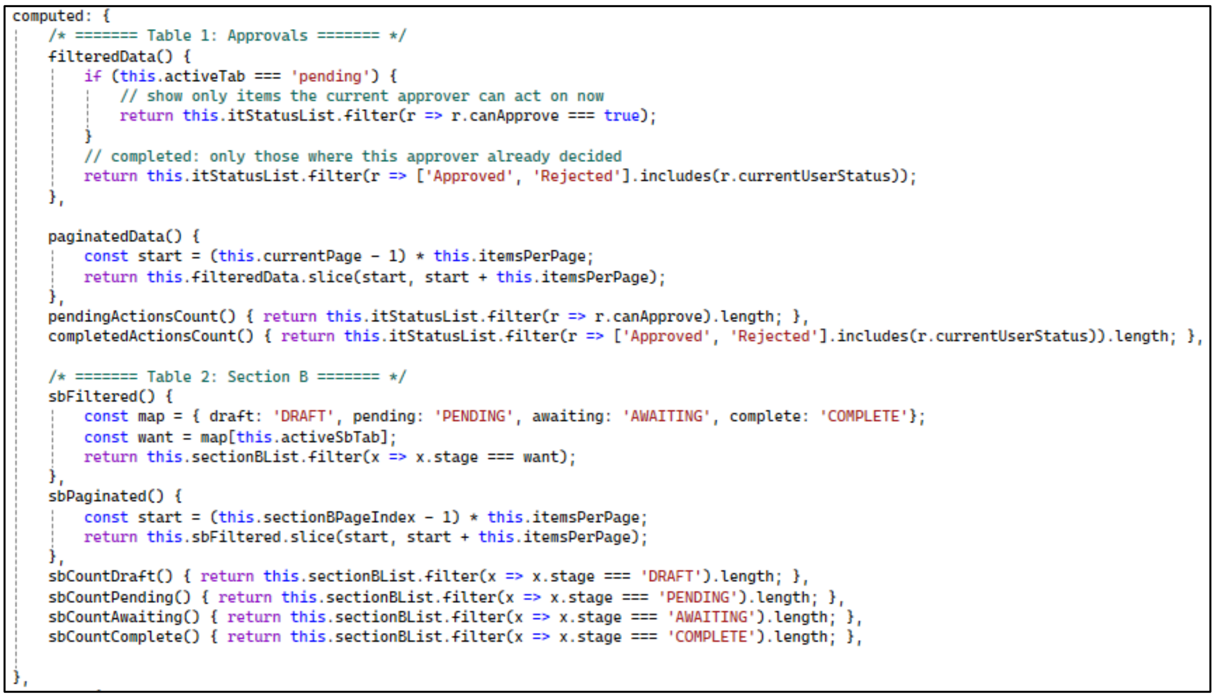


```
data() {
    const now = new Date();
    return {
        months: ['January', 'February', 'March',
        'April', 'May', 'June', 'July', 'August',
        'September', 'October', 'November', 'December'],
        years: Array.from({ length: 10 }, (_, i) => now.getFullYear() - 5 + i),
        selectedMonth: now.getMonth() + 1,
        selectedYear: now.getFullYear(),

        // Table 1 (Approvals)
        itStatusList: [],
        activeTab: 'pending',
        currentPage: 1,
        isApprover: false,
        approverChecked: false,

        // Table 2 (Section B)
        sectionBList: [],
        activeSbTab: 'draft',
        sectionBPageIndex: 1,
        isItMember: false,
        sbChecked: false,

        // Shared
        itemsPerPage: 10,
        busy: false,
        error: null
    };
},
```

✓ computed property

- Shapes the raw datasets into exactly-what-the-UI-needs slices: filtered views per active tab, page-sized subsets for tables, and quick counters for badges.
- **filteredData():** If activeTab === 'pending', return only rows the current user can act on now (r.canApprove === true). Otherwise (completed tab), return rows where this user has already made a decision (currentUserStatus ∈ {Approved, Rejected}).

- **paginatedData():** takes filteredData and returns just the current page using currentPage and itemsPerPage.
- **pendingActionsCount / completedActionsCount:** cheap counters for tab badges, computed directly from itStatusList.
- **sbFiltered():** maps the friendly tab keys (draft|pending|awaiting|complete) to canonical stage values (DRAFT|PENDING|AWAITING|COMPLETE), then filters sectionBList.
- **sbPaginated():** page-slices sbFiltered using sectionBPageIndex.
- **sbCountDraft/Pending/Awaiting/Complete**: per-stage counters for Section B tab badges.



```
computed: {
    /* ======= Table 1: Approvals ======= */
    filteredData() {
        if (this.activeTab === 'pending') {
            // show only items the current approver can act on now
            return this.itStatusList.filter(r => r.canApprove === true);
        }
        // completed: only those where this approver already decided
        return this.itStatusList.filter(r => ['Approved', 'Rejected'].includes(r.currentUserStatus));
    },

    paginatedData() {
        const start = (this.currentPage - 1) * this.itemsPerPage;
        return this.filteredData.slice(start, start + this.itemsPerPage);
    },
    pendingActionsCount() { return this.itStatusList.filter(r => r.canApprove).length; },
    completedActionsCount() { return this.itStatusList.filter(r => ['Approved', 'Rejected'].includes(r.currentUserStatus)).length; },

    /* ======= Table 2: Section B ======= */
    sbFiltered() {
        const map = { draft: 'DRAFT', pending: 'PENDING', awaiting: 'AWAITING', complete: 'COMPLETE'};
        const want = map[this.activeSbTab];
        return this.sectionBList.filter(x => x.stage === want);
    },
    sbPaginated() {
        const start = (this.sectionBPageIndex - 1) * this.itemsPerPage;
        return this.sbFiltered.slice(start, start + this.itemsPerPage);
    },
    sbCountDraft() { return this.sectionBList.filter(x => x.stage === 'DRAFT').length; },
    sbCountPending() { return this.sectionBList.filter(x => x.stage === 'PENDING').length; },
    sbCountAwaiting() { return this.sectionBList.filter(x => x.stage === 'AWAITING').length; },
    sbCountComplete() { return this.sectionBList.filter(x => x.stage === 'COMPLETE').length; },

},
```

✓ methods property

- **formatDate(str)**

  What: Safe date pretty-printer for table cells.

  How: Tries new Date(str) and falls back to the raw string if parsing fails; renders with toLocaleDateString().

  Why: Keeps the template clean and resilient to nulls or odd server formats.
- **getStatusBadgeClass(status)**

  What: Maps approval status to Bootstrap badge classes.

  How: Case-insensitive switch to bg-success/bg-danger/bg-warning text-dark/bg-secondary.

Why: Centralizes styling so templates don't hardcode CSS.

- **stageBadge(stage)**

  What: Normalizes Section B stage labels and classes for chips.

  How: switch on canonical stages to return { text, cls } pairs.

  Why: Human-friendly text plus consistent color coding in one place.

- **onPeriodChange()**

  What: Refreshes both tables when month/year changes.

  How: Calls loadApprovals() and loadSectionB() back-to-back.

  Why: A single hook keeps period changes predictable across sections.

- **switchTab(tab) / switchSbTab(tab)**

  What: Activates a tab and resets its pagination.

  How: Sets activeTab or activeSbTab and resets page index to 1.

  Why: Prevents "empty page" surprises when a new filter has fewer rows.

- **loadApprovals()**

  What: Fetches Approvals data for the selected period and resolves approver capability.

- **loadSectionB()**

  What: Fetches Section B list for the selected period and resolves IT-member capability.

- **updateStatus(statusId, decision)**

  What: Approver action to Approve or Reject a request.

- **acceptIt(statusId)**

  What: IT's acceptance for Section B.

  How: POST /ItRequestAPI/sectionB/accept with { by: 'IT' }, then loadSectionB().

  Why: Keeps role intent explicit in payload, even if the server checks it anyway.

- **acceptRequestor(statusId)**

  What: Requestor's acceptance for Section B.

  How: Same endpoint with { by: 'REQUESTOR' }, then loadSectionB().

  Why: Mirrors IT acceptance to keep flow symmetric.

- **viewRequest(statusId)**

  What: Deep-links to the Request Review page.

How: window.location.href to /IT/ApprovalDashboard/RequestReview?....

Why: Full page nav for a detail workflow.

- **openSectionB(statusId) / openSectionBEdit(statusId)**

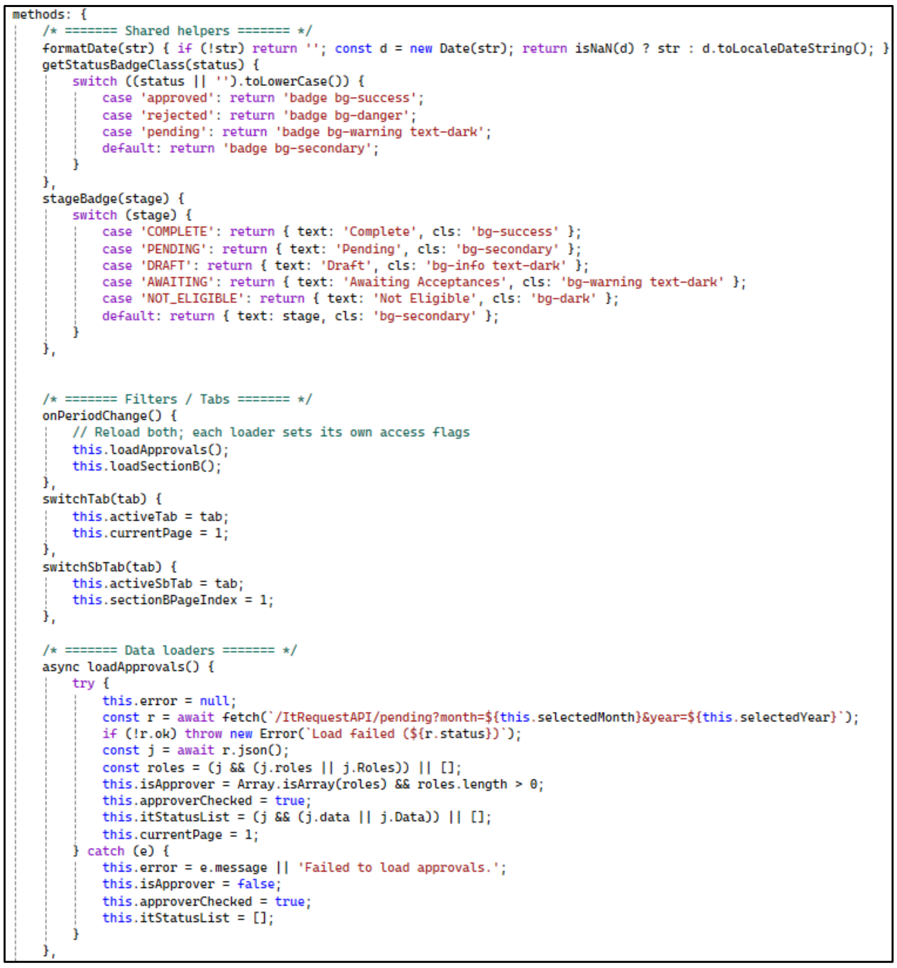
  What: Opens Section B (read or edit) and preserves return location.

- **downloadPdf(statusId)**

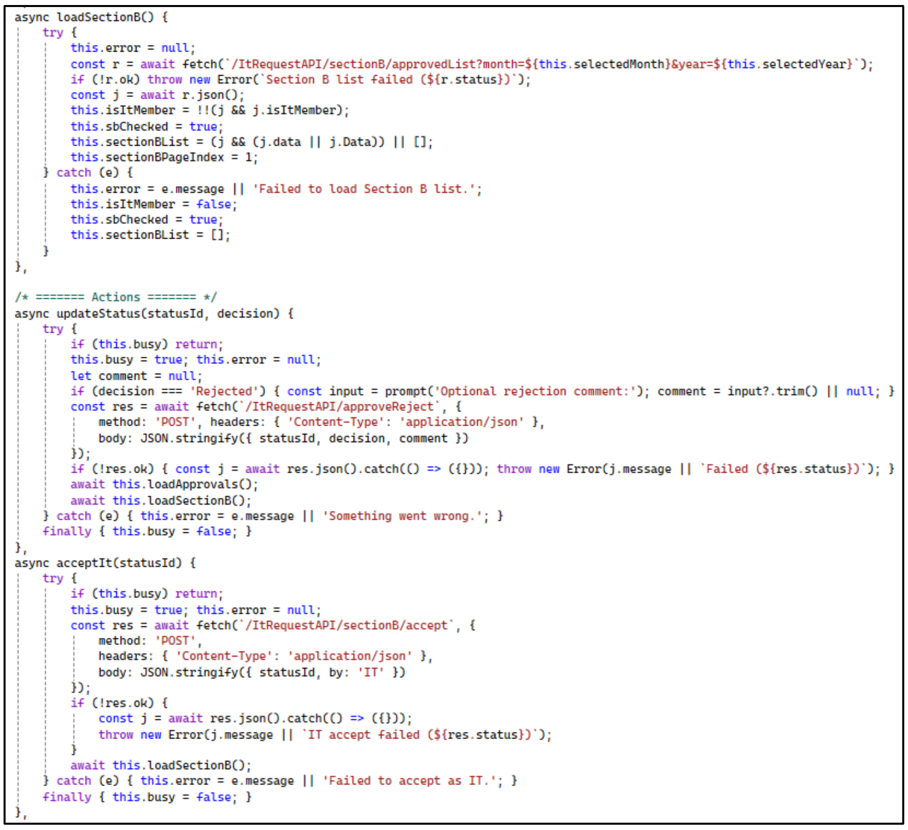
  What: Opens the Section B PDF in a new tab.

  How: window.open('/ItRequestAPI/sectionB/pdf?...', '_blank').

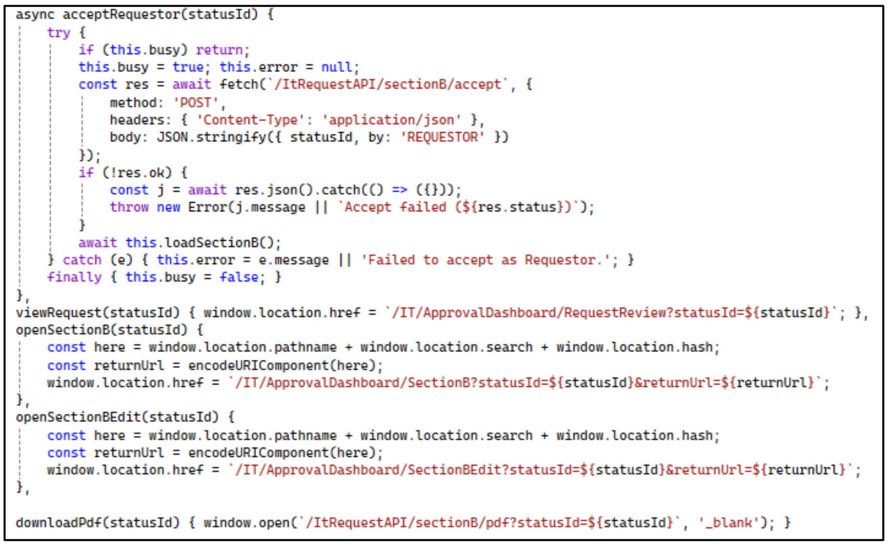
  Why: Lets the user preview/download via the browser's built-in PDF viewer.



```
methods: {
    /* ======= Shared helpers ======= */
    formatDate(str) { if (!str) return ''; const d = new Date(str); return isNaN(d) ? str : d.toLocaleDateString(); }
    getStatusBadgeClass(status) {
        switch ((status || '').toLowerCase()) {
            case 'approved': return 'badge bg-success';
            case 'rejected': return 'badge bg-danger';
            case 'pending': return 'badge bg-warning text-dark';
            default: return 'badge bg-secondary';
        }
    },
    stageBadge(stage) {
        switch (stage) {
            case 'COMPLETE': return { text: 'Complete', cls: 'bg-success' };
            case 'PENDING': return { text: 'Pending', cls: 'bg-secondary' };
            case 'DRAFT': return { text: 'Draft', cls: 'bg-info text-dark' };
            case 'AWAITING': return { text: 'Awaiting Acceptances', cls: 'bg-warning text-dark' };
            case 'NOT_ELIGIBLE': return { text: 'Not Eligible', cls: 'bg-dark' };
            default: return { text: stage, cls: 'bg-secondary' };
        }
    },


    /* ======= Filters / Tabs ======= */
    onPeriodChange() {
        // Reload both; each loader sets its own access flags
        this.loadApprovals();
        this.loadSectionB();
    },
    switchTab(tab) {
        this.activeTab = tab;
        this.currentPage = 1;
    },
    switchSbTab(tab) {
        this.activeSbTab = tab;
        this.sectionBPageIndex = 1;
    },

    /* ======= Data loaders ======= */
    async loadApprovals() {
        try {
            this.error = null;
            const r = await fetch(`/ItRequestAPI/pending?month=${this.selectedMonth}&year=${this.selectedYear}`);
            if (!r.ok) throw new Error(`Load failed (${r.status})`);
            const j = await r.json();
            const roles = (j && (j.roles || j.Roles)) || [];
            this.isApprover = Array.isArray(roles) && roles.length > 0;
            this.approverChecked = true;
            this.itStatusList = (j && (j.data || j.Data)) || [];
            this.currentPage = 1;
        } catch (e) {
            this.error = e.message || 'Failed to load approvals.';
            this.isApprover = false;
            this.approverChecked = true;
            this.itStatusList = [];
        }
    },
```



```
async loadSectionB() {
    try {
        this.error = null;
        const r = await fetch(`/ItRequestAPI/sectionB/approvedList?month=${this.selectedMonth}&year=${this.selectedYear}`);
        if (!r.ok) throw new Error(`Section B list failed (${r.status})`);
        const j = await r.json();
        this.isItMember = !!(j && j.isItMember);
        this.sbChecked = true;
        this.sectionBList = (j && (j.data || j.Data)) || [];
        this.sectionBPageIndex = 1;
    } catch (e) {
        this.error = e.message || 'Failed to load Section B list.';
        this.isItMember = false;
        this.sbChecked = true;
        this.sectionBList = [];
    }
},

/* ======= Actions ======= */
async updateStatus(statusId, decision) {
    try {
        if (this.busy) return;
        this.busy = true; this.error = null;
        let comment = null;
        if (decision === 'Rejected') { const input = prompt('Optional rejection comment:'); comment = input?.trim() || null; }
        const res = await fetch(`/ItRequestAPI/approveReject`, {
            method: 'POST', headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({ statusId, decision, comment })
        });
        if (!res.ok) { const j = await res.json().catch(() => ({})); throw new Error(j.message || `Failed (${res.status})`); }
        await this.loadApprovals();
        await this.loadSectionB();
    } catch (e) { this.error = e.message || 'Something went wrong.'; }
    finally { this.busy = false; }
},
async acceptIt(statusId) {
    try {
        if (this.busy) return;
        this.busy = true; this.error = null;
        const res = await fetch(`/ItRequestAPI/sectionB/accept`, {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({ statusId, by: 'IT' })
        });
        if (!res.ok) {
            const j = await res.json().catch(() => ({}));
            throw new Error(j.message || `IT accept failed (${res.status})`);
        }
        await this.loadSectionB();
    } catch (e) { this.error = e.message || 'Failed to accept as IT.'; }
    finally { this.busy = false; }
},
```

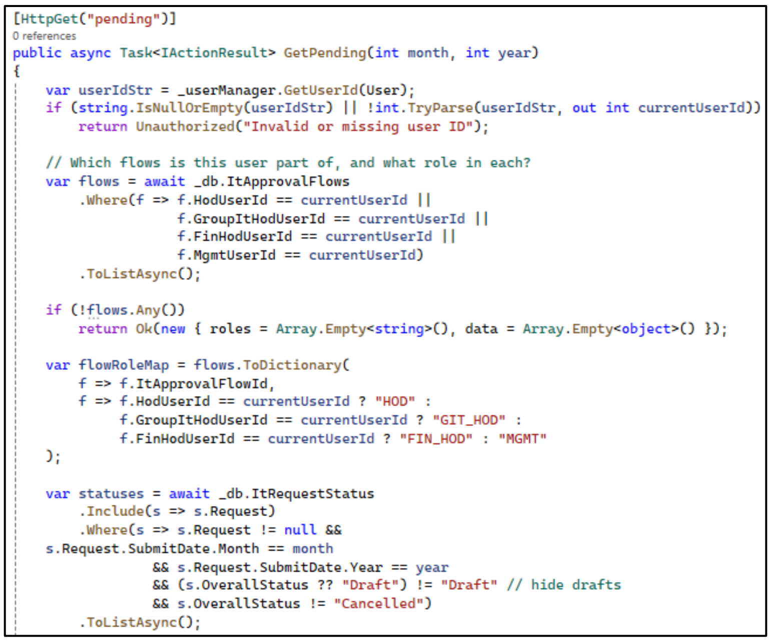


```
async acceptRequestor(statusId) {
    try {
        if (this.busy) return;
        this.busy = true; this.error = null;
        const res = await fetch(`/ItRequestAPI/sectionB/accept`, {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({ statusId, by: 'REQUESTOR' })
        });
        if (!res.ok) {
            const j = await res.json().catch(() => ({}));
            throw new Error(j.message || `Accept failed (${res.status})`);
        }
        await this.loadSectionB();
    } catch (e) { this.error = e.message || 'Failed to accept as Requestor.'; }
    finally { this.busy = false; }
},
viewRequest(statusId) { window.location.href = `/IT/ApprovalDashboard/RequestReview?statusId=${statusId}`; },
openSectionB(statusId) {
    const here = window.location.pathname + window.location.search + window.location.hash;
    const returnUrl = encodeURIComponent(here);
    window.location.href = `/IT/ApprovalDashboard/SectionB?statusId=${statusId}&returnUrl=${returnUrl}`;
},
openSectionBEdit(statusId) {
    const here = window.location.pathname + window.location.search + window.location.hash;
    const returnUrl = encodeURIComponent(here);
    window.location.href = `/IT/ApprovalDashboard/SectionBEdit?statusId=${statusId}&returnUrl=${returnUrl}`;
},

downloadPdf(statusId) { window.open(`/ItRequestAPI/sectionB/pdf?statusId=${statusId}`, '_blank'); }
```

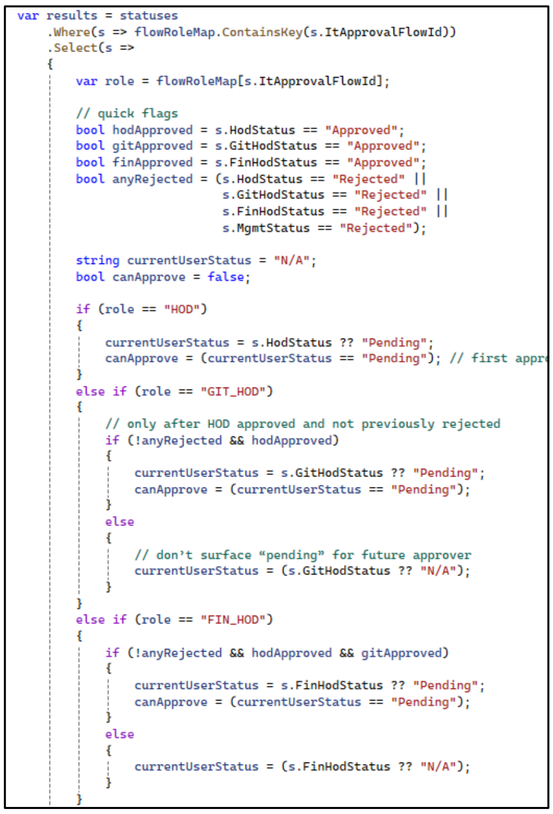
- ITRequestAPI (Approval.cshtml)
  - **[HttpGet("pending")] — Used by:** Approval.cshtml

- **Purpose:** Returns the IT requests (for a given month/year) that the logged-in user is involved with, plus whether they can act on each item now (approve/reject) and what their own step status is.

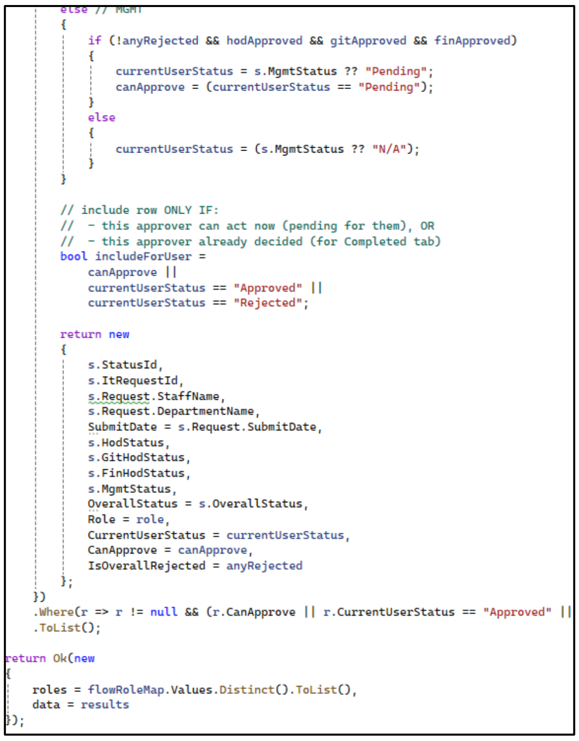
- **Key functionality:**
  - **User identification:** Gets currentUserId from ASP.NET Identity; 401 if missing/invalid.
  - **Find approver roles:** Looks up ItApprovalFlows where the user is HOD / GIT_HOD / FIN_HOD / MGMT, then builds a map { ItApprovalFlowId -> role }.
  - **Filter requests:** Pulls ItRequestStatus (with Request included) for the selected month & year, excluding Draft and Cancelled.
  - **Permission / stage logic:** For each status:
    - Detects prior approvals (hodApproved, gitApproved, finApproved) and if **any** step has rejected.
    - Calculates currentUserStatus for this user's role.
      - HOD sees Pending immediately.
      - GIT_HOD only sees Pending after HOD approved and no rejections.
      - FIN_HOD only after HOD & GIT_HOD approved and no rejections.
      - MGMT only after HOD, GIT_HOD, FIN_HOD approved and no rejections.
    - Sets canApprove = (currentUserStatus == "Pending") when it's their turn.
    - Keeps the row if the user can act **now** or if they already acted (Approved/Rejected) for Completed tabs.
- **Inputs (query):** month (int), year (int).



```
[HttpGet("pending")]
0 references
public async Task<IActionResult> GetPending(int month, int year)
{
    var userIdStr = _userManager.GetUserId(User);
    if (string.IsNullOrEmpty(userIdStr) || !int.TryParse(userIdStr, out int currentUserId))
        return Unauthorized("Invalid or missing user ID");

    // Which flows is this user part of, and what role in each?
    var flows = await _db.ItApprovalFlows
        .Where(f => f.HodUserId == currentUserId ||
                    f.GroupItHodUserId == currentUserId ||
                    f.FinHodUserId == currentUserId ||
                    f.MgmtUserId == currentUserId)
        .ToListAsync();

    if (!flows.Any())
        return Ok(new { roles = Array.Empty<string>(), data = Array.Empty<object>() });

    var flowRoleMap = flows.ToDictionary(
        f => f.ItApprovalFlowId,
        f => f.HodUserId == currentUserId ? "HOD" :
             f.GroupItHodUserId == currentUserId ? "GIT_HOD" :
             f.FinHodUserId == currentUserId ? "FIN_HOD" : "MGMT"
    );

    var statuses = await _db.ItRequestStatus
        .Include(s => s.Request)
        .Where(s => s.Request != null &&
    s.Request.SubmitDate.Month == month
                 && s.Request.SubmitDate.Year == year
                 && (s.OverallStatus ?? "Draft") != "Draft" // hide drafts
                 && s.OverallStatus != "Cancelled")
        .ToListAsync();
```



```
var results = statuses
    .Where(s => flowRoleMap.ContainsKey(s.ItApprovalFlowId))
    .Select(s =>
    {
        var role = flowRoleMap[s.ItApprovalFlowId];

        // quick flags
        bool hodApproved = s.HodStatus == "Approved";
        bool gitApproved = s.GitHodStatus == "Approved";
        bool finApproved = s.FinHodStatus == "Approved";
        bool anyRejected = (s.HodStatus == "Rejected" ||
                            s.GitHodStatus == "Rejected" ||
                            s.FinHodStatus == "Rejected" ||
                            s.MgmtStatus == "Rejected");

        string currentUserStatus = "N/A";
        bool canApprove = false;

        if (role == "HOD")
        {
            currentUserStatus = s.HodStatus ?? "Pending";
            canApprove = (currentUserStatus == "Pending"); // first appr
        }
        else if (role == "GIT_HOD")
        {
            // only after HOD approved and not previously rejected
            if (!anyRejected && hodApproved)
            {
                currentUserStatus = s.GitHodStatus ?? "Pending";
                canApprove = (currentUserStatus == "Pending");
            }
            else
            {
                // don’t surface “pending” for future approver
                currentUserStatus = (s.GitHodStatus ?? "N/A");
            }
        }
        else if (role == "FIN_HOD")
        {
            if (!anyRejected && hodApproved && gitApproved)
            {
                currentUserStatus = s.FinHodStatus ?? "Pending";
                canApprove = (currentUserStatus == "Pending");
            }
            else
            {
                currentUserStatus = (s.FinHodStatus ?? "N/A");
            }
        }
```



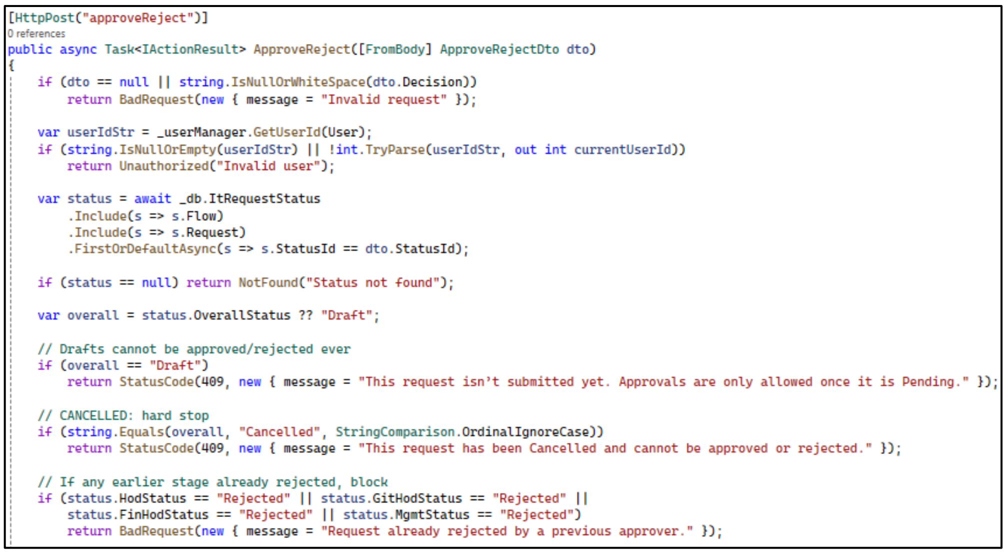
```
        else // MGMT
        {
            if (!anyRejected && hodApproved && gitApproved && finApproved)
            {
                currentUserStatus = s.MgmtStatus ?? "Pending";
                canApprove = (currentUserStatus == "Pending");
            }
            else
            {
                currentUserStatus = (s.MgmtStatus ?? "N/A");
            }
        }

        // include row ONLY IF:
        //  - this approver can act now (pending for them), OR
        //  - this approver already decided (for Completed tab)
        bool includeForUser =
            canApprove ||
            currentUserStatus == "Approved" ||
            currentUserStatus == "Rejected";

        return new
        {
            s.StatusId,
            s.ItRequestId,
            s.Request.StaffName,
            s.Request.DepartmentName,
            SubmitDate = s.Request.SubmitDate,
            s.HodStatus,
            s.GitHodStatus,
            s.FinHodStatus,
            s.MgmtStatus,
            OverallStatus = s.OverallStatus,
            Role = role,
            CurrentUserStatus = currentUserStatus,
            CanApprove = canApprove,
            IsOverallRejected = anyRejected
        };
    })
    .Where(r => r != null && (r.CanApprove || r.CurrentUserStatus == "Approved" ||
    .ToList();

return Ok(new
{
    roles = flowRoleMap.Values.Distinct().ToList(),
    data = results
});
```

- **[HttpPost("approveReject")] — Used by:** Approval.cshtml

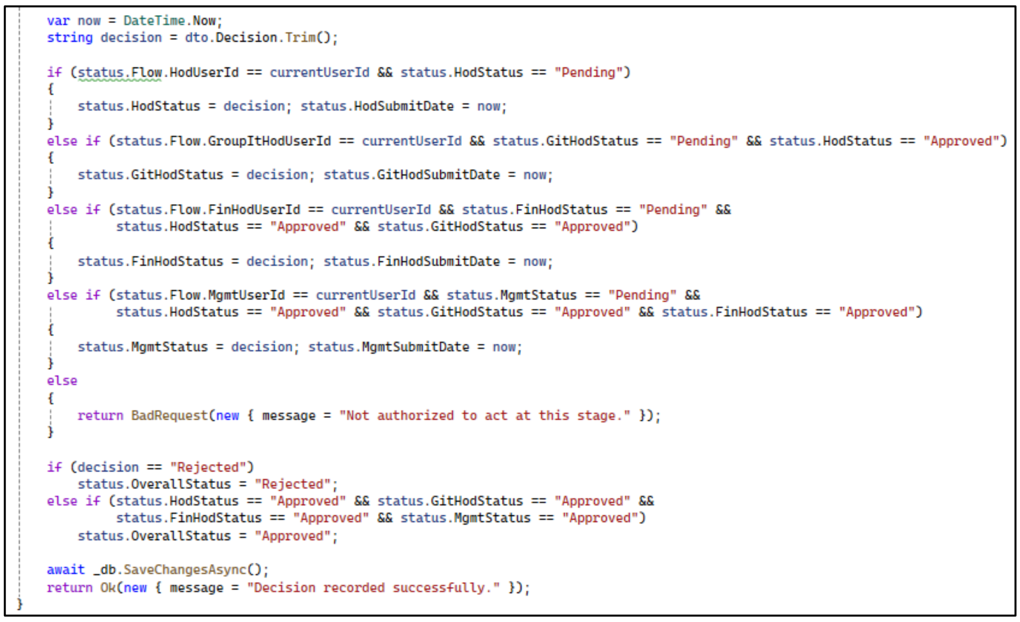
- **Purpose:** Records the approver's decision (Approve/Reject) for one IT request, enforcing the correct stage order.

- **Key functionality:**
    - **Auth & input:** Reads currentUserId from Identity; rejects empty payload or missing Decision.
    - **Load record:** Gets ItRequestStatus (with Flow and Request) by StatusId.
    - **Hard guards:**
        - Blocks if OverallStatus is Draft (not submitted) or Cancelled.
        - Blocks if any prior step already **Rejected**.
    - **Stage-by-stage routing:**
        - **HOD** may act only when HodStatus == "Pending".
        - **GIT_HOD** only after HOD Approved and GitHodStatus == "Pending".

        - **FIN_HOD** only after HOD & GIT_HOD Approved and FinHodStatus == "Pending".
        - **MGMT** only after HOD, GIT_HOD, FIN_HOD Approved and MgmtStatus == "Pending".
        - If none match, returns “Not authorized to act at this stage.”
    - **Overall status:**
        - If decision is **Rejected** → OverallStatus = "Rejected".
        - If all four steps are **Approved** → OverallStatus = "Approved".
    - **Audit fields:** Sets the corresponding *SubmitDate = now on the step that acted.
- **Input (body):**
    - { "StatusId": 123.., "Decision": "Approved" } // or "Rejected"
- **Output:**
    - 200 OK { message: "Decision recorded successfully." }
- **Errors & edge cases:**
    - 400 BadRequest for invalid payload, prior rejection exists, or acting out of turn.
    - 401 Unauthorized if no valid user.
    - 404 NotFound if StatusId missing.
    - 409 Conflict if Draft/Cancelled.

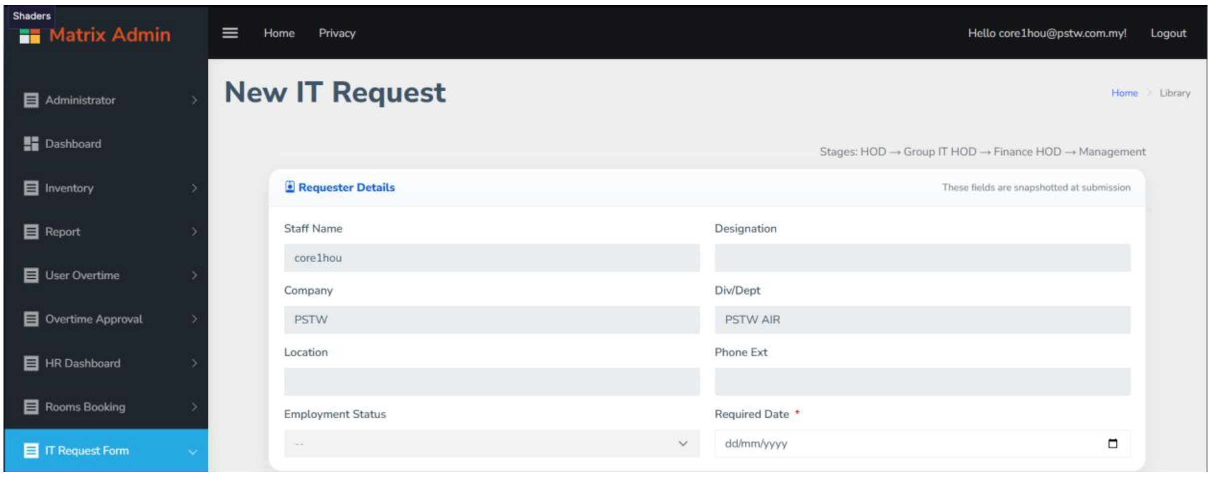


```
[HttpPost("approveReject")]
0 references
public async Task<IActionResult> ApproveReject([FromBody] ApproveRejectDto dto)
{
    if (dto == null || string.IsNullOrWhiteSpace(dto.Decision))
        return BadRequest(new { message = "Invalid request" });

    var userIdStr = _userManager.GetUserId(User);
    if (string.IsNullOrEmpty(userIdStr) || !int.TryParse(userIdStr, out int currentUserId))
        return Unauthorized("Invalid user");

    var status = await _db.ItRequestStatus
        .Include(s => s.Flow)
        .Include(s => s.Request)
        .FirstOrDefaultAsync(s => s.StatusId == dto.StatusId);

    if (status == null) return NotFound("Status not found");

    var overall = status.OverallStatus ?? "Draft";

    // Drafts cannot be approved/rejected ever
    if (overall == "Draft")
        return StatusCode(409, new { message = "This request isn't submitted yet. Approvals are only allowed once it is Pending." });

    // CANCELLED: hard stop
    if (string.Equals(overall, "Cancelled", StringComparison.OrdinalIgnoreCase))
        return StatusCode(409, new { message = "This request has been Cancelled and cannot be approved or rejected." });

    // If any earlier stage already rejected, block
    if (status.HodStatus == "Rejected" || status.GitHodStatus == "Rejected" ||
        status.FinHodStatus == "Rejected" || status.MgmtStatus == "Rejected")
        return BadRequest(new { message = "Request already rejected by a previous approver." });
```



```
    var now = DateTime.Now;
    string decision = dto.Decision.Trim();

    if (status.Flow.HodUserId == currentUserId && status.HodStatus == "Pending")
    {
        status.HodStatus = decision; status.HodSubmitDate = now;
    }
    else if (status.Flow.GroupItHodUserId == currentUserId && status.GitHodStatus == "Pending" && status.HodStatus == "Approved")
    {
        status.GitHodStatus = decision; status.GitHodSubmitDate = now;
    }
    else if (status.Flow.FinHodUserId == currentUserId && status.FinHodStatus == "Pending" &&
             status.HodStatus == "Approved" && status.GitHodStatus == "Approved")
    {
        status.FinHodStatus = decision; status.FinHodSubmitDate = now;
    }
    else if (status.Flow.MgmtUserId == currentUserId && status.MgmtStatus == "Pending" &&
             status.HodStatus == "Approved" && status.GitHodStatus == "Approved" && status.FinHodStatus == "Approved")
    {
        status.MgmtStatus = decision; status.MgmtSubmitDate = now;
    }
    else
    {
        return BadRequest(new { message = "Not authorized to act at this stage." });
    }

    if (decision == "Rejected")
        status.OverallStatus = "Rejected";
    else if (status.HodStatus == "Approved" && status.GitHodStatus == "Approved" &&
             status.FinHodStatus == "Approved" && status.MgmtStatus == "Approved")
        status.OverallStatus = "Approved";

    await _db.SaveChangesAsync();
    return Ok(new { message = "Decision recorded successfully." });
}
```

- Create.cshtml

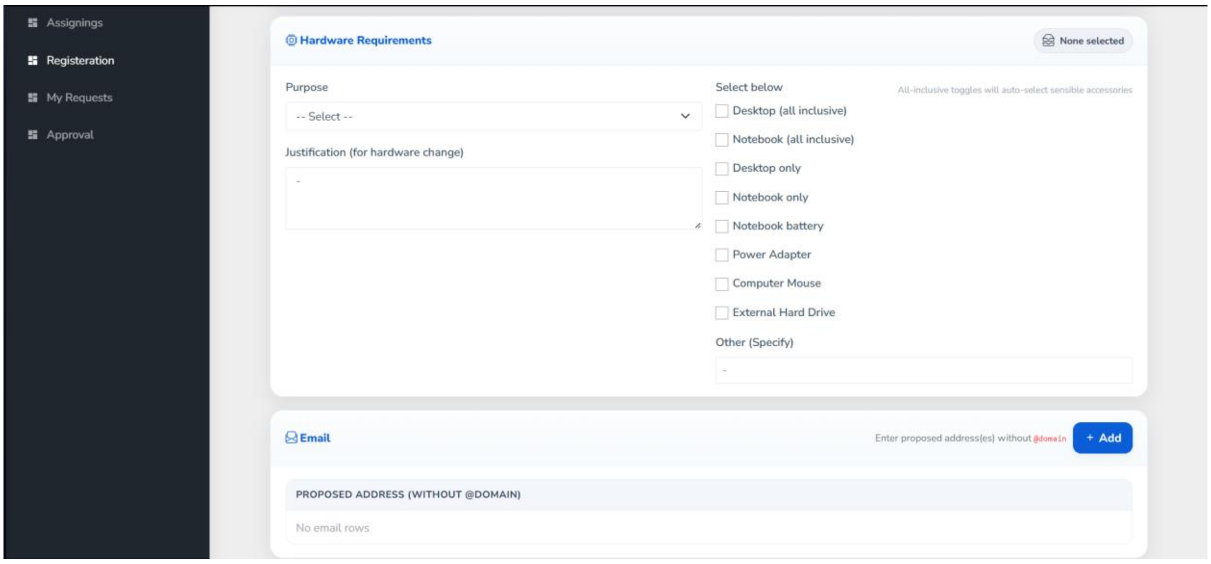


New IT Request

Stages: HOD → Group IT HOD → Finance HOD → Management

Requester Details These fields are snapshotted at submission

Staff Name: core1hou	Designation:
Company: PSTW	Div/Dept: PSTW AIR
Location:	Phone Ext:
Employment Status: --	Required Date *: dd/mm/yyyy



Hardware Requirements

Purpose: -- Select --

Justification (for hardware change): -

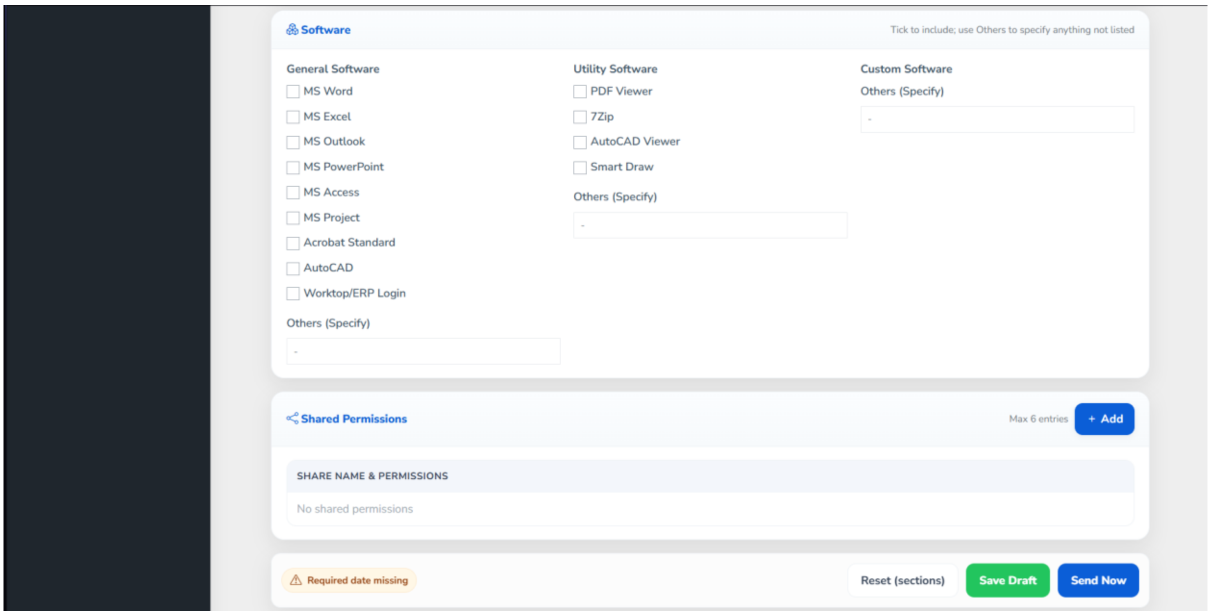
Select below — All-inclusive toggles will auto-select sensible accessories

- [ ] Desktop (all inclusive)
- [ ] Notebook (all inclusive)
- [ ] Desktop only
- [ ] Notebook only
- [ ] Notebook battery
- [ ] Power Adapter
- [ ] Computer Mouse
- [ ] External Hard Drive

Other (Specify): -

Email Enter proposed address(es) without @domain

PROPOSED ADDRESS (WITHOUT @DOMAIN)
No email rows



Software Tick to include; use Others to specify anything not listed

General Software	Utility Software	Custom Software
[ ] MS Word	[ ] PDF Viewer	Others (Specify): -
[ ] MS Excel	[ ] 7Zip	
[ ] MS Outlook	[ ] AutoCAD Viewer	
[ ] MS PowerPoint	[ ] Smart Draw	
[ ] MS Access	Others (Specify): -	
[ ] MS Project		
[ ] Acrobat Standard		
[ ] AutoCAD		
[ ] Worktop/ERP Login		
Others (Specify): -		

Shared Permissions Max 6 entries

SHARE NAME & PERMISSIONS
No shared permissions

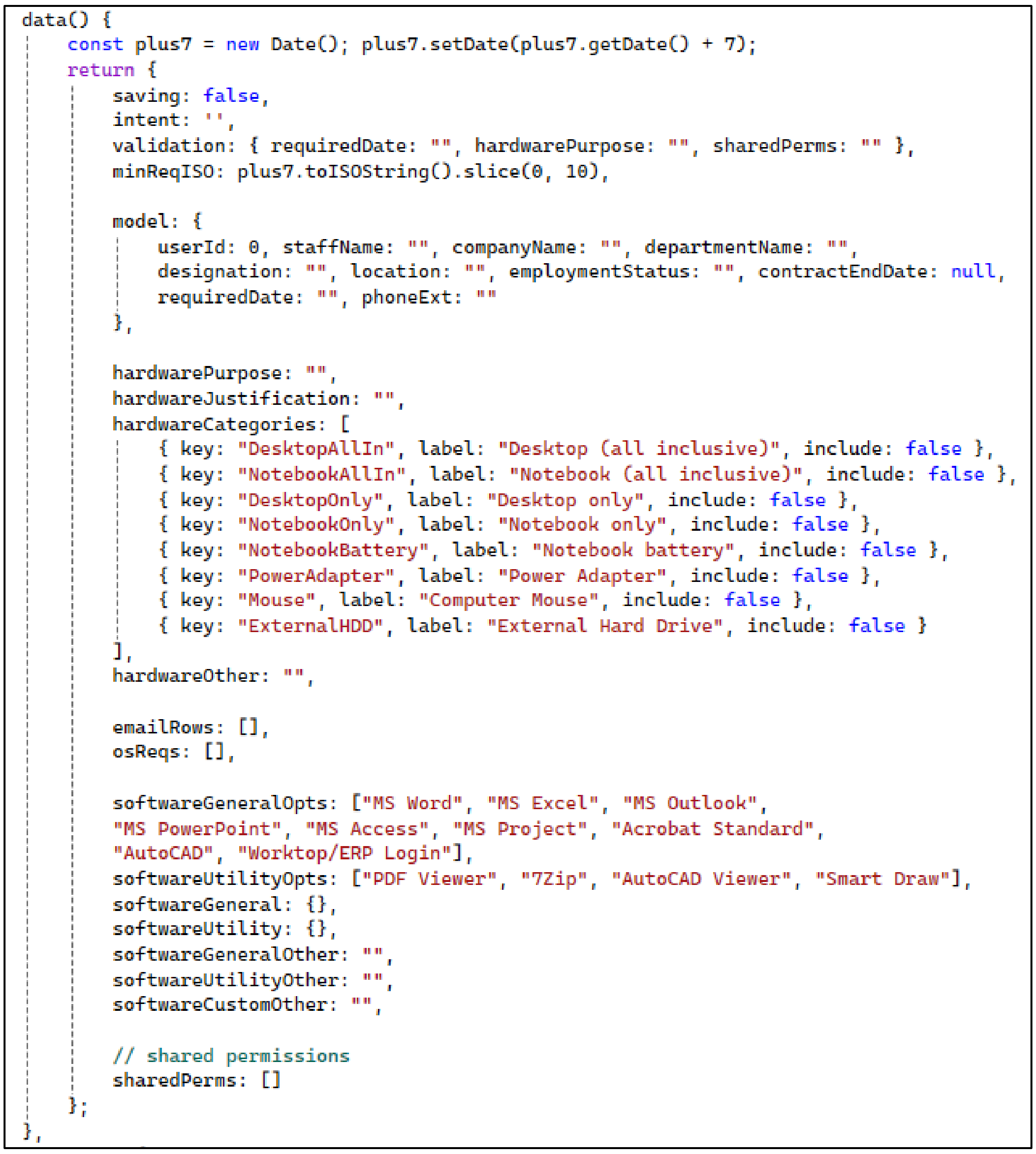
⚠ Required date missing

This file (Create.cshtml) is the self-service page where a staff member drafts and submits a new IT Request. It captures a snapshot of requester info, selected hardware and software, email needs, OS requirements, and shared folder permissions, then sends a single payload to the API.

- ✓ **<div id="itFormApp">…</div>**
  - **Purpose:** Root container for the Vue app so all inputs and UI states are reactive.

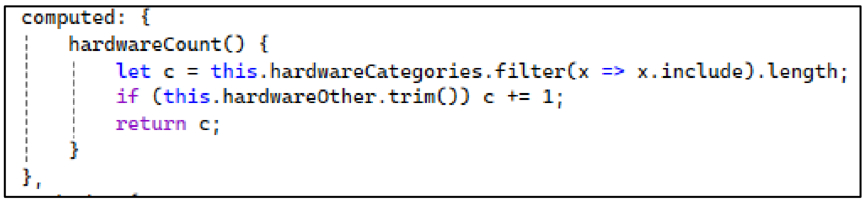
- ✓ **const app = Vue.createApp({...})**
  - **Purpose:** Boots the Vue instance and wires up form state, validation, helpers, and submission flow.

- ✓ **data()**
  - **saving, intent, validation, minReqISO**: These control form UX. saving disables buttons while requests are in flight, intent tags which action is running, validation stores field-level messages, and minReqISO enforces a required date at least seven days ahead.
  - **model { user snapshot }**: Stores the read-only requester fields that get snapshotted into the request at submit time. Values come from the server and include staff, org, location, employment status, and phone extension, plus the chosen requiredDate.
  - **hardwarePurpose, hardwareJustification, hardwareCategories, hardwareOther**: Collects the hardware section. Categories are defined as toggleable items, while hardwarePurpose and hardwareJustification capture the why, and hardwareOther lets the user specify anything not covered by presets.
  - **emailRows, osReqs**: Simple arrays for dynamic rows. Email holds proposed local parts without the domain, OS holds free-text requirements that justify platform choices.
  - *software groups**: Option lists are split into General and Utility for quick ticking, with corresponding state objects to track which boxes are checked. Three "Others" fields support free text for General, Utility, and Custom buckets
  - **sharedPerms**: An array of share-permission entries limited to six client side. Each entry carries a share name and four boolean flags for Read, Write, Delete, and Remove.



```
data() {
    const plus7 = new Date(); plus7.setDate(plus7.getDate() + 7);
    return {
        saving: false,
        intent: '',
        validation: { requiredDate: "", hardwarePurpose: "", sharedPerms: "" },
        minReqISO: plus7.toISOString().slice(0, 10),

        model: {
            userId: 0, staffName: "", companyName: "", departmentName: "",
            designation: "", location: "", employmentStatus: "", contractEndDate: null,
            requiredDate: "", phoneExt: ""
        },

        hardwarePurpose: "",
        hardwareJustification: "",
        hardwareCategories: [
            { key: "DesktopAllIn", label: "Desktop (all inclusive)", include: false },
            { key: "NotebookAllIn", label: "Notebook (all inclusive)", include: false },
            { key: "DesktopOnly", label: "Desktop only", include: false },
            { key: "NotebookOnly", label: "Notebook only", include: false },
            { key: "NotebookBattery", label: "Notebook battery", include: false },
            { key: "PowerAdapter", label: "Power Adapter", include: false },
            { key: "Mouse", label: "Computer Mouse", include: false },
            { key: "ExternalHDD", label: "External Hard Drive", include: false }
        ],
        hardwareOther: "",

        emailRows: [],
        osReqs: [],

        softwareGeneralOpts: ["MS Word", "MS Excel", "MS Outlook",
        "MS PowerPoint", "MS Access", "MS Project", "Acrobat Standard",
        "AutoCAD", "Worktop/ERP Login"],
        softwareUtilityOpts: ["PDF Viewer", "7Zip", "AutoCAD Viewer", "Smart Draw"],
        softwareGeneral: {},
        softwareUtility: {},
        softwareGeneralOther: "",
        softwareUtilityOther: "",
        softwareCustomOther: "",

        // shared permissions
        sharedPerms: []
    };
},
```

- ✓ **computed**
    - **hardwareCount**: Counts how many hardware presets are ticked and includes the “Other” textbox if it is filled. The header chip uses this to show None vs N selected and to decide whether purpose becomes mandatory.

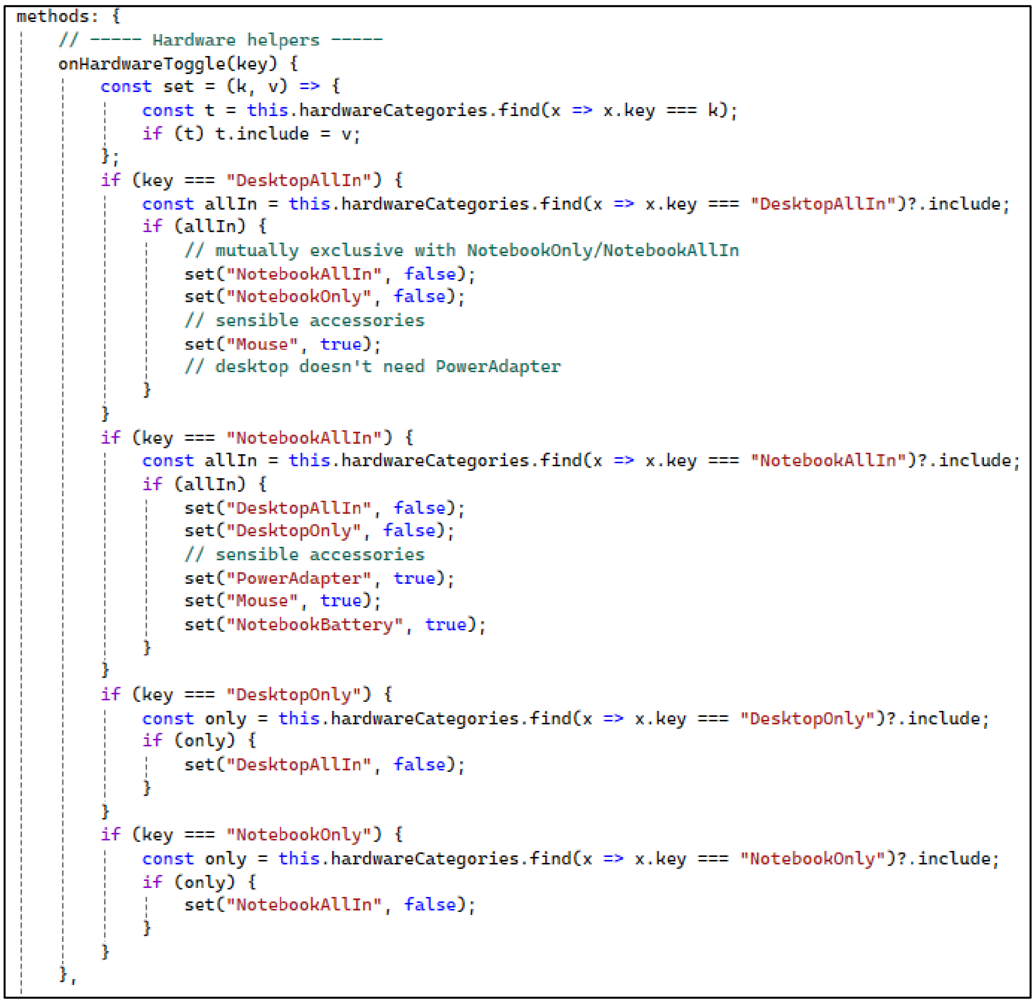


```
computed: {
    hardwareCount() {
        let c = this.hardwareCategories.filter(x => x.include).length;
        if (this.hardwareOther.trim()) c += 1;
        return c;
    }
},
```

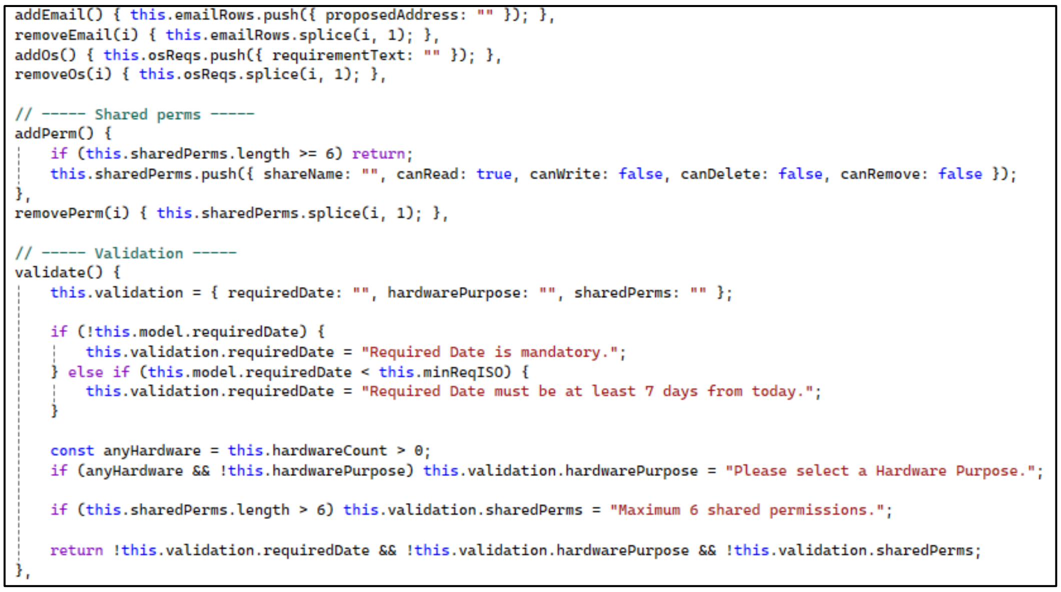
- ✓ **methods**

- **onHardwareToggle(key)**: Keeps the hardware toggles smart and conflict-free. Desktop All Inclusive disables Notebook options and preselects sensible accessories like Mouse, while Notebook All Inclusive disables Desktop options and turns on Power Adapter, Mouse, and Battery. The “Only” variants prevent their respective All Inclusive from staying on at the same time.
- **addEmail / removeEmail, addOs / removeOs**: Manage dynamic rows for Email and OS sections. These are straightforward array pushes and splices so the UI stays responsive and predictable.
- **addPerm / removePerm**: Adds a share permission row with safe defaults and enforces a maximum of six. Removing a row instantly updates the on-screen list and the validation state.
- **validate()**: Performs client checks before any network call. It requires a future requiredDate at least seven days out, enforces hardwarePurpose when any hardware is selected, and blocks more than six shared permissions.
- **buildDto()**: Assembles the exact payload the API expects in one place. It flattens hardware selections into objects that include category, purpose, justification, and “Other” text, maps email and OS rows, collects all ticked software into named buckets with optional “Others,” and trims shared permissions to six. It also includes editWindowHours so the backend knows how long the draft remains editable.
- **createRequest(sendNow = false)**: Posts the DTO to /ItRequestAPI/create with a sendNow flag. It normalizes the response, throws a meaningful error if the API fails, and returns the new statusId so the caller can decide what to do next.
- **saveDraft()**: Validates, sets intent to draft, then calls createRequest(false). On success it redirects to the My Requests dashboard so the user can see their new draft sitting in the list.
- **openConfirm()**: Shows a Bootstrap confirmation modal for final submission. If the Bootstrap bundle is not loaded yet, it falls back to a simple confirm dialog. The modal’s Yes button is wired to call sendNow.
- **sendNow()**: Validates and submits with sendNow = true, which locks the request and moves it into Pending. After success it redirects to My Requests, matching the draft flow so navigation stays consistent.

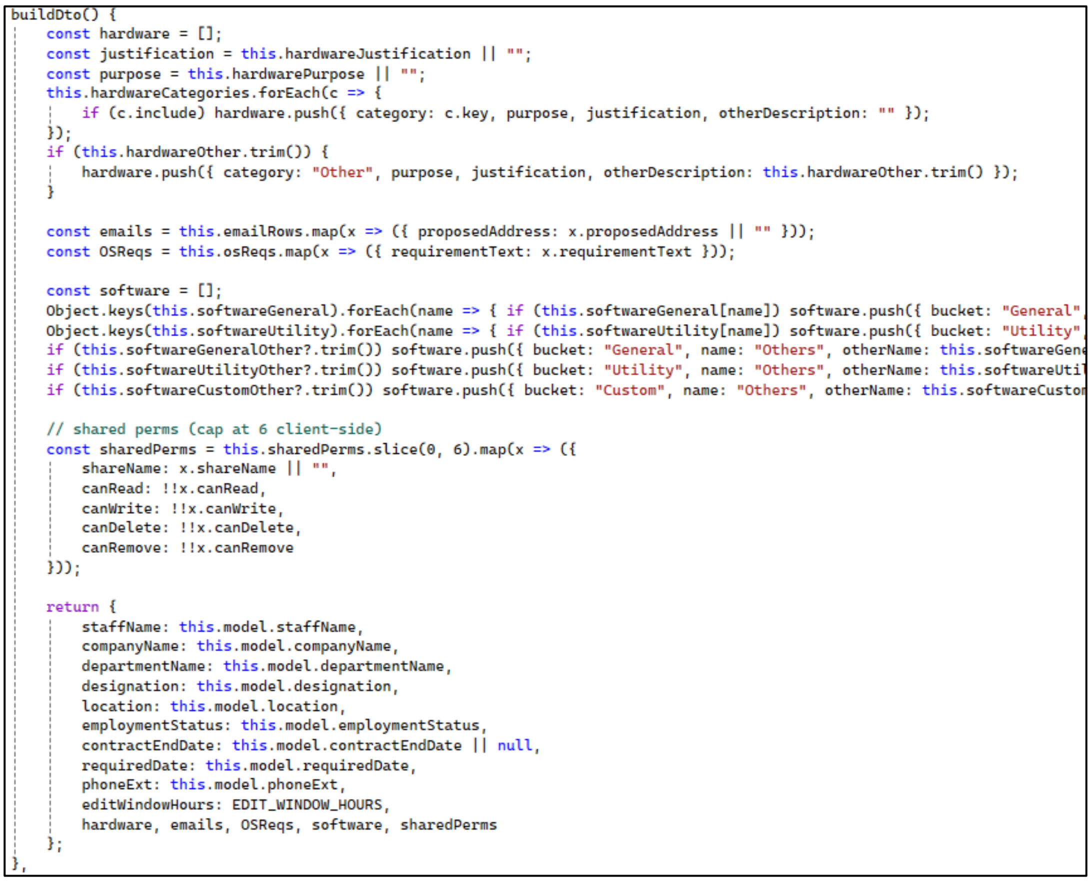
- **resetForm()**: Clears all section inputs and resets validation messages. Handy when a user changes their mind midway and wants a clean slate without reloading the page.
- **prefillFromServer()**
  Fetches /ItRequestAPI/me and fills the requester snapshot fields so the user does not have to type anything that HR already knows. These values are read-only and get embedded into the outgoing payload.



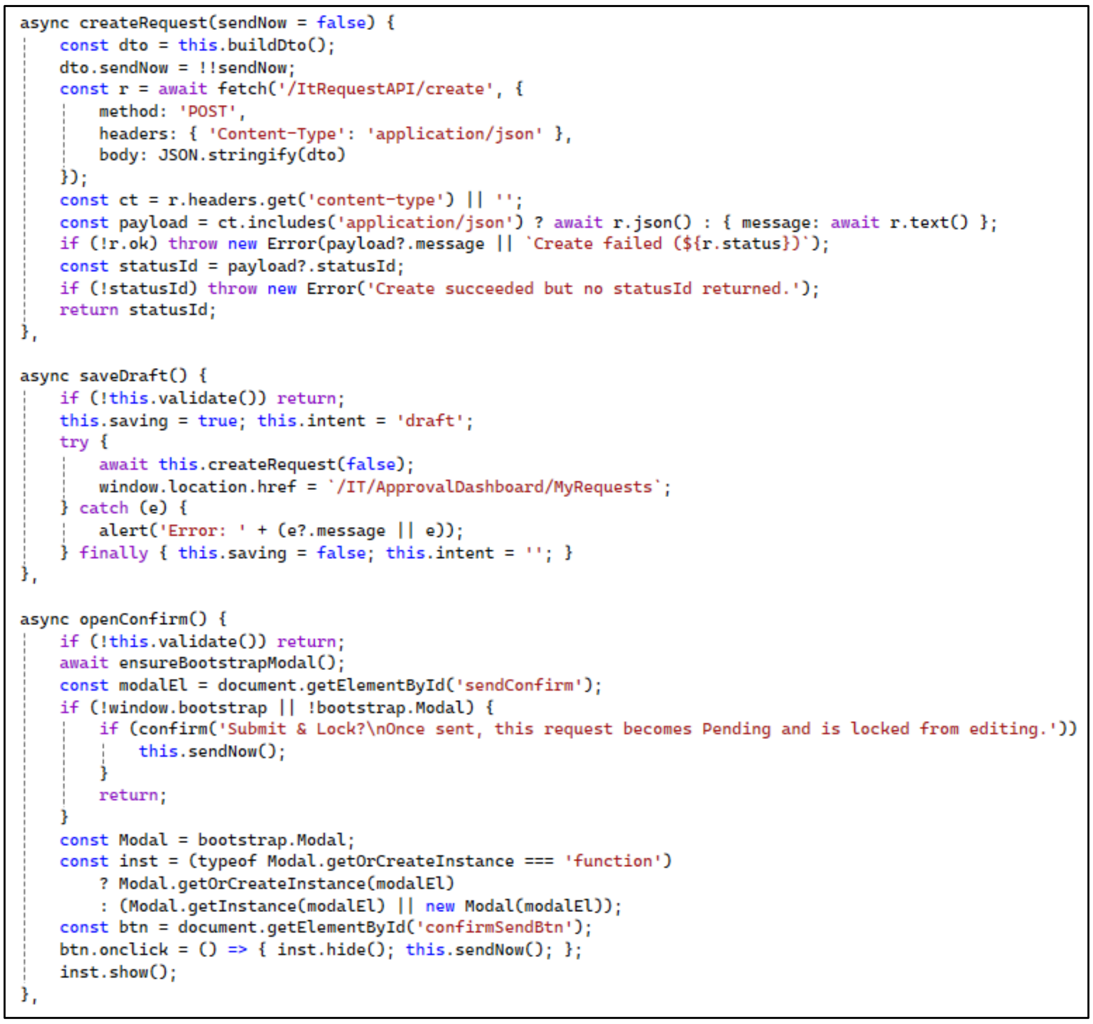
```
methods: {
    // ----- Hardware helpers -----
    onHardwareToggle(key) {
        const set = (k, v) => {
            const t = this.hardwareCategories.find(x => x.key === k);
            if (t) t.include = v;
        };
        if (key === "DesktopAllIn") {
            const allIn = this.hardwareCategories.find(x => x.key === "DesktopAllIn")?.include;
            if (allIn) {
                // mutually exclusive with NotebookOnly/NotebookAllIn
                set("NotebookAllIn", false);
                set("NotebookOnly", false);
                // sensible accessories
                set("Mouse", true);
                // desktop doesn't need PowerAdapter
            }
        }
        if (key === "NotebookAllIn") {
            const allIn = this.hardwareCategories.find(x => x.key === "NotebookAllIn")?.include;
            if (allIn) {
                set("DesktopAllIn", false);
                set("DesktopOnly", false);
                // sensible accessories
                set("PowerAdapter", true);
                set("Mouse", true);
                set("NotebookBattery", true);
            }
        }
        if (key === "DesktopOnly") {
            const only = this.hardwareCategories.find(x => x.key === "DesktopOnly")?.include;
            if (only) {
                set("DesktopAllIn", false);
            }
        }
        if (key === "NotebookOnly") {
            const only = this.hardwareCategories.find(x => x.key === "NotebookOnly")?.include;
            if (only) {
                set("NotebookAllIn", false);
            }
        }
    },
```



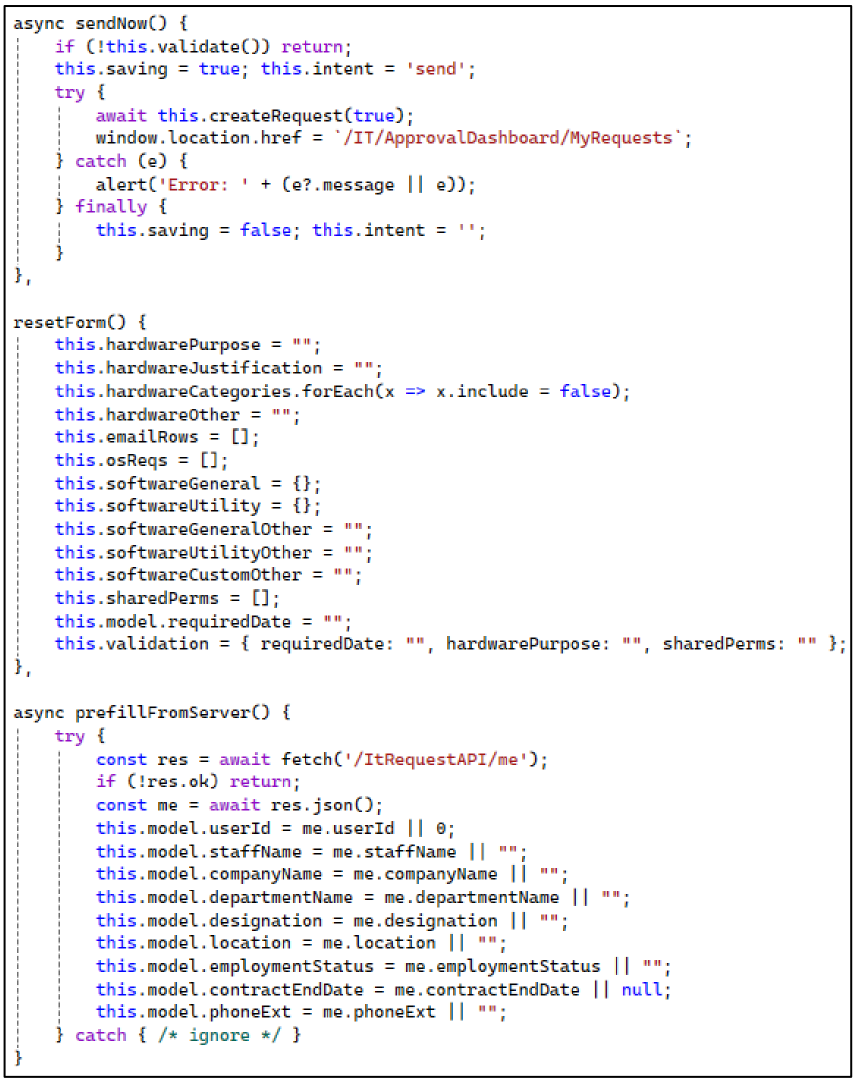
```
addEmail() { this.emailRows.push({ proposedAddress: "" }); },
removeEmail(i) { this.emailRows.splice(i, 1); },
addOs() { this.osReqs.push({ requirementText: "" }); },
removeOs(i) { this.osReqs.splice(i, 1); },

// ----- Shared perms -----
addPerm() {
    if (this.sharedPerms.length >= 6) return;
    this.sharedPerms.push({ shareName: "", canRead: true, canWrite: false, canDelete: false, canRemove: false });
},
removePerm(i) { this.sharedPerms.splice(i, 1); },

// ----- Validation -----
validate() {
    this.validation = { requiredDate: "", hardwarePurpose: "", sharedPerms: "" };

    if (!this.model.requiredDate) {
        this.validation.requiredDate = "Required Date is mandatory.";
    } else if (this.model.requiredDate < this.minReqISO) {
        this.validation.requiredDate = "Required Date must be at least 7 days from today.";
    }

    const anyHardware = this.hardwareCount > 0;
    if (anyHardware && !this.hardwarePurpose) this.validation.hardwarePurpose = "Please select a Hardware Purpose.";

    if (this.sharedPerms.length > 6) this.validation.sharedPerms = "Maximum 6 shared permissions.";

    return !this.validation.requiredDate && !this.validation.hardwarePurpose && !this.validation.sharedPerms;
},
```



```
buildDto() {
    const hardware = [];
    const justification = this.hardwareJustification || "";
    const purpose = this.hardwarePurpose || "";
    this.hardwareCategories.forEach(c => {
        if (c.include) hardware.push({ category: c.key, purpose, justification, otherDescription: "" });
    });
    if (this.hardwareOther.trim()) {
        hardware.push({ category: "Other", purpose, justification, otherDescription: this.hardwareOther.trim() });
    }

    const emails = this.emailRows.map(x => ({ proposedAddress: x.proposedAddress || "" }));
    const OSReqs = this.osReqs.map(x => ({ requirementText: x.requirementText }));

    const software = [];
    Object.keys(this.softwareGeneral).forEach(name => { if (this.softwareGeneral[name]) software.push({ bucket: "General",
    Object.keys(this.softwareUtility).forEach(name => { if (this.softwareUtility[name]) software.push({ bucket: "Utility",
    if (this.softwareGeneralOther?.trim()) software.push({ bucket: "General", name: "Others", otherName: this.softwareGene
    if (this.softwareUtilityOther?.trim()) software.push({ bucket: "Utility", name: "Others", otherName: this.softwareUti
    if (this.softwareCustomOther?.trim()) software.push({ bucket: "Custom", name: "Others", otherName: this.softwareCustom

    // shared perms (cap at 6 client-side)
    const sharedPerms = this.sharedPerms.slice(0, 6).map(x => ({
        shareName: x.shareName || "",
        canRead: !!x.canRead,
        canWrite: !!x.canWrite,
        canDelete: !!x.canDelete,
        canRemove: !!x.canRemove
    }));

    return {
        staffName: this.model.staffName,
        companyName: this.model.companyName,
        departmentName: this.model.departmentName,
        designation: this.model.designation,
        location: this.model.location,
        employmentStatus: this.model.employmentStatus,
        contractEndDate: this.model.contractEndDate || null,
        requiredDate: this.model.requiredDate,
        phoneExt: this.model.phoneExt,
        editWindowHours: EDIT_WINDOW_HOURS,
        hardware, emails, OSReqs, software, sharedPerms
    };
},
```



```
async createRequest(sendNow = false) {
    const dto = this.buildDto();
    dto.sendNow = !!sendNow;
    const r = await fetch('/ItRequestAPI/create', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(dto)
    });
    const ct = r.headers.get('content-type') || '';
    const payload = ct.includes('application/json') ? await r.json() : { message: await r.text() };
    if (!r.ok) throw new Error(payload?.message || `Create failed (${r.status})`);
    const statusId = payload?.statusId;
    if (!statusId) throw new Error('Create succeeded but no statusId returned.');
    return statusId;
},

async saveDraft() {
    if (!this.validate()) return;
    this.saving = true; this.intent = 'draft';
    try {
        await this.createRequest(false);
        window.location.href = `/IT/ApprovalDashboard/MyRequests`;
    } catch (e) {
        alert('Error: ' + (e?.message || e));
    } finally { this.saving = false; this.intent = ''; }
},

async openConfirm() {
    if (!this.validate()) return;
    await ensureBootstrapModal();
    const modalEl = document.getElementById('sendConfirm');
    if (!window.bootstrap || !bootstrap.Modal) {
        if (confirm('Submit & Lock?\nOnce sent, this request becomes Pending and is locked from editing.'))
            this.sendNow();
        }
        return;
    }
    const Modal = bootstrap.Modal;
    const inst = (typeof Modal.getOrCreateInstance === 'function')
        ? Modal.getOrCreateInstance(modalEl)
        : (Modal.getInstance(modalEl) || new Modal(modalEl));
    const btn = document.getElementById('confirmSendBtn');
    btn.onclick = () => { inst.hide(); this.sendNow(); };
    inst.show();
},
```



```
async sendNow() {
    if (!this.validate()) return;
    this.saving = true; this.intent = 'send';
    try {
        await this.createRequest(true);
        window.location.href = `/IT/ApprovalDashboard/MyRequests`;
    } catch (e) {
        alert('Error: ' + (e?.message || e));
    } finally {
        this.saving = false; this.intent = '';
    }
},

resetForm() {
    this.hardwarePurpose = "";
    this.hardwareJustification = "";
    this.hardwareCategories.forEach(x => x.include = false);
    this.hardwareOther = "";
    this.emailRows = [];
    this.osReqs = [];
    this.softwareGeneral = {};
    this.softwareUtility = {};
    this.softwareGeneralOther = "";
    this.softwareUtilityOther = "";
    this.softwareCustomOther = "";
    this.sharedPerms = [];
    this.model.requiredDate = "";
    this.validation = { requiredDate: "", hardwarePurpose: "", sharedPerms: "" };
},

async prefillFromServer() {
    try {
        const res = await fetch('/ItRequestAPI/me');
        if (!res.ok) return;
        const me = await res.json();
        this.model.userId = me.userId || 0;
        this.model.staffName = me.staffName || "";
        this.model.companyName = me.companyName || "";
        this.model.departmentName = me.departmentName || "";
        this.model.designation = me.designation || "";
        this.model.location = me.location || "";
        this.model.employmentStatus = me.employmentStatus || "";
        this.model.contractEndDate = me.contractEndDate || null;
        this.model.phoneExt = me.phoneExt || "";
    } catch { /* ignore */ }
}
```

✓ **mounted**

- **mounted()**: Calls prefillFromServer immediately so identity and org details are present before the user starts choosing requirements. This keeps the top card accurate and reinforces that the request will capture a server-verified snapshot

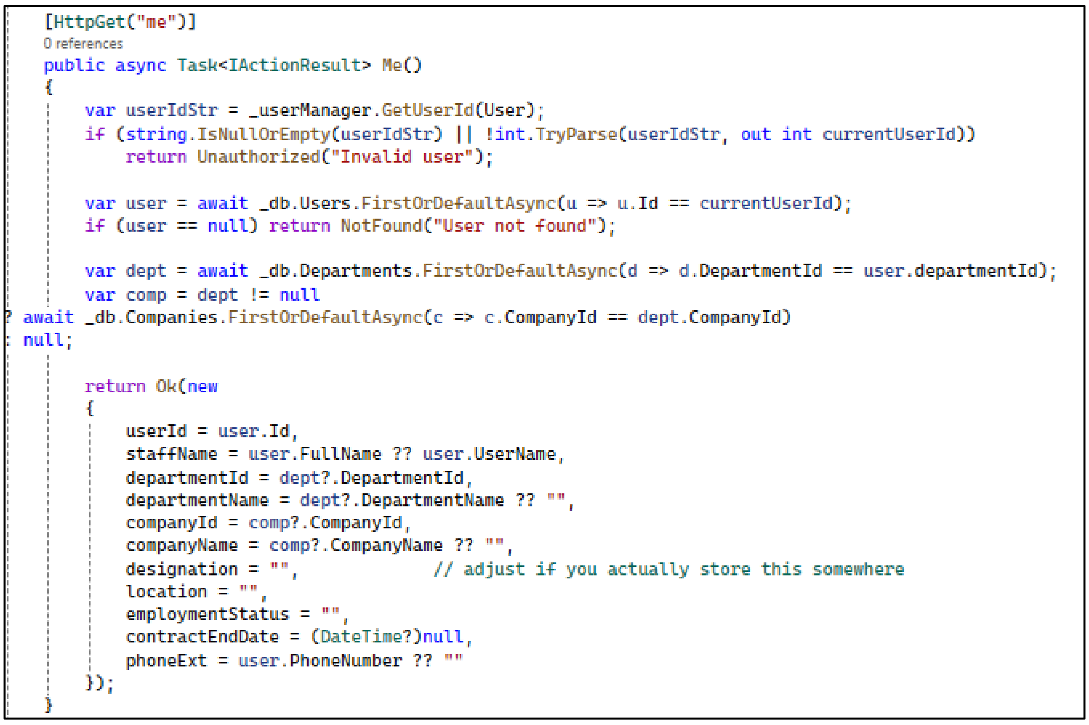
```
mounted() { this.prefillFromServer(); }
```

- ITRequestAPI (Create.cshtml)
  - **[HttpGet("me")]** (Section A prefill)

- **Purpose:** Returns the logged-in user's profile snapshot so the Create form can auto-populate Section A (staff, department, company, phone, etc.).

- **Key functionality:**

- **Auth and user lookup:** Reads currentUserId from ASP.NET Identity; 401 if missing/invalid. Fetches Users row; 404 if not found.
- **Department to Company chain:** Attempts Departments by user.departmentId. If found, fetches Companies by dept.CompanyId. Either may be null (endpoint guards by returning empty strings/nullable IDs so the form can still load).
- **Name fallback:** Uses FullName if available, otherwise UserName.
- **Placeholders:** designation, location, employmentStatus, contractEndDate are returned as blanks/null unless you later wire them to real columns.



```
    [HttpGet("me")]
    0 references
    public async Task<IActionResult> Me()
    {
        var userIdStr = _userManager.GetUserId(User);
        if (string.IsNullOrEmpty(userIdStr) || !int.TryParse(userIdStr, out int currentUserId))
            return Unauthorized("Invalid user");

        var user = await _db.Users.FirstOrDefaultAsync(u => u.Id == currentUserId);
        if (user == null) return NotFound("User not found");

        var dept = await _db.Departments.FirstOrDefaultAsync(d => d.DepartmentId == user.departmentId);
        var comp = dept != null
? await _db.Companies.FirstOrDefaultAsync(c => c.CompanyId == dept.CompanyId)
: null;

        return Ok(new
        {
            userId = user.Id,
            staffName = user.FullName ?? user.UserName,
            departmentId = dept?.DepartmentId,
            departmentName = dept?.DepartmentName ?? "",
            companyId = comp?.CompanyId,
            companyName = comp?.CompanyName ?? "",
            designation = "",            // adjust if you actually store this somewhere
            location = "",
            employmentStatus = "",
            contractEndDate = (DateTime?)null,
            phoneExt = user.PhoneNumber ?? ""
        });
    }
```

- **[HttpPost("create")]** (when user submits Section A)

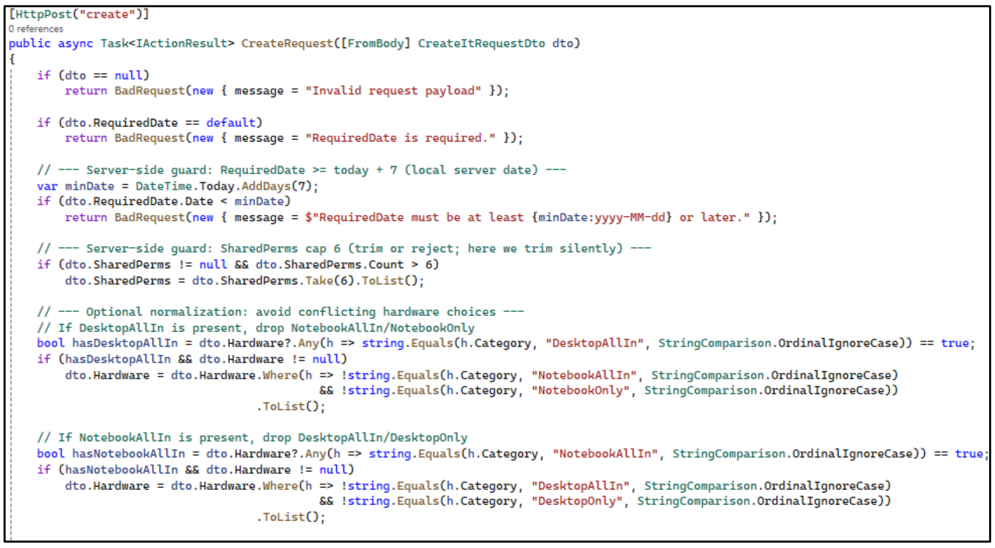
- **Purpose:**
  This endpoint is used when someone fills up the IT Request form and hits “Create” or “Send Now.” It saves all the entered details — like hardware, software, and shared access — and sets up the approval flow so it’s ready to go through the HOD > Group IT > Finance > Management stages.

- **Key functionalities:**

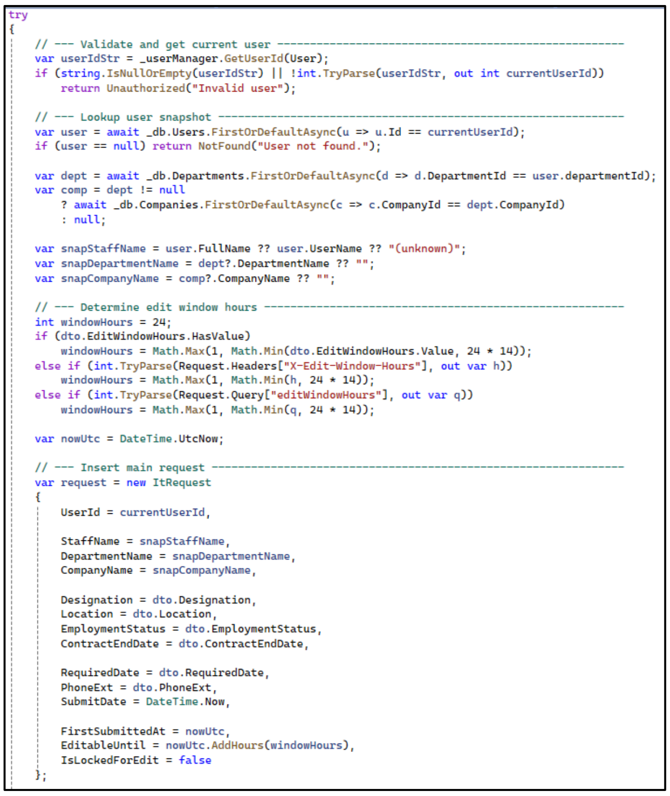
- **Initial checks:**
  It first makes sure the request body isn’t empty and that the RequiredDate is valid. The system also won’t allow you to pick a date that’s too soon — it must be at least 7 days from today.
  For shared permissions, it only keeps up to 6 entries to prevent overload.
- **Hardware rules:**
  If the user chooses “Desktop All-In,” the system automatically removes any notebook-related options (and vice versa). This avoids conflicting selections, like asking for both a desktop and a notebook at once.
- **User validation:**
  It identifies who’s creating the request by checking the logged-in user’s ID. Then it pulls their profile, department, and company info from the database. These values are saved as a snapshot inside the request (so even if the user moves department later, the form still shows their original details).
- **Edit window setup:**
  Every new request has an edit window which is 24 hours by default, but it can be changed (up to 14 days) using the body, header, or query string. Within this window, the requester can still edit the submission before it locks.
- **Saving the main record:**
  Once the user and details are confirmed, the main IT Request record is created. It stores things like staff name, department, designation, required date, and contact number.
- **Adding the sub-lists:**
  After saving the main form, the system adds all the smaller lists like hardware, software, emails, OS requirements, and shared permissions under that same request ID.
- **Approval setup:**
  It links the new request to the default approval flow and creates an entry in the ItRequestStatus table, setting all stages (HOD, GIT HOD, FIN HOD, MGMT) to “Pending,” while the overall request starts as “Draft.”

- **If "Send Now" is used:**

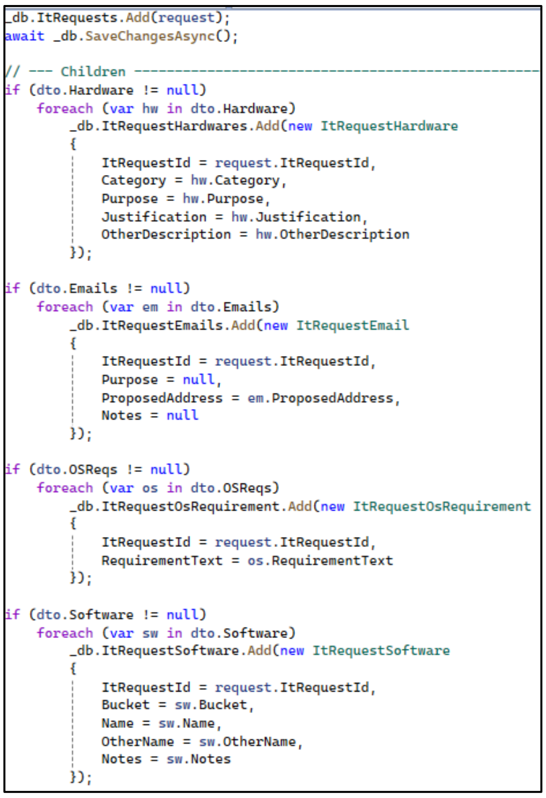
  The system immediately locks the form (no more edits) and changes the overall status from "Draft" to "Pending," officially starting the approval process.



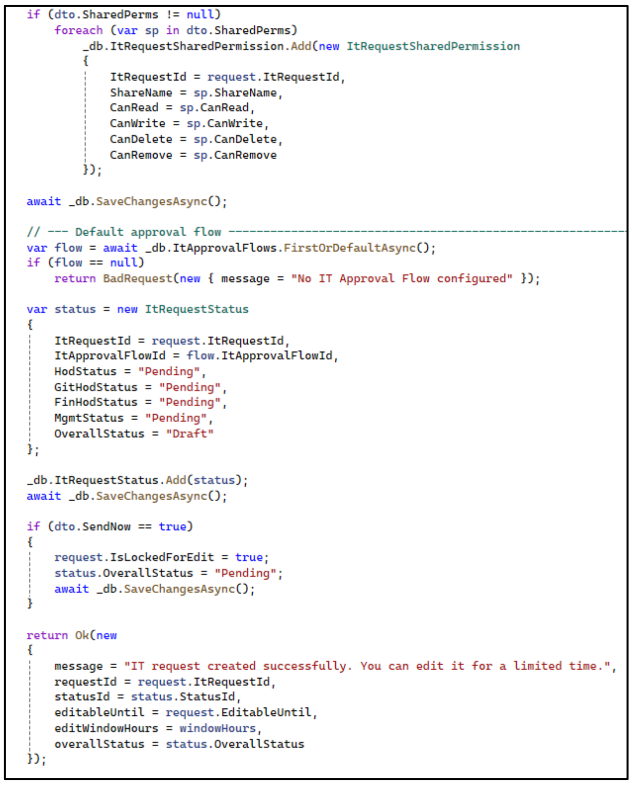
```
[HttpPost("create")]
0 references
public async Task<IActionResult> CreateRequest([FromBody] CreateItRequestDto dto)
{
    if (dto == null)
        return BadRequest(new { message = "Invalid request payload" });

    if (dto.RequiredDate == default)
        return BadRequest(new { message = "RequiredDate is required." });

    // --- Server-side guard: RequiredDate >= today + 7 (local server date) ---
    var minDate = DateTime.Today.AddDays(7);
    if (dto.RequiredDate.Date < minDate)
        return BadRequest(new { message = $"RequiredDate must be at least {minDate:yyyy-MM-dd} or later." });

    // --- Server-side guard: SharedPerms cap 6 (trim or reject; here we trim silently) ---
    if (dto.SharedPerms != null && dto.SharedPerms.Count > 6)
        dto.SharedPerms = dto.SharedPerms.Take(6).ToList();

    // --- Optional normalization: avoid conflicting hardware choices ---
    // If DesktopAllIn is present, drop NotebookAllIn/NotebookOnly
    bool hasDesktopAllIn = dto.Hardware?.Any(h => string.Equals(h.Category, "DesktopAllIn", StringComparison.OrdinalIgnoreCase)) == true;
    if (hasDesktopAllIn && dto.Hardware != null)
        dto.Hardware = dto.Hardware.Where(h => !string.Equals(h.Category, "NotebookAllIn", StringComparison.OrdinalIgnoreCase)
                                            && !string.Equals(h.Category, "NotebookOnly", StringComparison.OrdinalIgnoreCase))
                                   .ToList();

    // If NotebookAllIn is present, drop DesktopAllIn/DesktopOnly
    bool hasNotebookAllIn = dto.Hardware?.Any(h => string.Equals(h.Category, "NotebookAllIn", StringComparison.OrdinalIgnoreCase)) == true;
    if (hasNotebookAllIn && dto.Hardware != null)
        dto.Hardware = dto.Hardware.Where(h => !string.Equals(h.Category, "DesktopAllIn", StringComparison.OrdinalIgnoreCase)
                                            && !string.Equals(h.Category, "DesktopOnly", StringComparison.OrdinalIgnoreCase))
                                   .ToList();
```



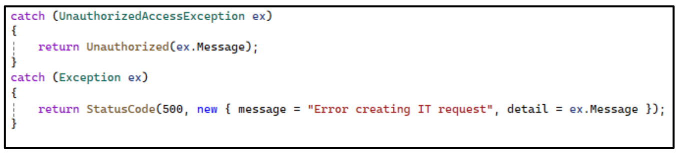
```
try
{
    // --- Validate and get current user -----------------------------------------------------
    var userIdStr = _userManager.GetUserId(User);
    if (string.IsNullOrEmpty(userIdStr) || !int.TryParse(userIdStr, out int currentUserId))
        return Unauthorized("Invalid user");

    // --- Lookup user snapshot --------------------------------------------------------------
    var user = await _db.Users.FirstOrDefaultAsync(u => u.Id == currentUserId);
    if (user == null) return NotFound("User not found.");

    var dept = await _db.Departments.FirstOrDefaultAsync(d => d.DepartmentId == user.departmentId);
    var comp = dept != null
        ? await _db.Companies.FirstOrDefaultAsync(c => c.CompanyId == dept.CompanyId)
        : null;

    var snapStaffName = user.FullName ?? user.UserName ?? "(unknown)";
    var snapDepartmentName = dept?.DepartmentName ?? "";
    var snapCompanyName = comp?.CompanyName ?? "";

    // --- Determine edit window hours -------------------------------------------------------
    int windowHours = 24;
    if (dto.EditWindowHours.HasValue)
        windowHours = Math.Max(1, Math.Min(dto.EditWindowHours.Value, 24 * 14));
    else if (int.TryParse(Request.Headers["X-Edit-Window-Hours"], out var h))
        windowHours = Math.Max(1, Math.Min(h, 24 * 14));
    else if (int.TryParse(Request.Query["editWindowHours"], out var q))
        windowHours = Math.Max(1, Math.Min(q, 24 * 14));

    var nowUtc = DateTime.UtcNow;

    // --- Insert main request ---------------------------------------------------------------
    var request = new ItRequest
    {
        UserId = currentUserId,

        StaffName = snapStaffName,
        DepartmentName = snapDepartmentName,
        CompanyName = snapCompanyName,

        Designation = dto.Designation,
        Location = dto.Location,
        EmploymentStatus = dto.EmploymentStatus,
        ContractEndDate = dto.ContractEndDate,

        RequiredDate = dto.RequiredDate,
        PhoneExt = dto.PhoneExt,
        SubmitDate = DateTime.Now,

        FirstSubmittedAt = nowUtc,
        EditableUntil = nowUtc.AddHours(windowHours),
        IsLockedForEdit = false
    };
```



```
_db.ItRequests.Add(request);
await _db.SaveChangesAsync();

// --- Children -----------------------------------------------------
if (dto.Hardware != null)
    foreach (var hw in dto.Hardware)
        _db.ItRequestHardwares.Add(new ItRequestHardware
        {
            ItRequestId = request.ItRequestId,
            Category = hw.Category,
            Purpose = hw.Purpose,
            Justification = hw.Justification,
            OtherDescription = hw.OtherDescription
        });

if (dto.Emails != null)
    foreach (var em in dto.Emails)
        _db.ItRequestEmails.Add(new ItRequestEmail
        {
            ItRequestId = request.ItRequestId,
            Purpose = null,
            ProposedAddress = em.ProposedAddress,
            Notes = null
        });

if (dto.OSReqs != null)
    foreach (var os in dto.OSReqs)
        _db.ItRequestOsRequirement.Add(new ItRequestOsRequirement
        {
            ItRequestId = request.ItRequestId,
            RequirementText = os.RequirementText
        });

if (dto.Software != null)
    foreach (var sw in dto.Software)
        _db.ItRequestSoftware.Add(new ItRequestSoftware
        {
            ItRequestId = request.ItRequestId,
            Bucket = sw.Bucket,
            Name = sw.Name,
            OtherName = sw.OtherName,
            Notes = sw.Notes
        });
```

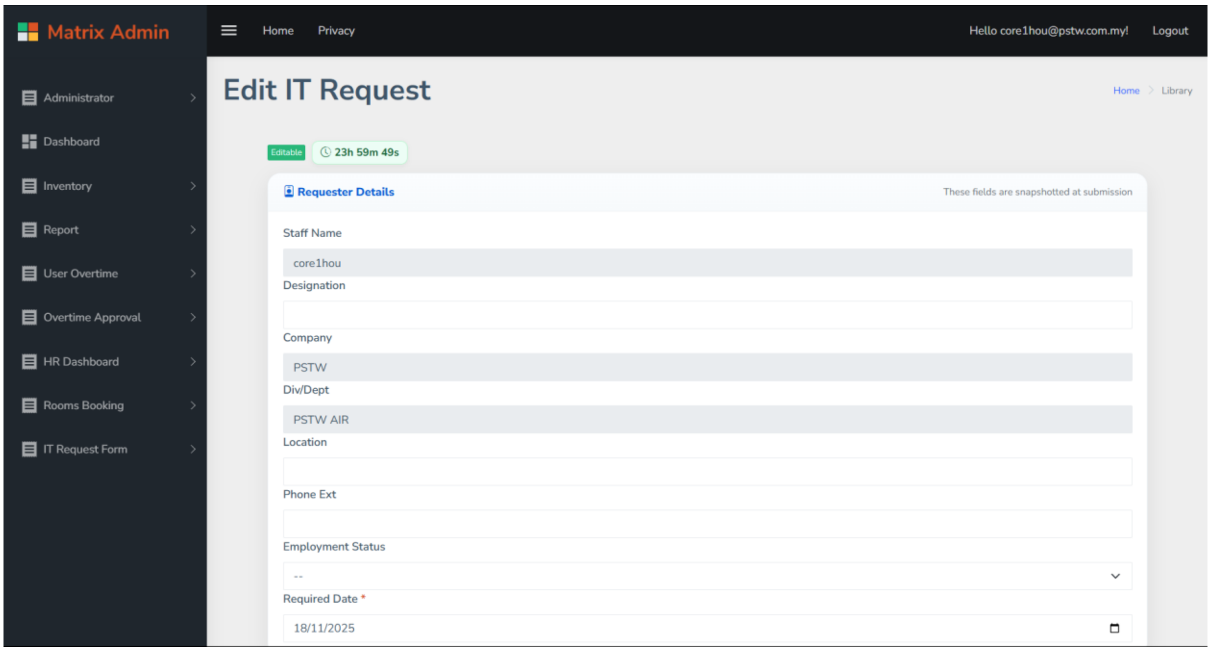


```
if (dto.SharedPerms != null)
    foreach (var sp in dto.SharedPerms)
        _db.ItRequestSharedPermission.Add(new ItRequestSharedPermission
        {
            ItRequestId = request.ItRequestId,
            ShareName = sp.ShareName,
            CanRead = sp.CanRead,
            CanWrite = sp.CanWrite,
            CanDelete = sp.CanDelete,
            CanRemove = sp.CanRemove
        });

await _db.SaveChangesAsync();

// --- Default approval flow ---------------------------------------------------------
var flow = await _db.ItApprovalFlows.FirstOrDefaultAsync();
if (flow == null)
    return BadRequest(new { message = "No IT Approval Flow configured" });

var status = new ItRequestStatus
{
    ItRequestId = request.ItRequestId,
    ItApprovalFlowId = flow.ItApprovalFlowId,
    HodStatus = "Pending",
    GitHodStatus = "Pending",
    FinHodStatus = "Pending",
    MgmtStatus = "Pending",
    OverallStatus = "Draft"
};

_db.ItRequestStatus.Add(status);
await _db.SaveChangesAsync();

if (dto.SendNow == true)
{
    request.IsLockedForEdit = true;
    status.OverallStatus = "Pending";
    await _db.SaveChangesAsync();
}

return Ok(new
{
    message = "IT request created successfully. You can edit it for a limited time.",
    requestId = request.ItRequestId,
    statusId = status.StatusId,
    editableUntil = request.EditableUntil,
    editWindowHours = windowHours,
    overallStatus = status.OverallStatus
});
```



```
catch (UnauthorizedAccessException ex)
{
    return Unauthorized(ex.Message);
}
catch (Exception ex)
{
    return StatusCode(500, new { message = "Error creating IT request", detail = ex.Message });
}
```

- Edit.cshtml



Edit IT Request

Editable 23h 59m 49s

Requester Details — These fields are snapshotted at submission

Staff Name: core1hou

Designation:

Company: PSTW

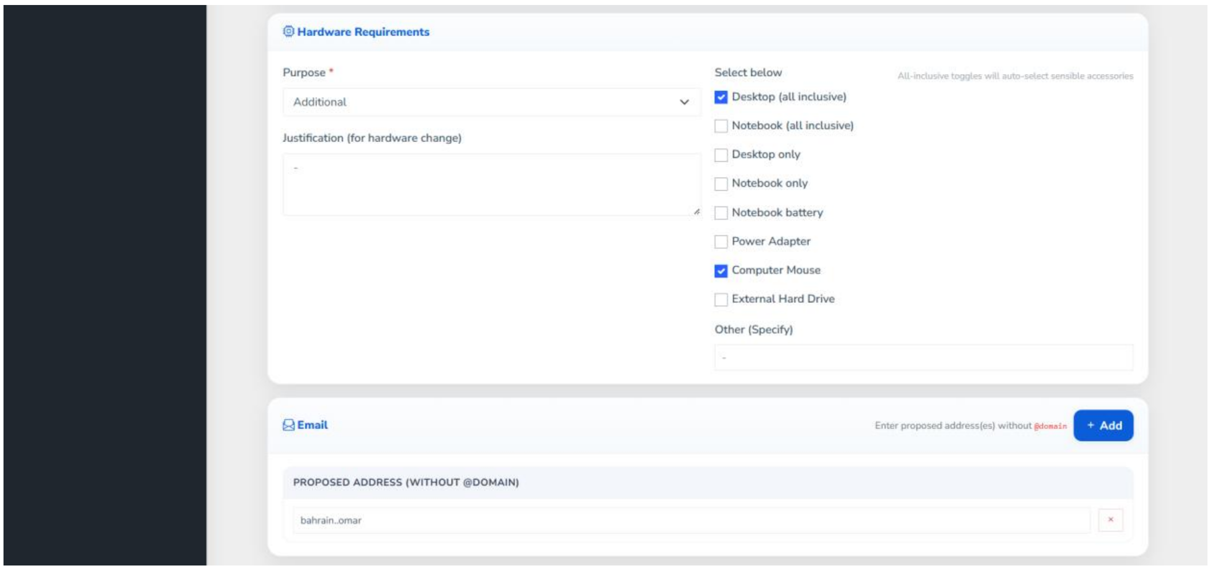
Div/Dept: PSTW AIR

Location:

Phone Ext:

Employment Status: --

Required Date *: 18/11/2025



Hardware Requirements

Purpose *: Additional

Justification (for hardware change): -

Select below — All-inclusive toggles will auto-select sensible accessories

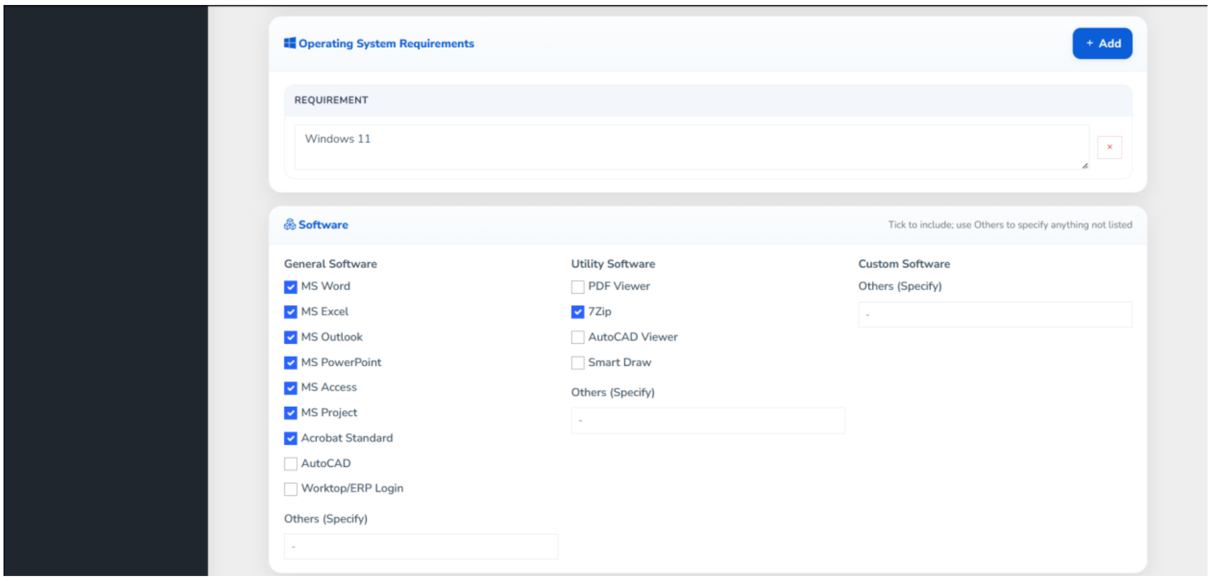
- ✓ Desktop (all inclusive)
- ☐ Notebook (all inclusive)
- ☐ Desktop only
- ☐ Notebook only
- ☐ Notebook battery
- ☐ Power Adapter
- ✓ Computer Mouse
- ☐ External Hard Drive

Other (Specify): -

Email — Enter proposed address(es) without @domain

PROPOSED ADDRESS (WITHOUT @DOMAIN)

bahrain_omar



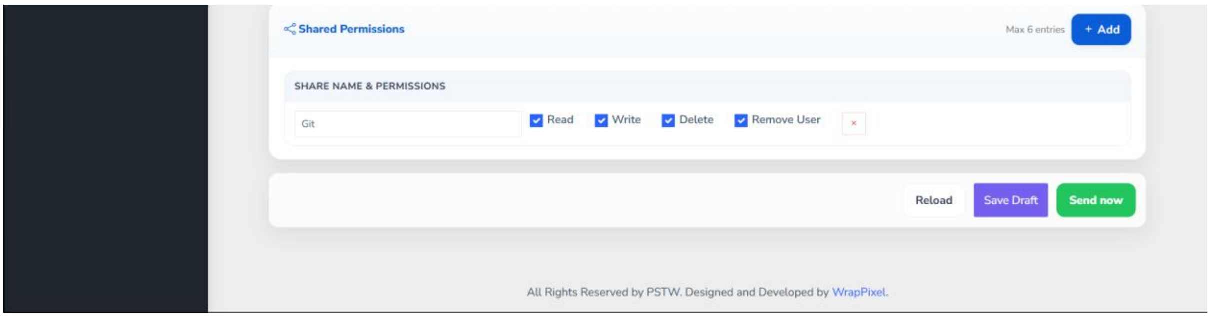
Operating System Requirements
+ Add

REQUIREMENT

Windows 11 ×

Software — Tick to include; use Others to specify anything not listed

General Software	Utility Software	Custom Software
✓ MS Word	☐ PDF Viewer	Others (Specify)
✓ MS Excel	✓ 7Zip	-
✓ MS Outlook	☐ AutoCAD Viewer	
✓ MS PowerPoint	☐ Smart Draw	
✓ MS Access	Others (Specify)	
✓ MS Project	-	
✓ Acrobat Standard		
☐ AutoCAD		
☐ Worktop/ERP Login		
Others (Specify)		
-		



Shared Permissions — Max 6 entries
+ Add

SHARE NAME & PERMISSIONS

Git ✓ Read ✓ Write ✓ Delete ✓ Remove User ×

Reload Save Draft Send now

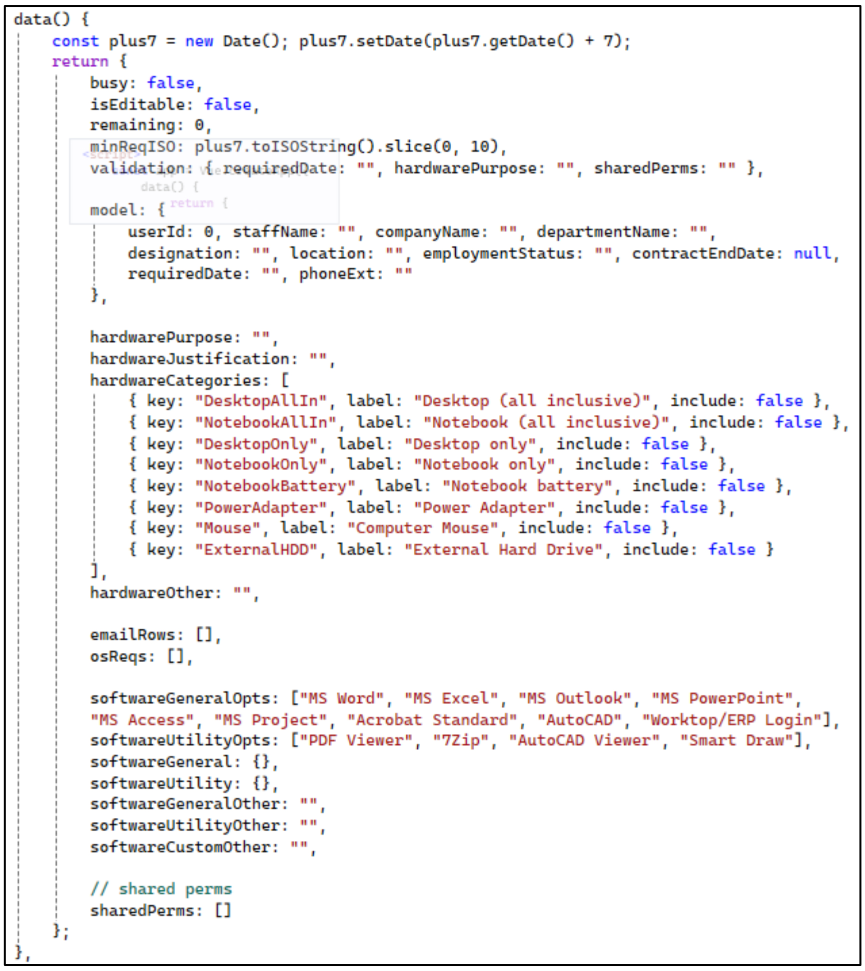
All Rights Reserved by PSTW. Designed and Developed by WrapPixel.

This file (Edit.cshtml) lets a requester reopen a saved IT request within a limited edit window, make changes, and either save again or submit to approvals. It shows a live countdown so the user knows exactly how much time is left before the form locks.

- ✓ **<div id="editApp">…</div>**
  - Purpose: Root container for the Vue app so all inputs and states are reactive.

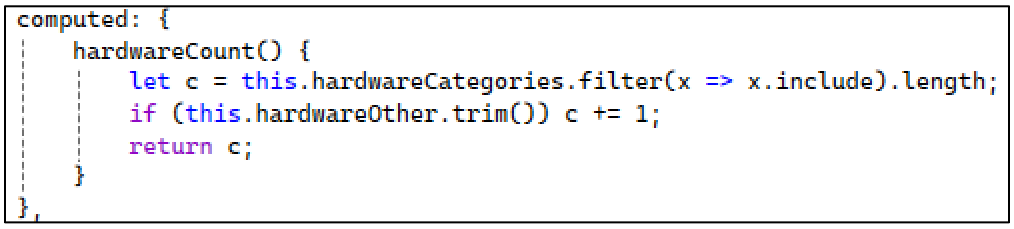
- ✓ **const app = Vue.createApp({...})**
  - Purpose: Starts the Vue instance and connects state, computed helpers, actions, and lifecycle.
- ✓ **data()**
  - **busy, isEditable, remaining, minReqISO, validation**: These variables drive the edit experience. busy guards buttons during network work, isEditable flips all inputs on or off, and remaining holds the server countdown in seconds. minReqISO pins the earliest required date and validation carries friendly field messages.

- **model { requester snapshot }**: Holds the person and organization details that were recorded with the original request. Some fields remain editable if the window is open, such as designation, location, phone extension, and required date.
- **hardwarePurpose, hardwareJustification, hardwareCategories, hardwareOther**: Represents the hardware section. Presets are toggled individually, purpose and justification explain the need, and an “Other” text box captures anything custom.
- **emailRows, osReqs**: Dynamic arrays for email local parts and operating system requirements. Each row maps cleanly to what the API expects when saving.
- **software groups and Others fields**: Two preset buckets, General and Utility, are tracked by simple dictionaries. Three “Others” inputs capture any software not covered by the checklists, including a Custom bucket.
- **sharedPerms**: Up to six entries for shared folders with four permission flags. Each row holds the share name plus Read, Write, Delete, and Remove user toggles.



```
data() {
    const plus7 = new Date(); plus7.setDate(plus7.getDate() + 7);
    return {
        busy: false,
        isEditable: false,
        remaining: 0,
        minReqISO: plus7.toISOString().slice(0, 10),
        validation: { requiredDate: "", hardwarePurpose: "", sharedPerms: "" },

        model: {
            userId: 0, staffName: "", companyName: "", departmentName: "",
            designation: "", location: "", employmentStatus: "", contractEndDate: null,
            requiredDate: "", phoneExt: ""
        },

        hardwarePurpose: "",
        hardwareJustification: "",
        hardwareCategories: [
            { key: "DesktopAllIn", label: "Desktop (all inclusive)", include: false },
            { key: "NotebookAllIn", label: "Notebook (all inclusive)", include: false },
            { key: "DesktopOnly", label: "Desktop only", include: false },
            { key: "NotebookOnly", label: "Notebook only", include: false },
            { key: "NotebookBattery", label: "Notebook battery", include: false },
            { key: "PowerAdapter", label: "Power Adapter", include: false },
            { key: "Mouse", label: "Computer Mouse", include: false },
            { key: "ExternalHDD", label: "External Hard Drive", include: false }
        ],
        hardwareOther: "",

        emailRows: [],
        osReqs: [],

        softwareGeneralOpts: ["MS Word", "MS Excel", "MS Outlook", "MS PowerPoint",
        "MS Access", "MS Project", "Acrobat Standard", "AutoCAD", "Worktop/ERP Login"],
        softwareUtilityOpts: ["PDF Viewer", "7Zip", "AutoCAD Viewer", "Smart Draw"],
        softwareGeneral: {},
        softwareUtility: {},
        softwareGeneralOther: "",
        softwareUtilityOther: "",
        softwareCustomOther: "",

        // shared perms
        sharedPerms: []
    };
},
```

- ✓ **computed**
  - **hardwareCount**: Counts how many hardware items are currently selected and includes one more if the "Other" box has text. The count feeds both validation and subtle chips in the UI.

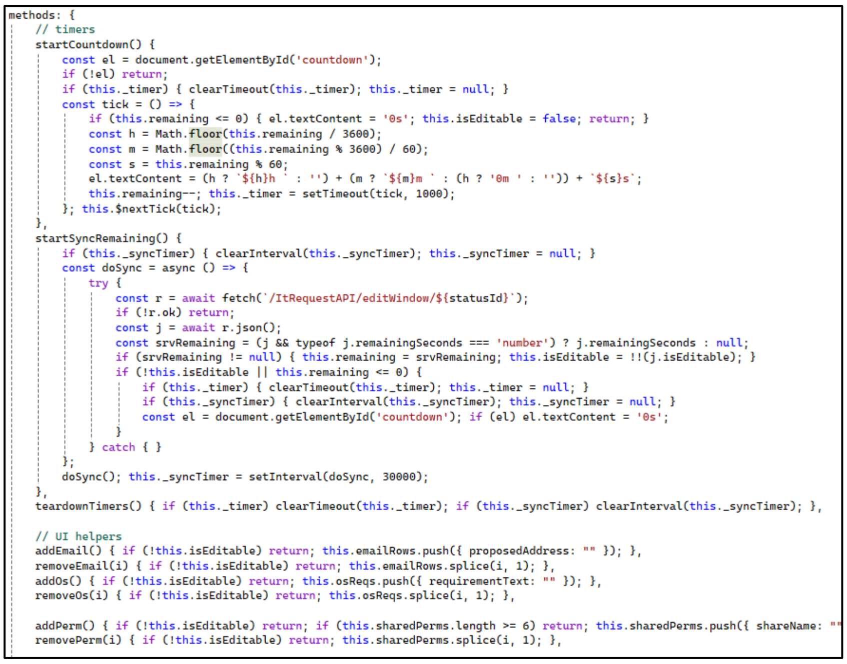


```
computed: {
    hardwareCount() {
        let c = this.hardwareCategories.filter(x => x.include).length;
        if (this.hardwareOther.trim()) c += 1;
        return c;
    }
},
```

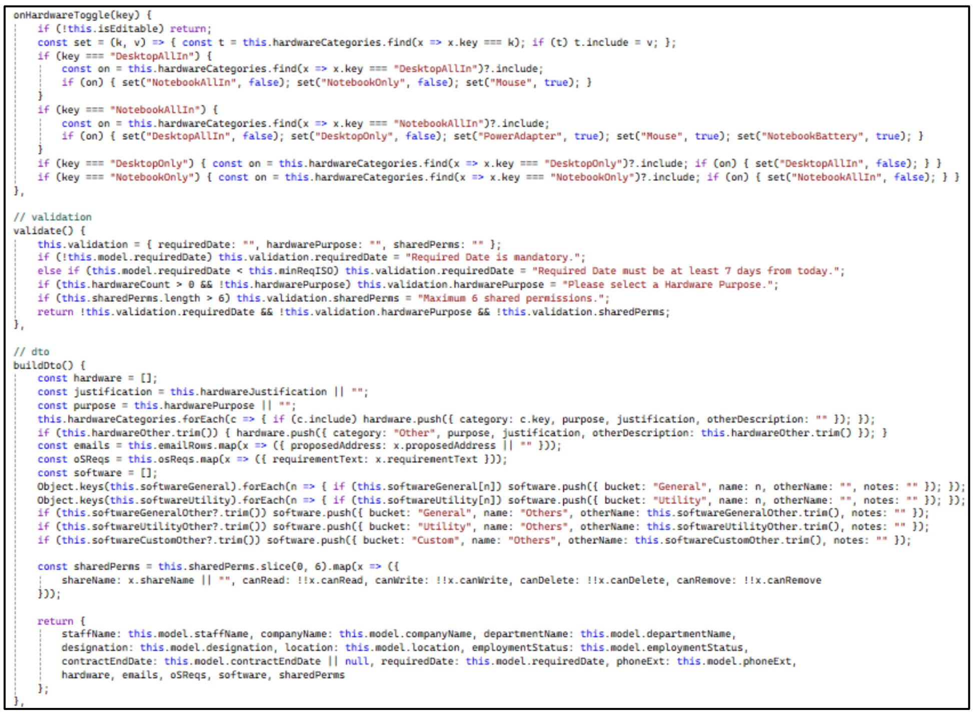
- ✓ **methods**
  - **startCountdown()**: Renders a second by second countdown into the small status box at the top. It formats hours, minutes, and seconds and stops itself once the time hits zero, which also flips isEditable to false.

- **startSyncRemaining()**: Polls the server every 30 seconds to stay in sync with the authoritative remaining time and the editable flag. If the server says the window has closed, it clears timers and sets the countdown to 0 seconds.
- **teardownTimers()**: Cleans up both the tick timer and the polling timer. This avoids orphaned timers when the user reloads or navigates away.
- **addEmail, removeEmail, addOs, removeOs:** Manage rows in the Email and OS sections. They respect isEditable and update arrays immediately so the UI stays responsive.
- **addPerm, removePerm**: Add a new shared permission row with safe defaults and delete the selected one. Addition stops at six to mirror the rule shown in the UI.
- **onHardwareToggle(key)**: Keeps hardware choices consistent. Desktop All Inclusive disables notebook sets and turns on accessories like Mouse, while Notebook All Inclusive disables desktop sets and turns on Power Adapter, Mouse, and Battery. The “Only” options also clear their related “All Inclusive” switch.
- **validate()**: Runs client checks before save or submit. It requires a required date that is at least seven days away, a hardware purpose whenever any hardware is selected, and a maximum of six shared permissions.
- **buildDto()**: Prepares the payload for the API in a single place. It flattens hardware toggles into objects with category, purpose, justification, and optional other description, maps email and OS to simple arrays, collects checked software by bucket while handling “Others,” and slices shared permissions to six.
- **load()**: Fetches the full request by statusId from /ItRequestAPI/request/{statusId} and hydrates all sections. It also reads the server provided isEditable and remainingSeconds, then starts the countdown and sync timers so the page reflects the true window.
- **save()**: Validates input, builds the DTO, and sends a PUT to /ItRequestAPI/edit/{statusId}. On success it refreshes remaining from the server response and restarts the countdown so the timer stays accurate.
- **sendNow()**: Validates, asks for a final confirmation, then posts to /ItRequestAPI/sendNow/{statusId}. If the server accepts, it shows a quick success message and navigates back to the My Requests dashboard.

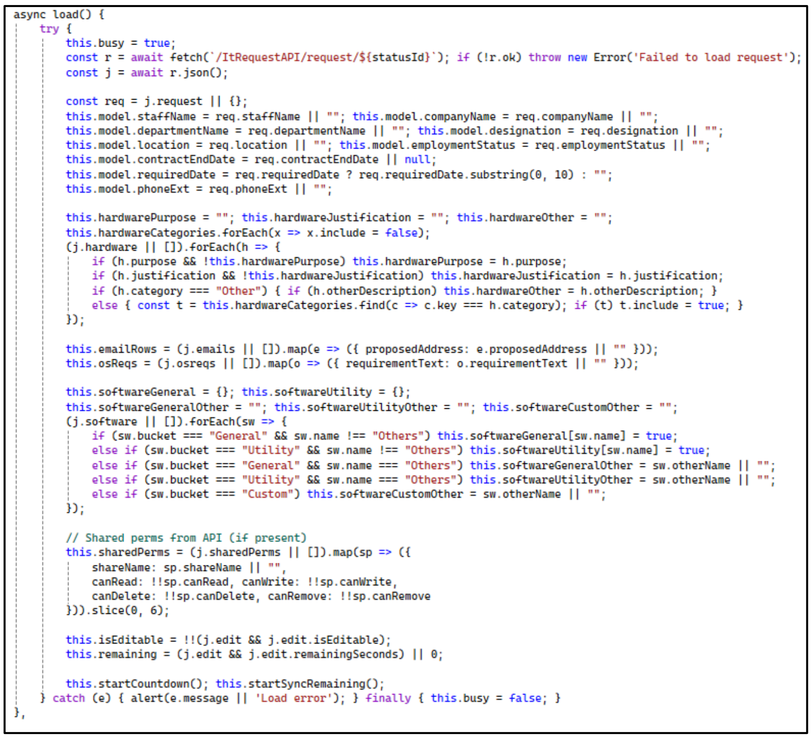
- **reload()**: Stops timers, reloads from the server, and restarts the countdown. Useful if the user suspects changes were made elsewhere or wants to reset the view.



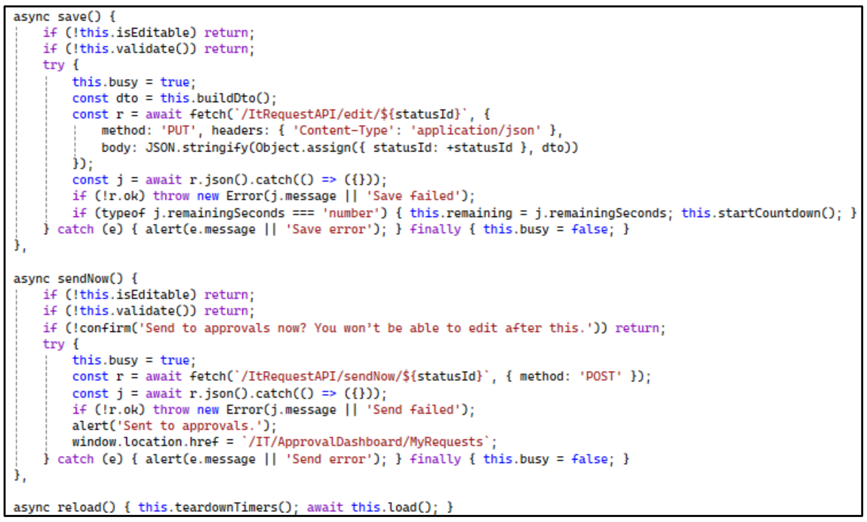
```
methods: {
    // timers
    startCountdown() {
        const el = document.getElementById('countdown');
        if (!el) return;
        if (this._timer) { clearTimeout(this._timer); this._timer = null; }
        const tick = () => {
            if (this.remaining <= 0) { el.textContent = '0s'; this.isEditable = false; return; }
            const h = Math.floor(this.remaining / 3600);
            const m = Math.floor((this.remaining % 3600) / 60);
            const s = this.remaining % 60;
            el.textContent = (h ? `${h}h ` : '') + (m ? `${m}m ` : (h ? '0m ' : '')) + `${s}s`;
            this.remaining--; this._timer = setTimeout(tick, 1000);
        }; this.$nextTick(tick);
    },
    startSyncRemaining() {
        if (this._syncTimer) { clearInterval(this._syncTimer); this._syncTimer = null; }
        const doSync = async () => {
            try {
                const r = await fetch(`/ItRequestAPI/editWindow/${statusId}`);
                if (!r.ok) return;
                const j = await r.json();
                const srvRemaining = (j && typeof j.remainingSeconds === 'number') ? j.remainingSeconds : null;
                if (srvRemaining != null) { this.remaining = srvRemaining; this.isEditable = !!(j.isEditable); }
                if (!this.isEditable || this.remaining <= 0) {
                    if (this._timer) { clearTimeout(this._timer); this._timer = null; }
                    if (this._syncTimer) { clearInterval(this._syncTimer); this._syncTimer = null; }
                    const el = document.getElementById('countdown'); if (el) el.textContent = '0s';
                }
            } catch { }
        };
        doSync(); this._syncTimer = setInterval(doSync, 30000);
    },
    teardownTimers() { if (this._timer) clearTimeout(this._timer); if (this._syncTimer) clearInterval(this._syncTimer); },

    // UI helpers
    addEmail() { if (!this.isEditable) return; this.emailRows.push({ proposedAddress: "" }); },
    removeEmail(i) { if (!this.isEditable) return; this.emailRows.splice(i, 1); },
    addOs() { if (!this.isEditable) return; this.osReqs.push({ requirementText: "" }); },
    removeOs(i) { if (!this.isEditable) return; this.osReqs.splice(i, 1); },

    addPerm() { if (!this.isEditable) return; if (this.sharedPerms.length >= 6) return; this.sharedPerms.push({ shareName: ""
    removePerm(i) { if (!this.isEditable) return; this.sharedPerms.splice(i, 1); },
```



```
onHardwareToggle(key) {
    if (!this.isEditable) return;
    const set = (k, v) => { const t = this.hardwareCategories.find(x => x.key === k); if (t) t.include = v; };
    if (key === "DesktopAllIn") {
        const on = this.hardwareCategories.find(x => x.key === "DesktopAllIn")?.include;
        if (on) { set("NotebookAllIn", false); set("NotebookOnly", false); set("Mouse", true); }
    }
    if (key === "NotebookAllIn") {
        const on = this.hardwareCategories.find(x => x.key === "NotebookAllIn")?.include;
        if (on) { set("DesktopAllIn", false); set("DesktopOnly", false); set("PowerAdapter", true); set("Mouse", true); set("NotebookBattery", true); }
    }
    if (key === "DesktopOnly") { const on = this.hardwareCategories.find(x => x.key === "DesktopOnly")?.include; if (on) { set("DesktopAllIn", false); } }
    if (key === "NotebookOnly") { const on = this.hardwareCategories.find(x => x.key === "NotebookOnly")?.include; if (on) { set("NotebookAllIn", false); } }
},

// validation
validate() {
    this.validation = { requiredDate: "", hardwarePurpose: "", sharedPerms: "" };
    if (!this.model.requiredDate) this.validation.requiredDate = "Required Date is mandatory.";
    else if (this.model.requiredDate < this.minReqISO) this.validation.requiredDate = "Required Date must be at least 7 days from today.";
    if (this.hardwareCount > 0 && !this.hardwarePurpose) this.validation.hardwarePurpose = "Please select a Hardware Purpose.";
    if (this.sharedPerms.length > 6) this.validation.sharedPerms = "Maximum 6 shared permissions.";
    return !this.validation.requiredDate && !this.validation.hardwarePurpose && !this.validation.sharedPerms;
},

// dto
buildDto() {
    const hardware = [];
    const justification = this.hardwareJustification || "";
    const purpose = this.hardwarePurpose || "";
    this.hardwareCategories.forEach(c => { if (c.include) hardware.push({ category: c.key, purpose, justification, otherDescription: "" }); });
    if (this.hardwareOther.trim()) { hardware.push({ category: "Other", purpose, justification, otherDescription: this.hardwareOther.trim() }); }
    const emails = this.emailRows.map(x => ({ proposedAddress: x.proposedAddress || "" }));
    const oSReqs = this.osReqs.map(x => ({ requirementText: x.requirementText }));
    const software = [];
    Object.keys(this.softwareGeneral).forEach(n => { if (this.softwareGeneral[n]) software.push({ bucket: "General", name: n, otherName: "", notes: "" }); });
    Object.keys(this.softwareUtility).forEach(n => { if (this.softwareUtility[n]) software.push({ bucket: "Utility", name: n, otherName: "", notes: "" }); });
    if (this.softwareGeneralOther?.trim()) software.push({ bucket: "General", name: "Others", otherName: this.softwareGeneralOther.trim(), notes: "" });
    if (this.softwareUtilityOther?.trim()) software.push({ bucket: "Utility", name: "Others", otherName: this.softwareUtilityOther.trim(), notes: "" });
    if (this.softwareCustomOther?.trim()) software.push({ bucket: "Custom", name: "Others", otherName: this.softwareCustomOther.trim(), notes: "" });

    const sharedPerms = this.sharedPerms.slice(0, 6).map(x => ({
        shareName: x.shareName || "", canRead: !!x.canRead, canWrite: !!x.canWrite, canDelete: !!x.canDelete, canRemove: !!x.canRemove
    }));

    return {
        staffName: this.model.staffName, companyName: this.model.companyName, departmentName: this.model.departmentName,
        designation: this.model.designation, location: this.model.location, employmentStatus: this.model.employmentStatus,
        contractEndDate: this.model.contractEndDate || null, requiredDate: this.model.requiredDate, phoneExt: this.model.phoneExt,
        hardware, emails, oSReqs, software, sharedPerms
    };
},
```

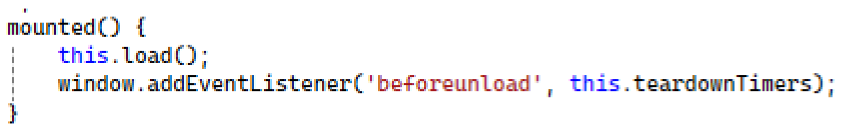


```
async load() {
    try {
        this.busy = true;
        const r = await fetch(`/ItRequestAPI/request/${statusId}`); if (!r.ok) throw new Error('Failed to load request');
        const j = await r.json();

        const req = j.request || {};
        this.model.staffName = req.staffName || ""; this.model.companyName = req.companyName || "";
        this.model.departmentName = req.departmentName || ""; this.model.designation = req.designation || "";
        this.model.location = req.location || ""; this.model.employmentStatus = req.employmentStatus || "";
        this.model.contractEndDate = req.contractEndDate || null;
        this.model.requiredDate = req.requiredDate ? req.requiredDate.substring(0, 10) : "";
        this.model.phoneExt = req.phoneExt || "";

        this.hardwarePurpose = ""; this.hardwareJustification = ""; this.hardwareOther = "";
        this.hardwareCategories.forEach(x => x.include = false);
        (j.hardware || []).forEach(h => {
            if (h.purpose && !this.hardwarePurpose) this.hardwarePurpose = h.purpose;
            if (h.justification && !this.hardwareJustification) this.hardwareJustification = h.justification;
            if (h.category === "Other") { if (h.otherDescription) this.hardwareOther = h.otherDescription; }
            else { const t = this.hardwareCategories.find(c => c.key === h.category); if (t) t.include = true; }
        });

        this.emailRows = (j.emails || []).map(e => ({ proposedAddress: e.proposedAddress || "" }));
        this.osReqs = (j.osreqs || []).map(o => ({ requirementText: o.requirementText || "" }));

        this.softwareGeneral = {}; this.softwareUtility = {};
        this.softwareGeneralOther = ""; this.softwareUtilityOther = ""; this.softwareCustomOther = "";
        (j.software || []).forEach(sw => {
            if (sw.bucket === "General" && sw.name !== "Others") this.softwareGeneral[sw.name] = true;
            else if (sw.bucket === "Utility" && sw.name !== "Others") this.softwareUtility[sw.name] = true;
            else if (sw.bucket === "General" && sw.name === "Others") this.softwareGeneralOther = sw.otherName || "";
            else if (sw.bucket === "Utility" && sw.name === "Others") this.softwareUtilityOther = sw.otherName || "";
            else if (sw.bucket === "Custom") this.softwareCustomOther = sw.otherName || "";
        });

        // Shared perms from API (if present)
        this.sharedPerms = (j.sharedPerms || []).map(sp => ({
            shareName: sp.shareName || "",
            canRead: !!sp.canRead, canWrite: !!sp.canWrite,
            canDelete: !!sp.canDelete, canRemove: !!sp.canRemove
        })).slice(0, 6);

        this.isEditable = !!(j.edit && j.edit.isEditable);
        this.remaining = (j.edit && j.edit.remainingSeconds) || 0;

        this.startCountdown(); this.startSyncRemaining();
    } catch (e) { alert(e.message || 'Load error'); } finally { this.busy = false; }
},
```



```
async save() {
    if (!this.isEditable) return;
    if (!this.validate()) return;
    try {
        this.busy = true;
        const dto = this.buildDto();
        const r = await fetch(`/ItRequestAPI/edit/${statusId}`, {
            method: 'PUT', headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify(Object.assign({ statusId: +statusId }, dto))
        });
        const j = await r.json().catch(() => ({}));
        if (!r.ok) throw new Error(j.message || 'Save failed');
        if (typeof j.remainingSeconds === 'number') { this.remaining = j.remainingSeconds; this.startCountdown(); }
    } catch (e) { alert(e.message || 'Save error'); } finally { this.busy = false; }
},

async sendNow() {
    if (!this.isEditable) return;
    if (!this.validate()) return;
    if (!confirm('Send to approvals now? You won't be able to edit after this.')) return;
    try {
        this.busy = true;
        const r = await fetch(`/ItRequestAPI/sendNow/${statusId}`, { method: 'POST' });
        const j = await r.json().catch(() => ({}));
        if (!r.ok) throw new Error(j.message || 'Send failed');
        alert('Sent to approvals.');
        window.location.href = `/IT/ApprovalDashboard/MyRequests`;
    } catch (e) { alert(e.message || 'Send error'); } finally { this.busy = false; }
},

async reload() { this.teardownTimers(); await this.load(); }
```

- ✓ mounted
    - **mounted()**: Calls load() immediately to populate the form and to pull the edit window state from the server. It also registers a beforeunload handler to tear down timers cleanly when the user leaves the page.



```
mounted() {
    this.load();
    window.addEventListener('beforeunload', this.teardownTimers);
}
```

- ITRequestAPI (Edit.cshtml)
  - **[HttpGet("request/{statusId}")] — Used by:** Edit.cshtml

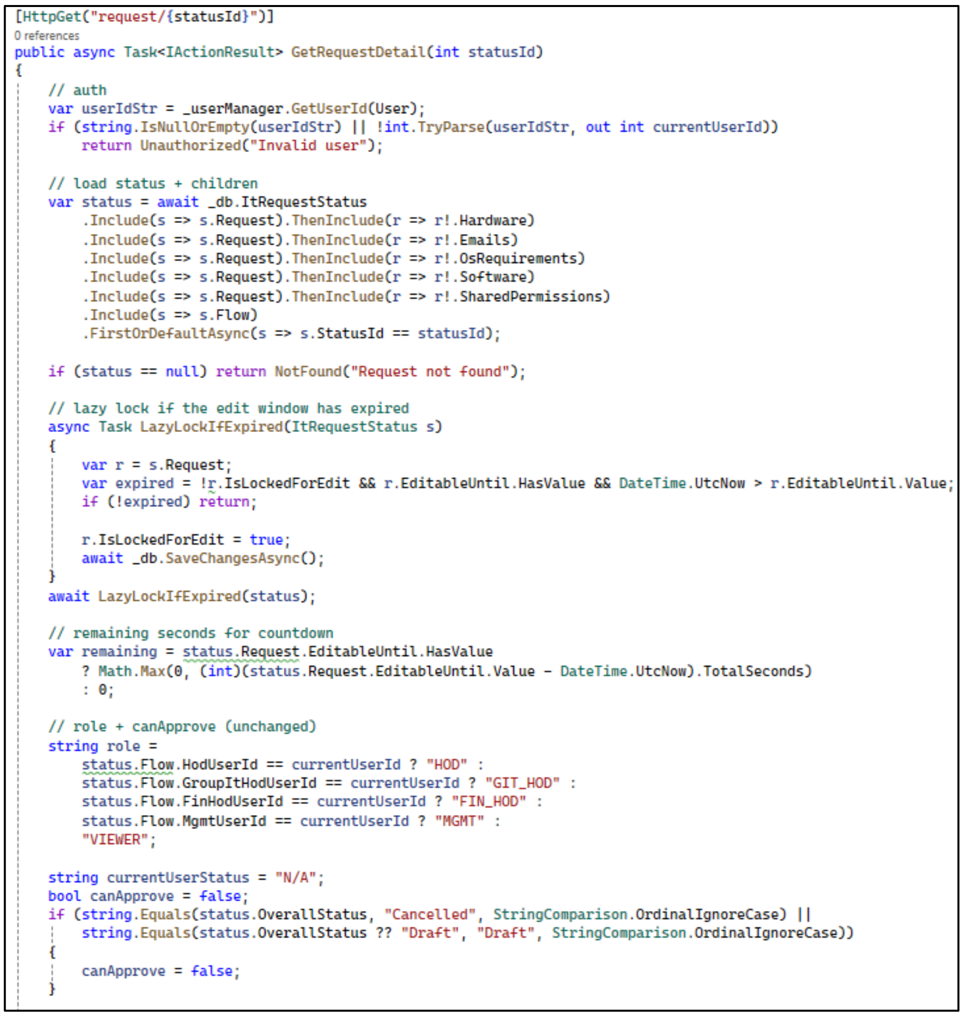
- **Purpose:**
  This endpoint loads everything the Edit page needs the full request details, its approval status, and whether it's still within the editable time window. Basically, when you open an existing request to view or update it, this is the API that supplies all the data.

- **Key functionalities:**
  - **User validation:**
    It first checks who's currently logged in and makes sure the user ID is valid. If not, it immediately returns an unauthorized message.
  - **Load request and all related data:**
    It fetches the full ItRequestStatus record by statusId, including its related tables:
    - Main request info (Section A)
    - Hardware, Emails, OS Requirements, Software, Shared Permissions
    - Approval Flow (to check roles and statuses)
      If the record doesn't exist, it returns a "Request not found" error.
  - **Auto-lock check:**
    There's a small built-in check that runs every time this endpoint is called.
    If the request is still marked editable but its edit window has expired (based on EditableUntil), the system will **auto-lock** it right away and save the update.
    This prevents users from editing an old request that should already be locked.
  - **Edit countdown timer:**
    It calculates how many seconds are left before the edit window closes. This value is sent to the frontend so the Edit page can display a live countdown (e.g., "You can still edit for another 3 hours").
  - **Determine user's role and approval rights:**
    Based on the approval flow, the system checks whether the current user is the

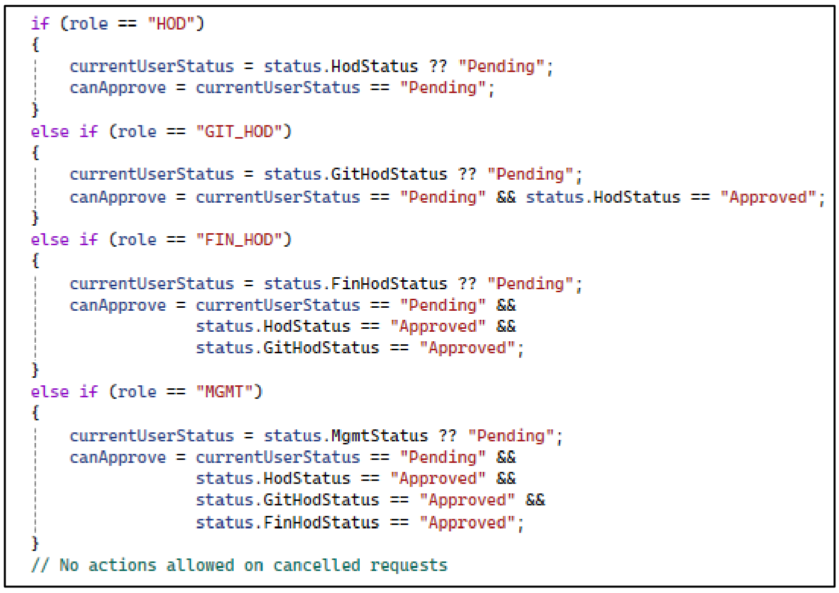
HOD, Group IT HOD, Finance HOD, or Management.
Then it determines two things:

- currentUserStatus — what stage the user is at (e.g., “Pending”, “Approved”, “Rejected”)
- canApprove — whether the user can take action right now.
  For example:
  - HOD can act if their status is still Pending.
  - Group IT HOD can only act after HOD has approved.
  - Finance HOD follows after both HOD and Group IT HOD approvals.
  - Management only steps in after everyone else has approved.

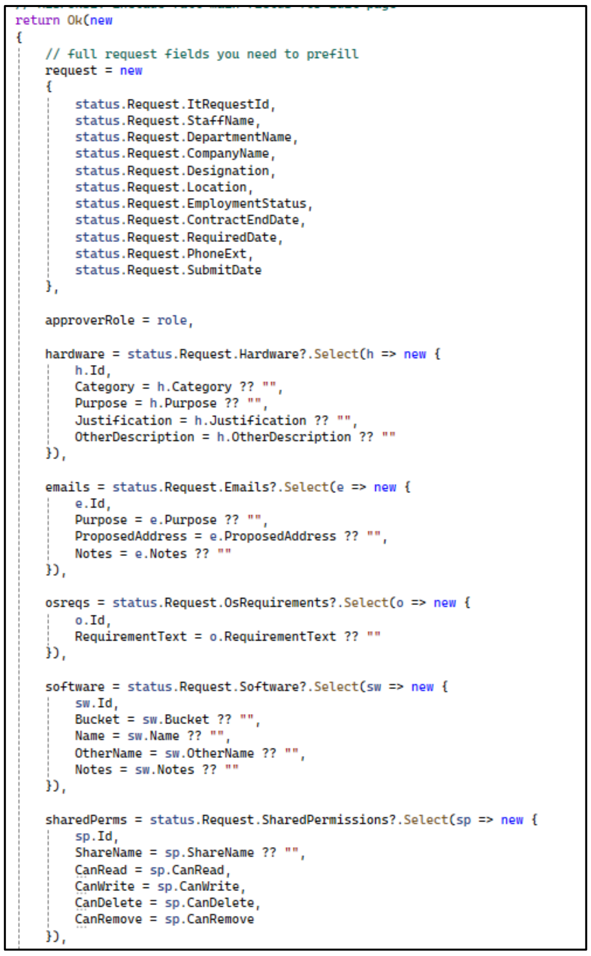
- **Response content:**
  Once all that’s done, the endpoint returns a structured JSON with everything needed to prefill the Edit page — all form fields, sub-lists (hardware, software, etc.), each approver’s status, and edit window info like remainingSeconds and isEditable.



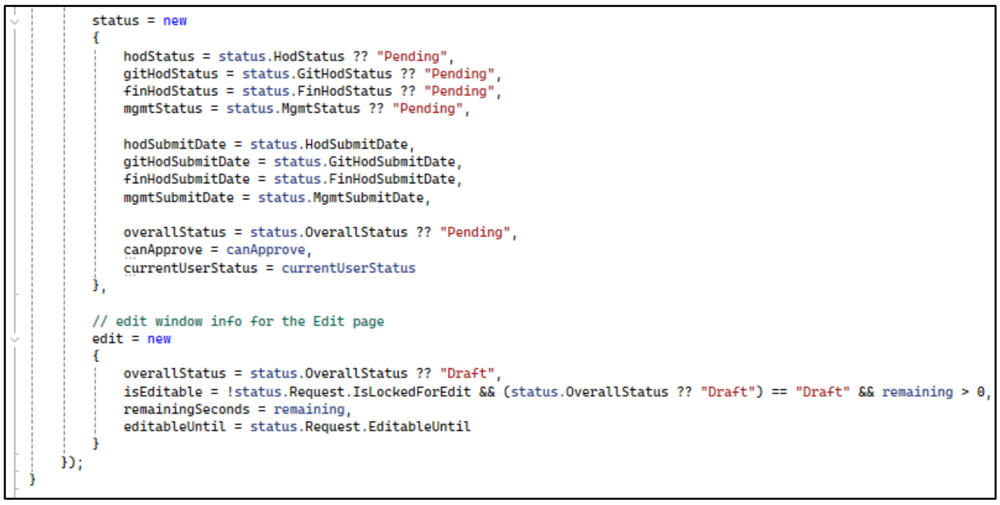
```
[HttpGet("request/{statusId}")]
0 references
public async Task<IActionResult> GetRequestDetail(int statusId)
{
    // auth
    var userIdStr = _userManager.GetUserId(User);
    if (string.IsNullOrEmpty(userIdStr) || !int.TryParse(userIdStr, out int currentUserId))
        return Unauthorized("Invalid user");

    // load status + children
    var status = await _db.ItRequestStatus
        .Include(s => s.Request).ThenInclude(r => r!.Hardware)
        .Include(s => s.Request).ThenInclude(r => r!.Emails)
        .Include(s => s.Request).ThenInclude(r => r!.OsRequirements)
        .Include(s => s.Request).ThenInclude(r => r!.Software)
        .Include(s => s.Request).ThenInclude(r => r!.SharedPermissions)
        .Include(s => s.Flow)
        .FirstOrDefaultAsync(s => s.StatusId == statusId);

    if (status == null) return NotFound("Request not found");

    // lazy lock if the edit window has expired
    async Task LazyLockIfExpired(ItRequestStatus s)
    {
        var r = s.Request;
        var expired = !r.IsLockedForEdit && r.EditableUntil.HasValue && DateTime.UtcNow > r.EditableUntil.Value;
        if (!expired) return;

        r.IsLockedForEdit = true;
        await _db.SaveChangesAsync();
    }
    await LazyLockIfExpired(status);

    // remaining seconds for countdown
    var remaining = status.Request.EditableUntil.HasValue
        ? Math.Max(0, (int)(status.Request.EditableUntil.Value - DateTime.UtcNow).TotalSeconds)
        : 0;

    // role + canApprove (unchanged)
    string role =
        status.Flow.HodUserId == currentUserId ? "HOD" :
        status.Flow.GroupItHodUserId == currentUserId ? "GIT_HOD" :
        status.Flow.FinHodUserId == currentUserId ? "FIN_HOD" :
        status.Flow.MgmtUserId == currentUserId ? "MGMT" :
        "VIEWER";

    string currentUserStatus = "N/A";
    bool canApprove = false;
    if (string.Equals(status.OverallStatus, "Cancelled", StringComparison.OrdinalIgnoreCase) ||
        string.Equals(status.OverallStatus ?? "Draft", "Draft", StringComparison.OrdinalIgnoreCase))
    {
        canApprove = false;
    }
```



```
if (role == "HOD")
{
    currentUserStatus = status.HodStatus ?? "Pending";
    canApprove = currentUserStatus == "Pending";
}
else if (role == "GIT_HOD")
{
    currentUserStatus = status.GitHodStatus ?? "Pending";
    canApprove = currentUserStatus == "Pending" && status.HodStatus == "Approved";
}
else if (role == "FIN_HOD")
{
    currentUserStatus = status.FinHodStatus ?? "Pending";
    canApprove = currentUserStatus == "Pending" &&
                 status.HodStatus == "Approved" &&
                 status.GitHodStatus == "Approved";
}
else if (role == "MGMT")
{
    currentUserStatus = status.MgmtStatus ?? "Pending";
    canApprove = currentUserStatus == "Pending" &&
                 status.HodStatus == "Approved" &&
                 status.GitHodStatus == "Approved" &&
                 status.FinHodStatus == "Approved";
}
// No actions allowed on cancelled requests
```



```
return Ok(new
{
    // full request fields you need to prefill
    request = new
    {
        status.Request.ItRequestId,
        status.Request.StaffName,
        status.Request.DepartmentName,
        status.Request.CompanyName,
        status.Request.Designation,
        status.Request.Location,
        status.Request.EmploymentStatus,
        status.Request.ContractEndDate,
        status.Request.RequiredDate,
        status.Request.PhoneExt,
        status.Request.SubmitDate
    },

    approverRole = role,

    hardware = status.Request.Hardware?.Select(h => new {
        h.Id,
        Category = h.Category ?? "",
        Purpose = h.Purpose ?? "",
        Justification = h.Justification ?? "",
        OtherDescription = h.OtherDescription ?? ""
    }),

    emails = status.Request.Emails?.Select(e => new {
        e.Id,
        Purpose = e.Purpose ?? "",
        ProposedAddress = e.ProposedAddress ?? "",
        Notes = e.Notes ?? ""
    }),

    osreqs = status.Request.OsRequirements?.Select(o => new {
        o.Id,
        RequirementText = o.RequirementText ?? ""
    }),

    software = status.Request.Software?.Select(sw => new {
        sw.Id,
        Bucket = sw.Bucket ?? "",
        Name = sw.Name ?? "",
        OtherName = sw.OtherName ?? "",
        Notes = sw.Notes ?? ""
    }),

    sharedPerms = status.Request.SharedPermissions?.Select(sp => new {
        sp.Id,
        ShareName = sp.ShareName ?? "",
        CanRead = sp.CanRead,
        CanWrite = sp.CanWrite,
        CanDelete = sp.CanDelete,
        CanRemove = sp.CanRemove
    }),
```



```
        status = new
        {
            hodStatus = status.HodStatus ?? "Pending",
            gitHodStatus = status.GitHodStatus ?? "Pending",
            finHodStatus = status.FinHodStatus ?? "Pending",
            mgmtStatus = status.MgmtStatus ?? "Pending",

            hodSubmitDate = status.HodSubmitDate,
            gitHodSubmitDate = status.GitHodSubmitDate,
            finHodSubmitDate = status.FinHodSubmitDate,
            mgmtSubmitDate = status.MgmtSubmitDate,

            overallStatus = status.OverallStatus ?? "Pending",
            canApprove = canApprove,
            currentUserStatus = currentUserStatus
        },

        // edit window info for the Edit page
        edit = new
        {
            overallStatus = status.OverallStatus ?? "Draft",
            isEditable = !status.Request.IsLockedForEdit && (status.OverallStatus ?? "Draft") == "Draft" && remaining > 0,
            remainingSeconds = remaining,
            editableUntil = status.Request.EditableUntil
        }
    });
}
```

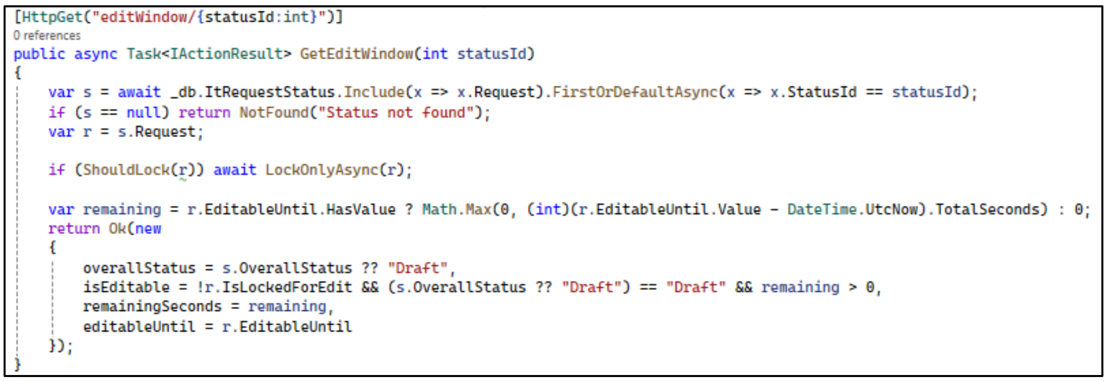
- **[HttpGet("editWindow/{statusId:int}")]** — **Used by:** Edit.cshtml

- **Purpose:**

  This endpoint checks whether the IT request is still within its editable time window.

It’s usually called when the Edit page loads or after saving, to update the countdown timer and determine if the user can still make changes.

- **Key functionalities:**
  - It loads the ItRequestStatus and its linked Request by statusId.
  - If the request no longer exists, it returns a simple “Status not found.”
  - It runs a background check to see if the edit window has expired. If the window has passed, the request automatically locks itself by calling LockOnlyAsync().
  - It calculates how many seconds remain before the edit window closes.
  - The response tells the frontend three things:
    1. The overall status (Draft, Pending, etc.)
    2. Whether it’s still editable (isEditable = true/false)
    3. The countdown in seconds (remainingSeconds) so the UI can display a live timer



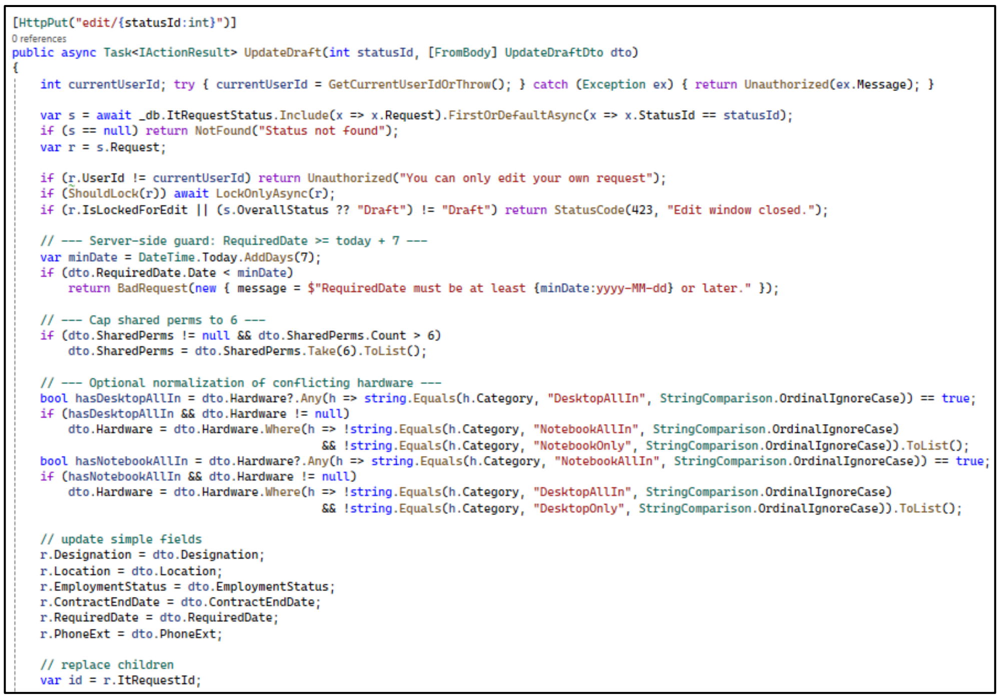
```
[HttpGet("editWindow/{statusId:int}")]
0 references
public async Task<IActionResult> GetEditWindow(int statusId)
{
    var s = await _db.ItRequestStatus.Include(x => x.Request).FirstOrDefaultAsync(x => x.StatusId == statusId);
    if (s == null) return NotFound("Status not found");
    var r = s.Request;

    if (ShouldLock(r)) await LockOnlyAsync(r);

    var remaining = r.EditableUntil.HasValue ? Math.Max(0, (int)(r.EditableUntil.Value - DateTime.UtcNow).TotalSeconds) : 0;
    return Ok(new
    {
        overallStatus = s.OverallStatus ?? "Draft",
        isEditable = !r.IsLockedForEdit && (s.OverallStatus ?? "Draft") == "Draft" && remaining > 0,
        remainingSeconds = remaining,
        editableUntil = r.EditableUntil
    });
}
```

▪ **[HttpPut("edit/{statusId:int}")] — Used by:** Edit.cshtml (Save Draft button)

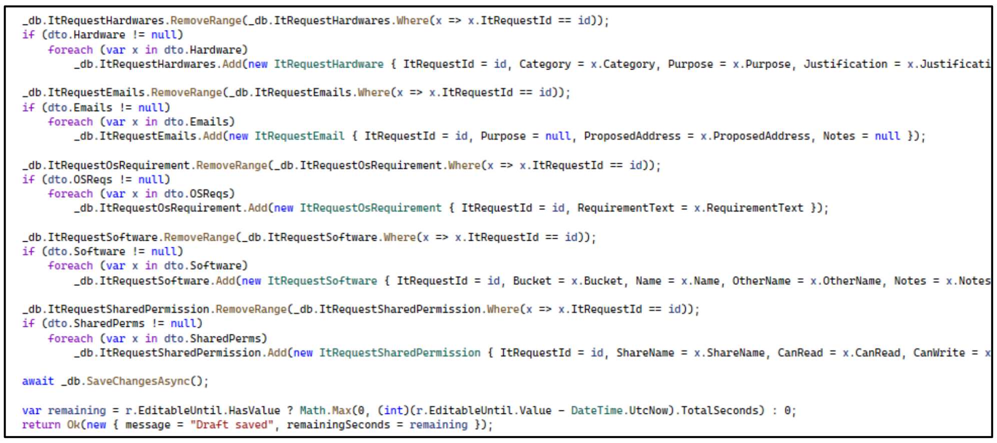
- **Purpose:**
  This endpoint updates an existing IT request that’s still in **Draft** mode. It’s used when the requester makes edits during their allowed window and clicks “Save Draft.”
- **Key functionalities:**
  - **User validation:**
    Confirms the logged-in user and ensures they’re the original creator of the

request. If someone else tries to edit, it returns “You can only edit your own request.”

- **Edit window check:**
  Before updating, it revalidates the edit window. If it’s expired or already locked, it blocks the update with a “423 Edit window closed.” message.
- **Server-side validation:**
  Makes sure the new RequiredDate is still at least 7 days from today.
  Also limits shared permissions to 6 entries max.
- **Hardware cleanup:**
  Prevents conflicting selections by removing notebook-related options if “Desktop All-In” is chosen (and vice versa).
- **Updating fields:**
  Refreshes simple request info like designation, location, employment status, required date, and phone extension.
- **Replacing all sublists:**
  It clears and repopulates all related tables (hardware, emails, OS requirements, software, shared permissions) with the new data from the frontend.
- **Save confirmation:**
  After saving changes, it returns a success message and updates the remaining edit time in seconds, so the Edit page can keep the countdown accurate.



```
[HttpPut("edit/{statusId:int}")]
0 references
public async Task<IActionResult> UpdateDraft(int statusId, [FromBody] UpdateDraftDto dto)
{
    int currentUserId; try { currentUserId = GetCurrentUserIdOrThrow(); } catch (Exception ex) { return Unauthorized(ex.Message); }

    var s = await _db.ItRequestStatus.Include(x => x.Request).FirstOrDefaultAsync(x => x.StatusId == statusId);
    if (s == null) return NotFound("Status not found");
    var r = s.Request;

    if (r.UserId != currentUserId) return Unauthorized("You can only edit your own request");
    if (ShouldLock(r)) await LockOnlyAsync(r);
    if (r.IsLockedForEdit || (s.OverallStatus ?? "Draft") != "Draft") return StatusCode(423, "Edit window closed.");

    // --- Server-side guard: RequiredDate >= today + 7 ---
    var minDate = DateTime.Today.AddDays(7);
    if (dto.RequiredDate.Date < minDate)
        return BadRequest(new { message = $"RequiredDate must be at least {minDate:yyyy-MM-dd} or later." });

    // --- Cap shared perms to 6 ---
    if (dto.SharedPerms != null && dto.SharedPerms.Count > 6)
        dto.SharedPerms = dto.SharedPerms.Take(6).ToList();

    // --- Optional normalization of conflicting hardware ---
    bool hasDesktopAllIn = dto.Hardware?.Any(h => string.Equals(h.Category, "DesktopAllIn", StringComparison.OrdinalIgnoreCase)) == true;
    if (hasDesktopAllIn && dto.Hardware != null)
        dto.Hardware = dto.Hardware.Where(h => !string.Equals(h.Category, "NotebookAllIn", StringComparison.OrdinalIgnoreCase)
                                           && !string.Equals(h.Category, "NotebookOnly", StringComparison.OrdinalIgnoreCase)).ToList();
    bool hasNotebookAllIn = dto.Hardware?.Any(h => string.Equals(h.Category, "NotebookAllIn", StringComparison.OrdinalIgnoreCase)) == true;
    if (hasNotebookAllIn && dto.Hardware != null)
        dto.Hardware = dto.Hardware.Where(h => !string.Equals(h.Category, "DesktopAllIn", StringComparison.OrdinalIgnoreCase)
                                           && !string.Equals(h.Category, "DesktopOnly", StringComparison.OrdinalIgnoreCase)).ToList();

    // update simple fields
    r.Designation = dto.Designation;
    r.Location = dto.Location;
    r.EmploymentStatus = dto.EmploymentStatus;
    r.ContractEndDate = dto.ContractEndDate;
    r.RequiredDate = dto.RequiredDate;
    r.PhoneExt = dto.PhoneExt;

    // replace children
    var id = r.ItRequestId;
```



```
_db.ItRequestHardwares.RemoveRange(_db.ItRequestHardwares.Where(x => x.ItRequestId == id));
if (dto.Hardware != null)
    foreach (var x in dto.Hardware)
        _db.ItRequestHardwares.Add(new ItRequestHardware { ItRequestId = id, Category = x.Category, Purpose = x.Purpose, Justification = x.Justificati

_db.ItRequestEmails.RemoveRange(_db.ItRequestEmails.Where(x => x.ItRequestId == id));
if (dto.Emails != null)
    foreach (var x in dto.Emails)
        _db.ItRequestEmails.Add(new ItRequestEmail { ItRequestId = id, Purpose = null, ProposedAddress = x.ProposedAddress, Notes = null });

_db.ItRequestOsRequirement.RemoveRange(_db.ItRequestOsRequirement.Where(x => x.ItRequestId == id));
if (dto.OSReqs != null)
    foreach (var x in dto.OSReqs)
        _db.ItRequestOsRequirement.Add(new ItRequestOsRequirement { ItRequestId = id, RequirementText = x.RequirementText });

_db.ItRequestSoftware.RemoveRange(_db.ItRequestSoftware.Where(x => x.ItRequestId == id));
if (dto.Software != null)
    foreach (var x in dto.Software)
        _db.ItRequestSoftware.Add(new ItRequestSoftware { ItRequestId = id, Bucket = x.Bucket, Name = x.Name, OtherName = x.OtherName, Notes = x.Notes

_db.ItRequestSharedPermission.RemoveRange(_db.ItRequestSharedPermission.Where(x => x.ItRequestId == id));
if (dto.SharedPerms != null)
    foreach (var x in dto.SharedPerms)
        _db.ItRequestSharedPermission.Add(new ItRequestSharedPermission { ItRequestId = id, ShareName = x.ShareName, CanRead = x.CanRead, CanWrite = x

await _db.SaveChangesAsync();

var remaining = r.EditableUntil.HasValue ? Math.Max(0, (int)(r.EditableUntil.Value - DateTime.UtcNow).TotalSeconds) : 0;
return Ok(new { message = "Draft saved", remainingSeconds = remaining });
```

- **[HttpPost("sendNow/{statusId:int}")] — Used by:** Edit.cshtml (Send Now button)

- **Purpose:**

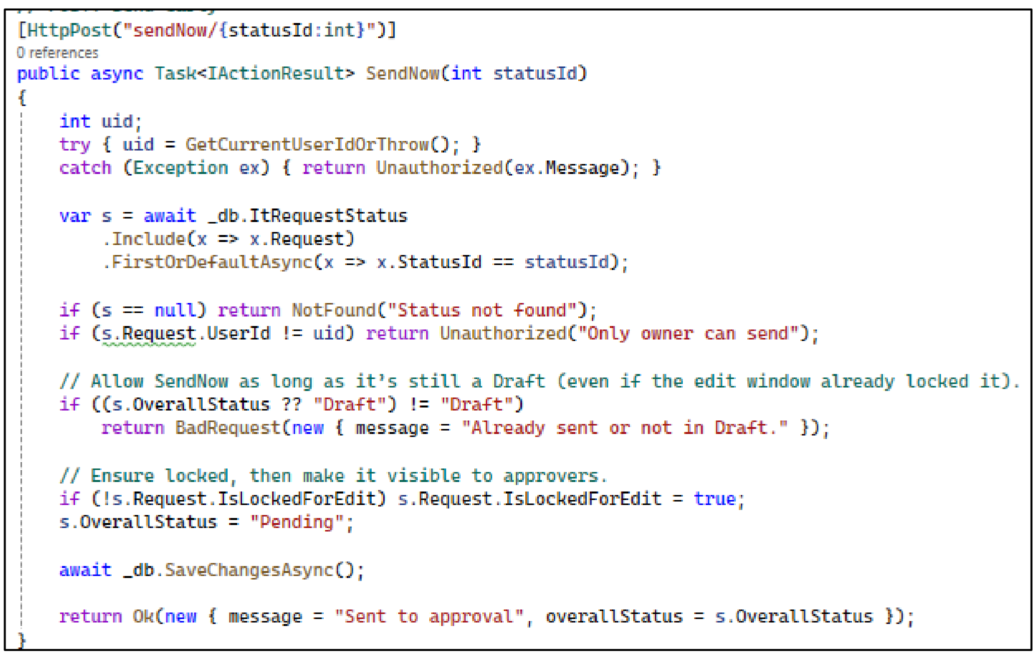
  Sends a Draft request into the approval pipeline. It locks the form and flips the overall status to Pending so approvers can see and act on it.

- **Key functionalities:**

  - Verifies the logged-in user and ensures they are the owner of the request. If not, it blocks the action.

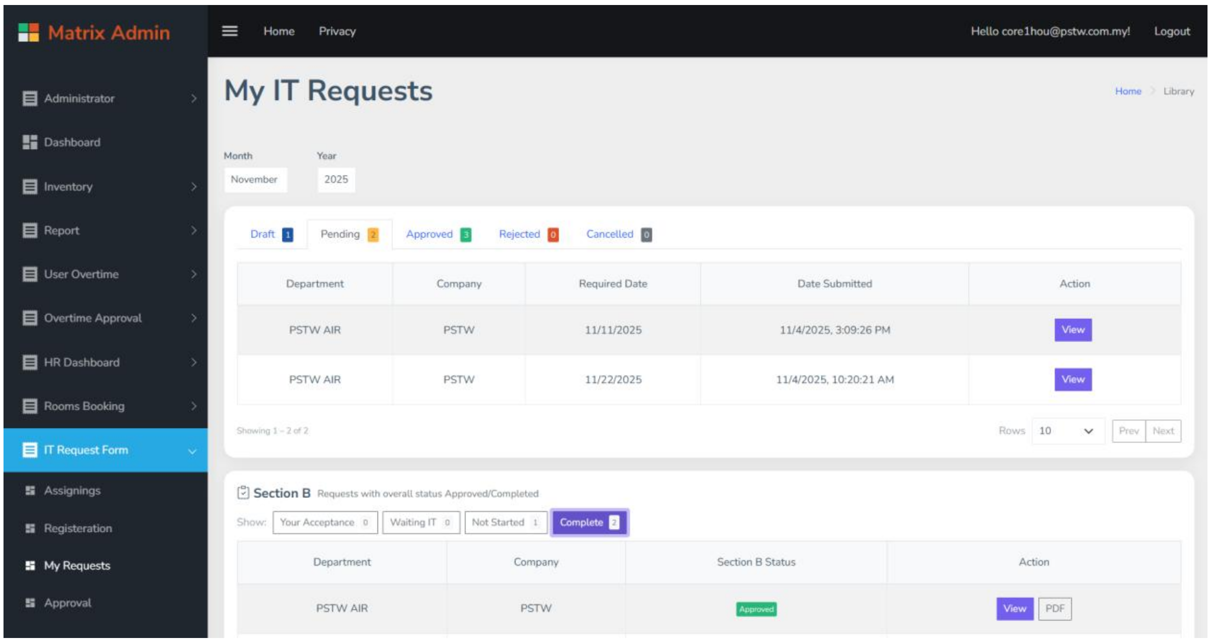
  - Loads the ItRequestStatus with its Request by statusId. Returns “Status not found” if it doesn’t exist.

- Only allows send if the request is still in Draft. If it’s already Pending, Approved, Rejected, or Cancelled, it returns a clear error message.
- Locks the request for editing (IsLockedForEdit = true) and sets OverallStatus = "Pending". This works even if the edit window has already expired, as long as the overall status is still Draft.
- Saves and returns a simple success payload so the UI can update the status immediately.



```
[HttpPost("sendNow/{statusId:int}")]
0 references
public async Task<IActionResult> SendNow(int statusId)
{
    int uid;
    try { uid = GetCurrentUserIdOrThrow(); }
    catch (Exception ex) { return Unauthorized(ex.Message); }

    var s = await _db.ItRequestStatus
        .Include(x => x.Request)
        .FirstOrDefaultAsync(x => x.StatusId == statusId);

    if (s == null) return NotFound("Status not found");
    if (s.Request.UserId != uid) return Unauthorized("Only owner can send");

    // Allow SendNow as long as it's still a Draft (even if the edit window already locked it).
    if ((s.OverallStatus ?? "Draft") != "Draft")
        return BadRequest(new { message = "Already sent or not in Draft." });

    // Ensure locked, then make it visible to approvers.
    if (!s.Request.IsLockedForEdit) s.Request.IsLockedForEdit = true;
    s.OverallStatus = "Pending";

    await _db.SaveChangesAsync();

    return Ok(new { message = "Sent to approval", overallStatus = s.OverallStatus });
}
```

- MyRequests.cshtml



My IT Requests

Month: November Year: 2025

Draft 1 | Pending 2 | Approved 3 | Rejected 0 | Cancelled 0

Department	Company	Required Date	Date Submitted	Action
PSTW AIR	PSTW	11/11/2025	11/4/2025, 3:09:26 PM	View
PSTW AIR	PSTW	11/22/2025	11/4/2025, 10:20:21 AM	View

Showing 1 – 2 of 2   Rows 10   Prev Next

Section B Requests with overall status Approved/Completed

Show: Your Acceptance 0 | Waiting IT 0 | Not Started 1 | Complete 2

Department	Company	Section B Status	Action
PSTW AIR	PSTW	Approved	View PDF

This file (MyRequests.cshtml) shows your own IT requests by month and year, split into Draft, Pending, Approved, Rejected and Cancelled. It also includes a second table dedicated to Section B so you can quickly see what needs your acceptance and what is waiting on IT.

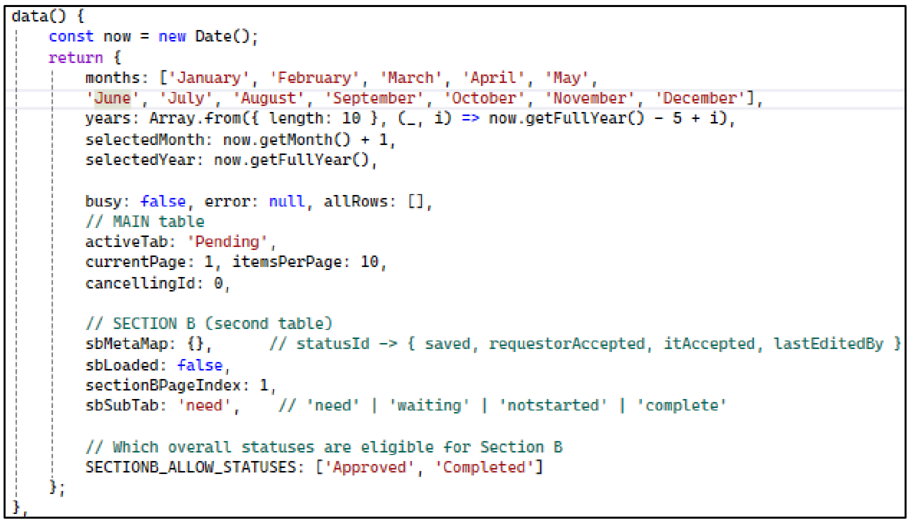
✓ **<div id="myReqApp">…</div>**
  - **Purpose:** Root container for the Vue app so the filters, tables and buttons are reactive.

✓ **const app = Vue.createApp({...})**
  - **Purpose:** Spins up the Vue instance and connects state, computed lists, user actions and lifecycle.

✓ **data()**
  - **months, years, selectedMonth, selectedYear**: These drive the date filters above the table. When the month or year changes, the page fetches that range again so your lists stay in sync with the selected period.
  - **busy, error, allRows**: busy shows loading and disables actions while a request is in flight. error surfaces API issues in a clear alert. allRows holds the raw request data for the chosen period and is the single source of truth for both tables.
  - **activeTab, currentPage, itemsPerPage, cancellingId**: These control the main table's tab state, pagination and cancel button feedback. cancellingId is

set while a cancel call is running so only that row shows a spinner and disables its button.

- **sbMetaMap, sbLoaded, sectionBPageIndex, sbSubTab, SECTIONB_ALLOW_STATUSES**: These power the Section B table. Meta is cached per statusId, sbLoaded tells the UI when the meta is ready, sectionBPageIndex handles its pagination, sbSubTab applies a sub-filter, and SECTIONB_ALLOW_STATUSES restricts Section B to Approved or Completed requests.

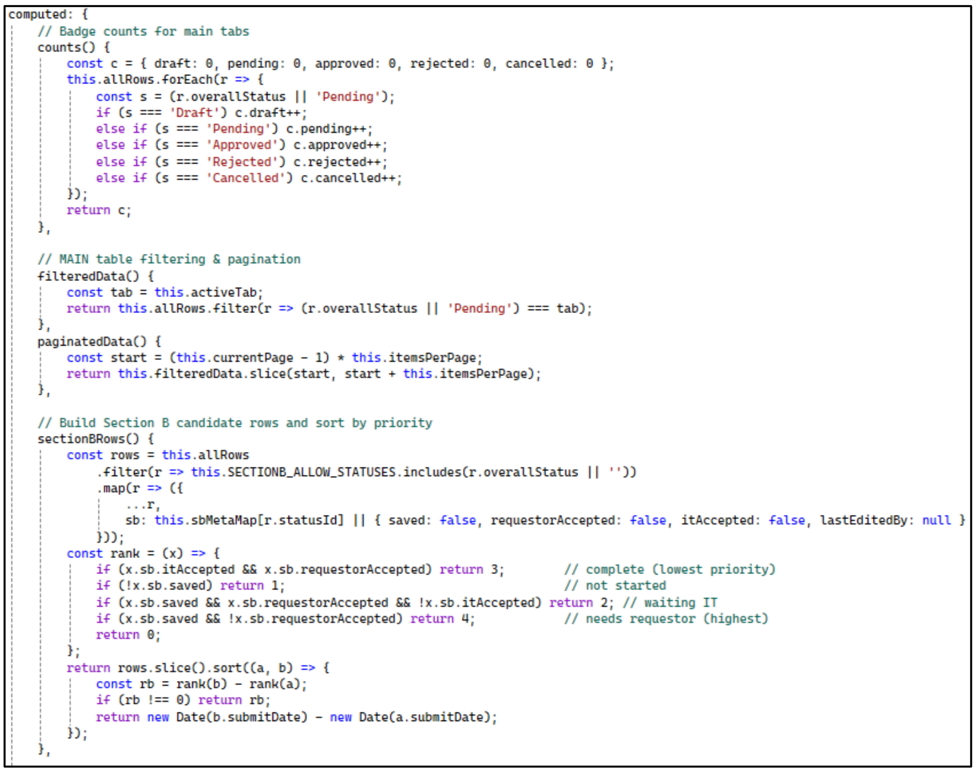


```
data() {
    const now = new Date();
    return {
        months: ['January', 'February', 'March', 'April', 'May',
        'June', 'July', 'August', 'September', 'October', 'November', 'December'],
        years: Array.from({ length: 10 }, (_, i) => now.getFullYear() - 5 + i),
        selectedMonth: now.getMonth() + 1,
        selectedYear: now.getFullYear(),

        busy: false, error: null, allRows: [],
        // MAIN table
        activeTab: 'Pending',
        currentPage: 1, itemsPerPage: 10,
        cancellingId: 0,

        // SECTION B (second table)
        sbMetaMap: {},      // statusId -> { saved, requestorAccepted, itAccepted, lastEditedBy }
        sbLoaded: false,
        sectionBPageIndex: 1,
        sbSubTab: 'need',    // 'need' | 'waiting' | 'notstarted' | 'complete'

        // Which overall statuses are eligible for Section B
        SECTIONB_ALLOW_STATUSES: ['Approved', 'Completed']
    };
},
```

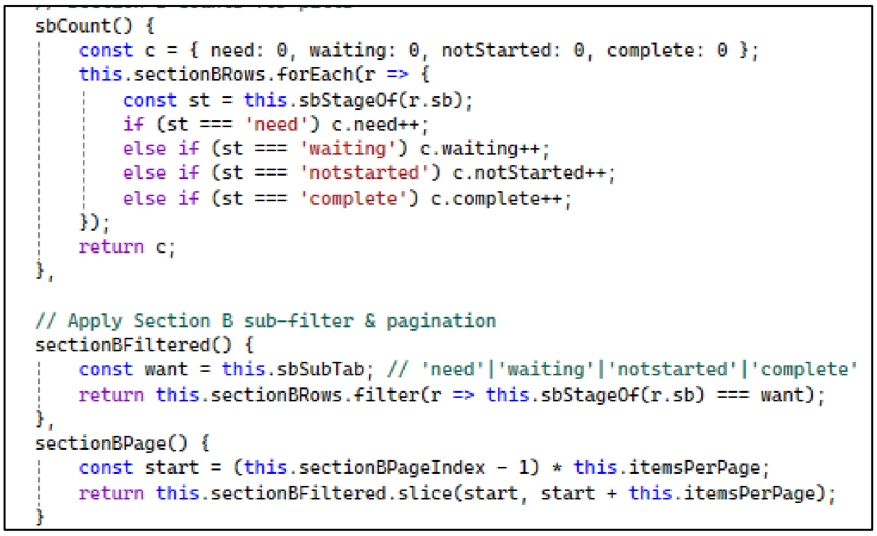
✓ **computed**

- **Counts**: Builds the badge totals for each main tab by walking allRows once. This keeps the headings accurate without extra API calls.
- **filteredData, paginatedData**: filteredData returns the slice of allRows that matches the current main tab. paginatedData then applies page and size so the table shows only what is needed, with a small footer that tells you how many you are viewing.
- **sectionBRows**: Constructs candidate rows for Section B by taking only Approved and Completed items and attaching their meta from sbMetaMap. It sorts them by priority so items that need your action float to the top, then by newest submission.

- **sbCount**: Tallies Section B by stage into four buckets: need, waiting, not started, and complete. These numbers feed the pill buttons and the callout that nudges you when your acceptance is required.
- **sectionBFiltered, sectionBPage**: Applies the current Section B sub-filter and paginates it for the second table. This mirrors the main table pattern so navigation is consistent.



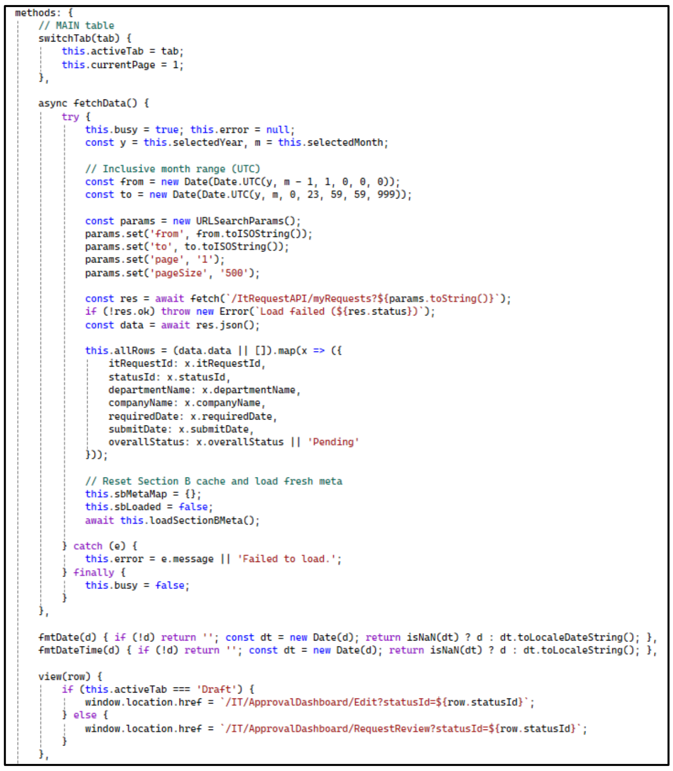
```
computed: {
    // Badge counts for main tabs
    counts() {
        const c = { draft: 0, pending: 0, approved: 0, rejected: 0, cancelled: 0 };
        this.allRows.forEach(r => {
            const s = (r.overallStatus || 'Pending');
            if (s === 'Draft') c.draft++;
            else if (s === 'Pending') c.pending++;
            else if (s === 'Approved') c.approved++;
            else if (s === 'Rejected') c.rejected++;
            else if (s === 'Cancelled') c.cancelled++;
        });
        return c;
    },

    // MAIN table filtering & pagination
    filteredData() {
        const tab = this.activeTab;
        return this.allRows.filter(r => (r.overallStatus || 'Pending') === tab);
    },
    paginatedData() {
        const start = (this.currentPage - 1) * this.itemsPerPage;
        return this.filteredData.slice(start, start + this.itemsPerPage);
    },

    // Build Section B candidate rows and sort by priority
    sectionBRows() {
        const rows = this.allRows
            .filter(r => this.SECTIONB_ALLOW_STATUSES.includes(r.overallStatus || ''))
            .map(r => ({
                ...r,
                sb: this.sbMetaMap[r.statusId] || { saved: false, requestorAccepted: false, itAccepted: false, lastEditedBy: null }
            }));
        const rank = (x) => {
            if (x.sb.itAccepted && x.sb.requestorAccepted) return 3;        // complete (lowest priority)
            if (!x.sb.saved) return 1;                                      // not started
            if (x.sb.saved && x.sb.requestorAccepted && !x.sb.itAccepted) return 2; // waiting IT
            if (x.sb.saved && !x.sb.requestorAccepted) return 4;            // needs requestor (highest)
            return 0;
        };
        return rows.slice().sort((a, b) => {
            const rb = rank(b) - rank(a);
            if (rb !== 0) return rb;
            return new Date(b.submitDate) - new Date(a.submitDate);
        });
    },
```



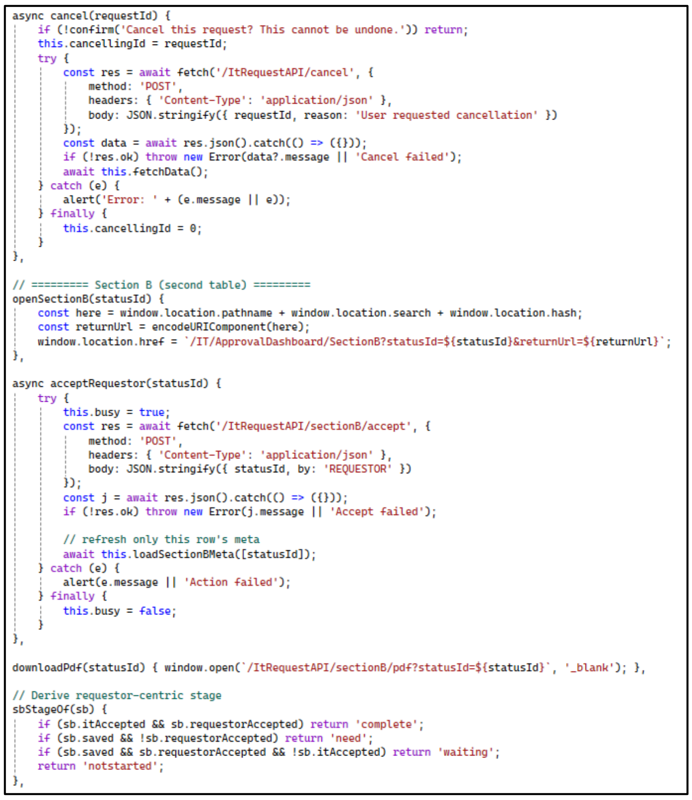
```
    sbCount() {
        const c = { need: 0, waiting: 0, notStarted: 0, complete: 0 };
        this.sectionBRows.forEach(r => {
            const st = this.sbStageOf(r.sb);
            if (st === 'need') c.need++;
            else if (st === 'waiting') c.waiting++;
            else if (st === 'notstarted') c.notStarted++;
            else if (st === 'complete') c.complete++;
        });
        return c;
    },

    // Apply Section B sub-filter & pagination
    sectionBFiltered() {
        const want = this.sbSubTab; // 'need'|'waiting'|'notstarted'|'complete'
        return this.sectionBRows.filter(r => this.sbStageOf(r.sb) === want);
    },
    sectionBPage() {
        const start = (this.sectionBPageIndex - 1) * this.itemsPerPage;
        return this.sectionBFiltered.slice(start, start + this.itemsPerPage);
    }
```

✓ **methods**

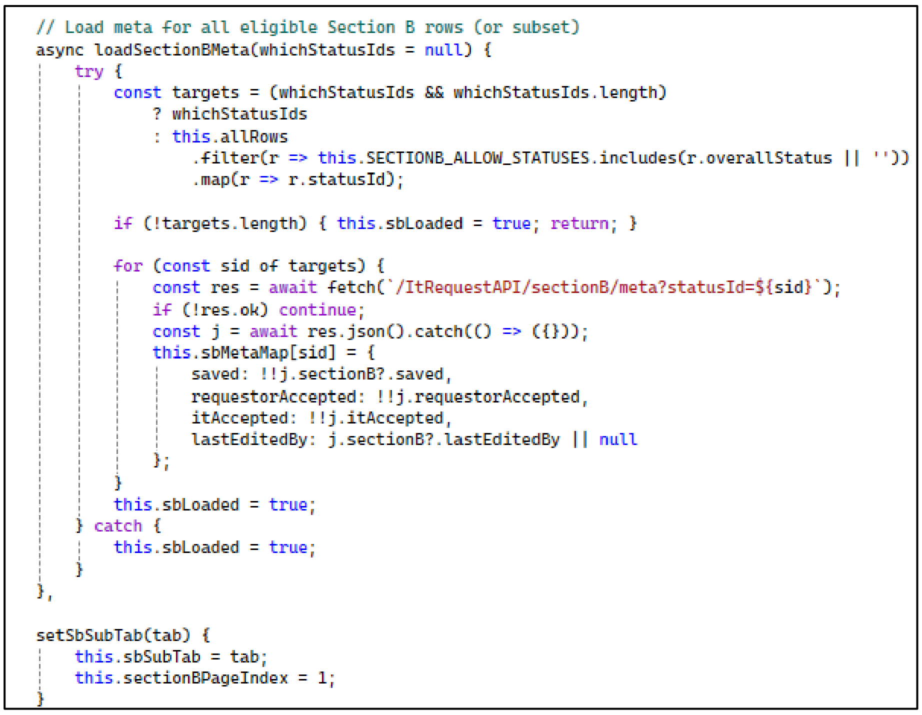
- **switchTab(tab)**: Switches the main table view and resets pagination to the first page. This avoids confusing empty pages when moving between tabs with different totals.
- **fetchData()**: Loads your requests for the selected month range using UTC boundaries and a generous page size. It normalizes the results into allRows, clears and reloads Section B meta, and sets busy and error so the UI reflects what is happening.
- **fmtDate, fmtDateTime**: Small display helpers for date only and date time cells. They fall back safely if the backend returns unexpected formats.
- **view(row)**: Navigates to Edit when the row is a Draft and to RequestReview for every other status. This matches business rules where only drafts are editable by the requester.
- **cancel(requestId)**: Confirms with the user, posts to /ItRequestAPI/cancel, and then refreshes the list. cancellingId ensures only the clicked row shows a spinner so the rest of the table remains usable.
- **openSectionB(statusId)**: Sends you to the Section B page for that request and includes a returnUrl back to this page. When you are done, you land back on the same filters and state you left.
- **acceptRequestor(statusId)**: Calls /ItRequestAPI/sectionB/accept with by: 'REQUESTOR' for a single item. On success it reloads meta only for that statusId so the page updates instantly without a full refresh.
- **downloadPdf(statusId)**: Opens the Section B PDF in a new tab for completed items. This keeps your current list intact while you view or download the document.
- **sbStageOf(sb)**: Converts raw meta flags into a simple stage string: complete, need, waiting or not started. The rest of the UI uses this single source for pills, badges and filters.
- **loadSectionBMeta(whichStatusIds = null)**: Fetches Section B meta for either all eligible rows or a provided subset. Results are stored in sbMetaMap and sbLoaded flips when done so the table knows when to render real rows.
- **setSbSubTab(tab)**: Changes the Section B sub-filter and resets its pagination to page one. This avoids landing on an empty page after switching pills.



```
methods: {
    // MAIN table
    switchTab(tab) {
        this.activeTab = tab;
        this.currentPage = 1;
    },

    async fetchData() {
        try {
            this.busy = true; this.error = null;
            const y = this.selectedYear, m = this.selectedMonth;

            // Inclusive month range (UTC)
            const from = new Date(Date.UTC(y, m - 1, 1, 0, 0, 0));
            const to = new Date(Date.UTC(y, m, 0, 23, 59, 59, 999));

            const params = new URLSearchParams();
            params.set('from', from.toISOString());
            params.set('to', to.toISOString());
            params.set('page', '1');
            params.set('pageSize', '500');

            const res = await fetch(`/ItRequestAPI/myRequests?${params.toString()}`);
            if (!res.ok) throw new Error(`Load failed (${res.status})`);
            const data = await res.json();

            this.allRows = (data.data || []).map(x => ({
                itRequestId: x.itRequestId,
                statusId: x.statusId,
                departmentName: x.departmentName,
                companyName: x.companyName,
                requiredDate: x.requiredDate,
                submitDate: x.submitDate,
                overallStatus: x.overallStatus || 'Pending'
            }));

            // Reset Section B cache and load fresh meta
            this.sbMetaMap = {};
            this.sbLoaded = false;
            await this.loadSectionBMeta();

        } catch (e) {
            this.error = e.message || 'Failed to load.';
        } finally {
            this.busy = false;
        }
    },

    fmtDate(d) { if (!d) return ''; const dt = new Date(d); return isNaN(dt) ? d : dt.toLocaleDateString(); },
    fmtDateTime(d) { if (!d) return ''; const dt = new Date(d); return isNaN(dt) ? d : dt.toLocaleString(); },

    view(row) {
        if (this.activeTab === 'Draft') {
            window.location.href = `/IT/ApprovalDashboard/Edit?statusId=${row.statusId}`;
        } else {
            window.location.href = `/IT/ApprovalDashboard/RequestReview?statusId=${row.statusId}`;
        }
    },
```

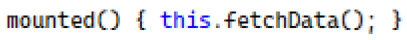


```
async cancel(requestId) {
    if (!confirm('Cancel this request? This cannot be undone.')) return;
    this.cancellingId = requestId;
    try {
        const res = await fetch('/ItRequestAPI/cancel', {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({ requestId, reason: 'User requested cancellation' })
        });
        const data = await res.json().catch(() => ({}));
        if (!res.ok) throw new Error(data?.message || 'Cancel failed');
        await this.fetchData();
    } catch (e) {
        alert('Error: ' + (e.message || e));
    } finally {
        this.cancellingId = 0;
    }
},

// ========= Section B (second table) =========
openSectionB(statusId) {
    const here = window.location.pathname + window.location.search + window.location.hash;
    const returnUrl = encodeURIComponent(here);
    window.location.href = `/IT/ApprovalDashboard/SectionB?statusId=${statusId}&returnUrl=${returnUrl}`;
},

async acceptRequestor(statusId) {
    try {
        this.busy = true;
        const res = await fetch('/ItRequestAPI/sectionB/accept', {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({ statusId, by: 'REQUESTOR' })
        });
        const j = await res.json().catch(() => ({}));
        if (!res.ok) throw new Error(j.message || 'Accept failed');

        // refresh only this row's meta
        await this.loadSectionBMeta([statusId]);
    } catch (e) {
        alert(e.message || 'Action failed');
    } finally {
        this.busy = false;
    }
},

downloadPdf(statusId) { window.open(`/ItRequestAPI/sectionB/pdf?statusId=${statusId}`, '_blank'); },

// Derive requestor-centric stage
sbStageOf(sb) {
    if (sb.itAccepted && sb.requestorAccepted) return 'complete';
    if (sb.saved && !sb.requestorAccepted) return 'need';
    if (sb.saved && sb.requestorAccepted && !sb.itAccepted) return 'waiting';
    return 'notstarted';
},
```



```
// Load meta for all eligible Section B rows (or subset)
async loadSectionBMeta(whichStatusIds = null) {
    try {
        const targets = (whichStatusIds && whichStatusIds.length)
            ? whichStatusIds
            : this.allRows
                .filter(r => this.SECTIONB_ALLOW_STATUSES.includes(r.overallStatus || ''))
                .map(r => r.statusId);

        if (!targets.length) { this.sbLoaded = true; return; }

        for (const sid of targets) {
            const res = await fetch(`/ItRequestAPI/sectionB/meta?statusId=${sid}`);
            if (!res.ok) continue;
            const j = await res.json().catch(() => ({}));
            this.sbMetaMap[sid] = {
                saved: !!j.sectionB?.saved,
                requestorAccepted: !!j.requestorAccepted,
                itAccepted: !!j.itAccepted,
                lastEditedBy: j.sectionB?.lastEditedBy || null
            };
        }
        this.sbLoaded = true;
    } catch {
        this.sbLoaded = true;
    }
},

setSbSubTab(tab) {
    this.sbSubTab = tab;
    this.sectionBPageIndex = 1;
}
```

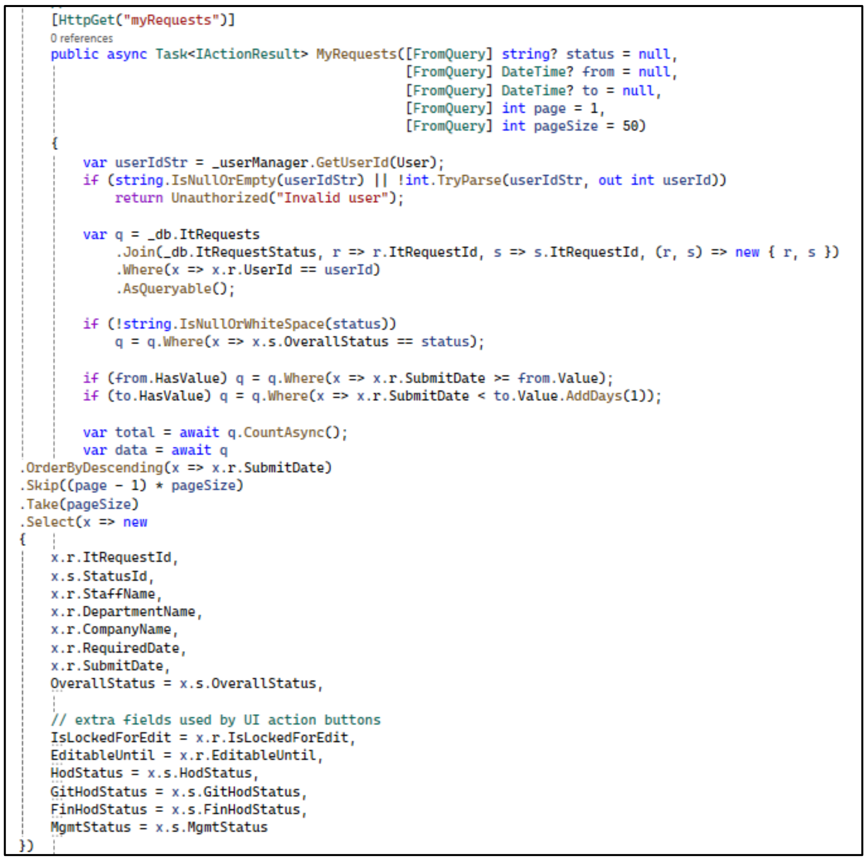
- ✓ mounted
  - **mounted()**: Immediately calls fetchData() for the default month and year. The page shows skeleton rows while loading and then hydrates both tables and the Section B meta without extra clicks.



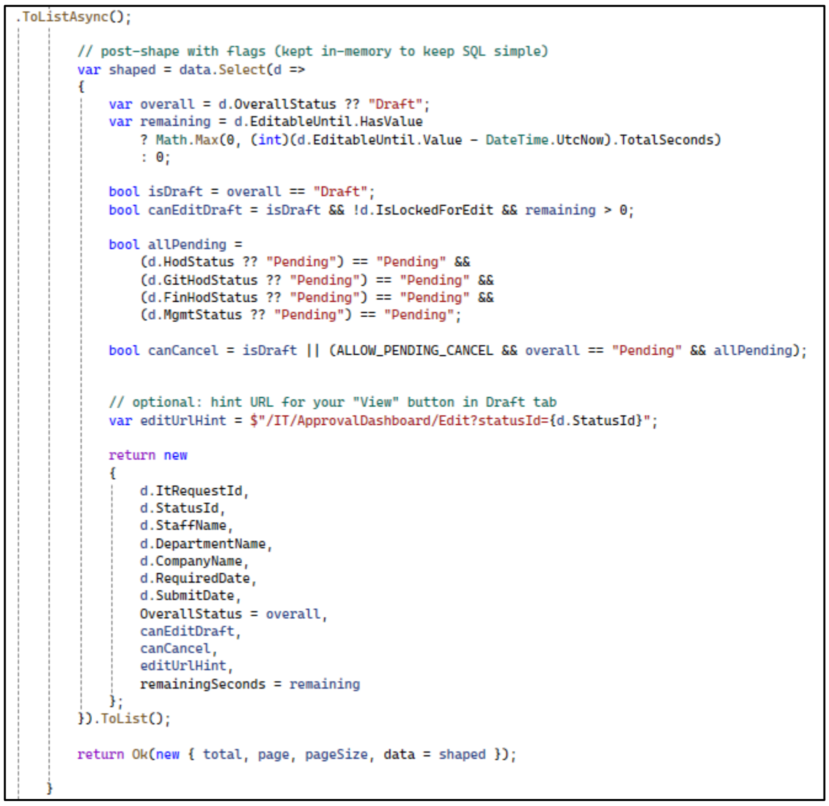
```
mounted() { this.fetchData(); }
```

- ITRequestAPI (MyRequests.cshtml)
  - ▪ **[HttpGet("myRequests")]** (Used by: MyRequests.cshtml)
  - **Purpose:**
    Returns the current user's own IT requests with simple filters and paging, plus handy flags the page can use for Edit, Cancel, and countdown UI.
  - **Key functionalities:**
    - Authenticates the caller and gets userId. If missing or invalid, it returns Unauthorized.
    - Builds a base query that joins ItRequests to ItRequestStatus, limited to r.UserId == userId.
    - Optional filters:

    - status filters by OverallStatus exactly.
    - from and to filter by SubmitDate. to is treated as end of day using to.Value.AddDays(1) on the upper bound.
- Pagination: counts total, then orders by SubmitDate descending, applies Skip and Take.
- Shapes each row with the main fields and extra approval fields that the UI needs.
- Post-processing in memory to compute UI flags:
    - remainingSeconds based on EditableUntil - UtcNow.
    - canEditDraft only if OverallStatus == "Draft", not locked, and time remains.
    - allPending means every approver is still "Pending".
    - canCancel is true if it is a Draft, or if ALLOW_PENDING_CANCEL is enabled and the request is "Pending" and allPending is true.
    - editUrlHint is prebuilt so the UI can navigate to the Edit page quickly.
- Returns a paged payload: { total, page, pageSize, data }, where each item already includes OverallStatus, canEditDraft, canCancel, editUrlHint, and remainingSeconds.



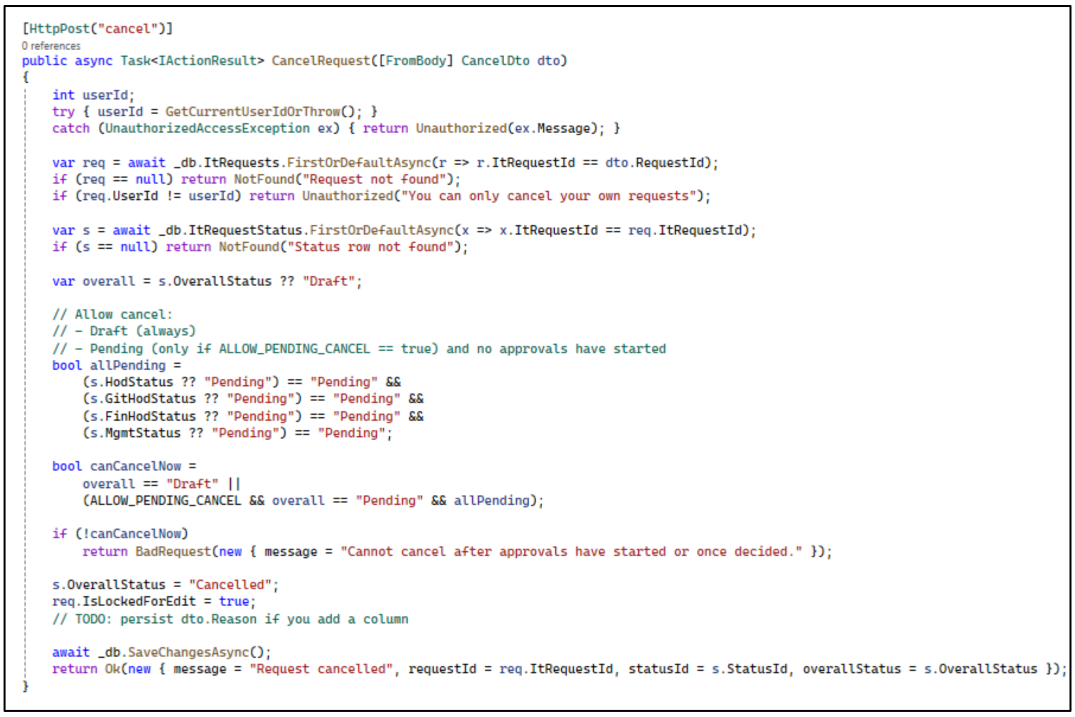
```
    [HttpGet("myRequests")]
    0 references
    public async Task<IActionResult> MyRequests([FromQuery] string? status = null,
                                                [FromQuery] DateTime? from = null,
                                                [FromQuery] DateTime? to = null,
                                                [FromQuery] int page = 1,
                                                [FromQuery] int pageSize = 50)
    {
        var userIdStr = _userManager.GetUserId(User);
        if (string.IsNullOrEmpty(userIdStr) || !int.TryParse(userIdStr, out int userId))
            return Unauthorized("Invalid user");

        var q = _db.ItRequests
            .Join(_db.ItRequestStatus, r => r.ItRequestId, s => s.ItRequestId, (r, s) => new { r, s })
            .Where(x => x.r.UserId == userId)
            .AsQueryable();

        if (!string.IsNullOrWhiteSpace(status))
            q = q.Where(x => x.s.OverallStatus == status);

        if (from.HasValue) q = q.Where(x => x.r.SubmitDate >= from.Value);
        if (to.HasValue) q = q.Where(x => x.r.SubmitDate < to.Value.AddDays(1));

        var total = await q.CountAsync();
        var data = await q
.OrderByDescending(x => x.r.SubmitDate)
.Skip((page - 1) * pageSize)
.Take(pageSize)
.Select(x => new
{
    x.r.ItRequestId,
    x.s.StatusId,
    x.r.StaffName,
    x.r.DepartmentName,
    x.r.CompanyName,
    x.r.RequiredDate,
    x.r.SubmitDate,
    OverallStatus = x.s.OverallStatus,

    // extra fields used by UI action buttons
    IsLockedForEdit = x.r.IsLockedForEdit,
    EditableUntil = x.r.EditableUntil,
    HodStatus = x.s.HodStatus,
    GitHodStatus = x.s.GitHodStatus,
    FinHodStatus = x.s.FinHodStatus,
    MgmtStatus = x.s.MgmtStatus
})
```



```
.ToListAsync();

        // post-shape with flags (kept in-memory to keep SQL simple)
        var shaped = data.Select(d =>
        {
            var overall = d.OverallStatus ?? "Draft";
            var remaining = d.EditableUntil.HasValue
                ? Math.Max(0, (int)(d.EditableUntil.Value - DateTime.UtcNow).TotalSeconds)
                : 0;

            bool isDraft = overall == "Draft";
            bool canEditDraft = isDraft && !d.IsLockedForEdit && remaining > 0;

            bool allPending =
                (d.HodStatus ?? "Pending") == "Pending" &&
                (d.GitHodStatus ?? "Pending") == "Pending" &&
                (d.FinHodStatus ?? "Pending") == "Pending" &&
                (d.MgmtStatus ?? "Pending") == "Pending";

            bool canCancel = isDraft || (ALLOW_PENDING_CANCEL && overall == "Pending" && allPending);


            // optional: hint URL for your "View" button in Draft tab
            var editUrlHint = $"/IT/ApprovalDashboard/Edit?statusId={d.StatusId}";

            return new
            {
                d.ItRequestId,
                d.StatusId,
                d.StaffName,
                d.DepartmentName,
                d.CompanyName,
                d.RequiredDate,
                d.SubmitDate,
                OverallStatus = overall,
                canEditDraft,
                canCancel,
                editUrlHint,
                remainingSeconds = remaining
            };
        }).ToList();

        return Ok(new { total, page, pageSize, data = shaped });
    }
```

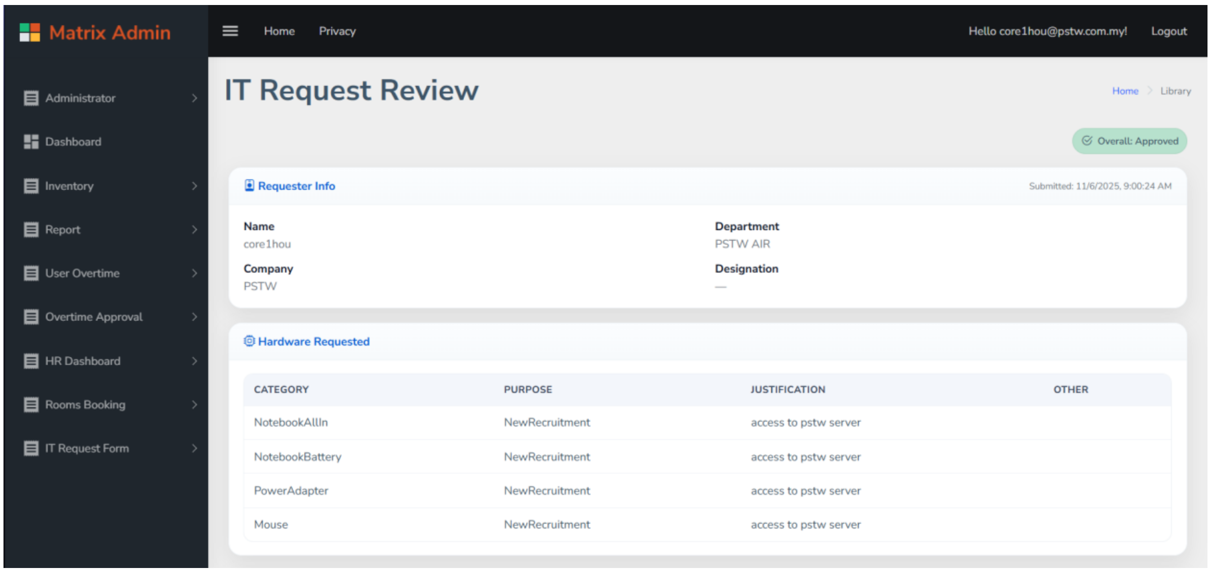
- **[HttpPost("cancel")] — Used by:** MyRequests.cshtml (Cancel button)

- **Purpose:**
  This endpoint lets a requester cancel their own IT request, either while it’s still in **Draft** or when it’s **Pending** and no one in the approval chain has acted yet.
- **Key functionalities:**
  - It checks the logged-in user and makes sure they’re the actual owner of the request. If someone else tries to cancel, the system blocks it immediately.
  - It finds the main request (ItRequests) and its matching status row (ItRequestStatus). If either can’t be found, it returns a clear “not found” message.
  - The logic then looks at the overall status:
    - **Draft:** Always allowed to cancel.
    - **Pending:** Allowed only if the system’s ALLOW_PENDING_CANCEL flag is true **and** every approver (HOD, Group IT, Finance, Management) is still “Pending.”
  - If any approval has already been made (Approved or Rejected), the cancellation is blocked with a friendly error message so the user knows why.
  - When a valid cancellation happens, the system updates the status to **Cancelled**, locks the request so it can’t be edited anymore, and saves the changes.
  - It then sends back a short success response so the page can refresh or update the table immediately.



```
[HttpPost("cancel")]
0 references
public async Task<IActionResult> CancelRequest([FromBody] CancelDto dto)
{
    int userId;
    try { userId = GetCurrentUserIdOrThrow(); }
    catch (UnauthorizedAccessException ex) { return Unauthorized(ex.Message); }

    var req = await _db.ItRequests.FirstOrDefaultAsync(r => r.ItRequestId == dto.RequestId);
    if (req == null) return NotFound("Request not found");
    if (req.UserId != userId) return Unauthorized("You can only cancel your own requests");

    var s = await _db.ItRequestStatus.FirstOrDefaultAsync(x => x.ItRequestId == req.ItRequestId);
    if (s == null) return NotFound("Status row not found");

    var overall = s.OverallStatus ?? "Draft";

    // Allow cancel:
    // - Draft (always)
    // - Pending (only if ALLOW_PENDING_CANCEL == true) and no approvals have started
    bool allPending =
        (s.HodStatus ?? "Pending") == "Pending" &&
        (s.GitHodStatus ?? "Pending") == "Pending" &&
        (s.FinHodStatus ?? "Pending") == "Pending" &&
        (s.MgmtStatus ?? "Pending") == "Pending";

    bool canCancelNow =
        overall == "Draft" ||
        (ALLOW_PENDING_CANCEL && overall == "Pending" && allPending);

    if (!canCancelNow)
        return BadRequest(new { message = "Cannot cancel after approvals have started or once decided." });

    s.OverallStatus = "Cancelled";
    req.IsLockedForEdit = true;
    // TODO: persist dto.Reason if you add a column

    await _db.SaveChangesAsync();
    return Ok(new { message = "Request cancelled", requestId = req.ItRequestId, statusId = s.StatusId, overallStatus = s.OverallStatus });
}
```

- RequestReview.cshtml



IT Request Review

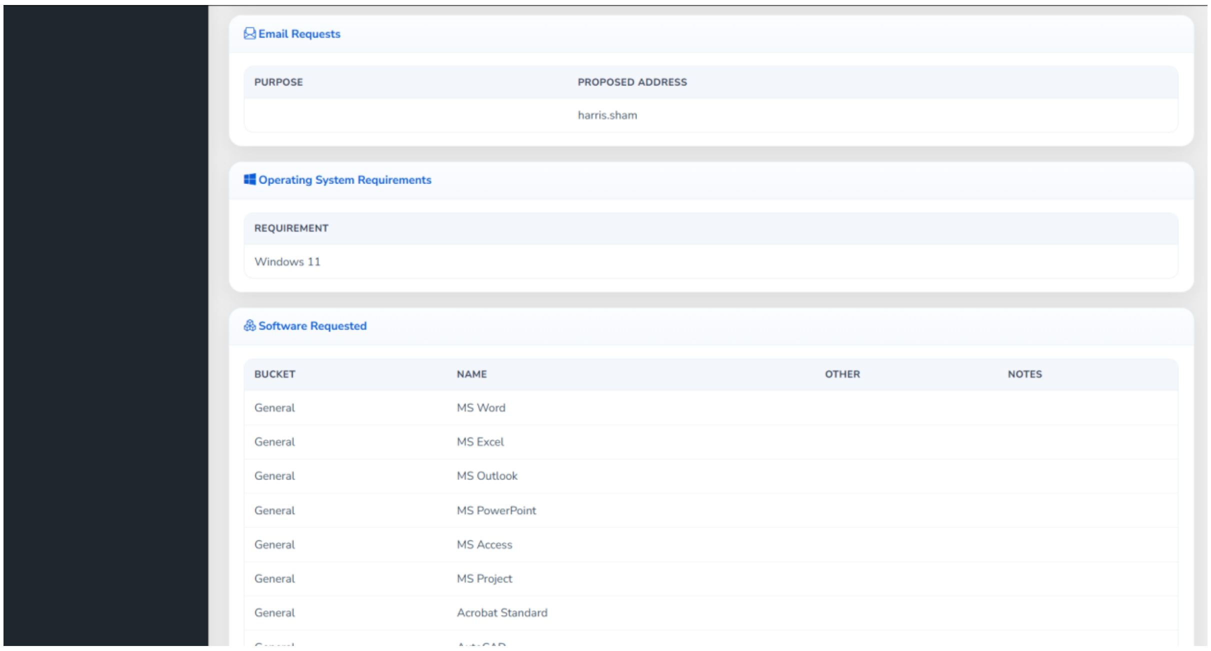
Overall: Approved

Requester Info — Submitted: 11/6/2025, 9:00:24 AM

Name core1hou	**Department** PSTW AIR
Company PSTW	**Designation** —

Hardware Requested

CATEGORY	PURPOSE	JUSTIFICATION	OTHER
NotebookAllIn	NewRecruitment	access to pstw server	
NotebookBattery	NewRecruitment	access to pstw server	
PowerAdapter	NewRecruitment	access to pstw server	
Mouse	NewRecruitment	access to pstw server	



Email Requests

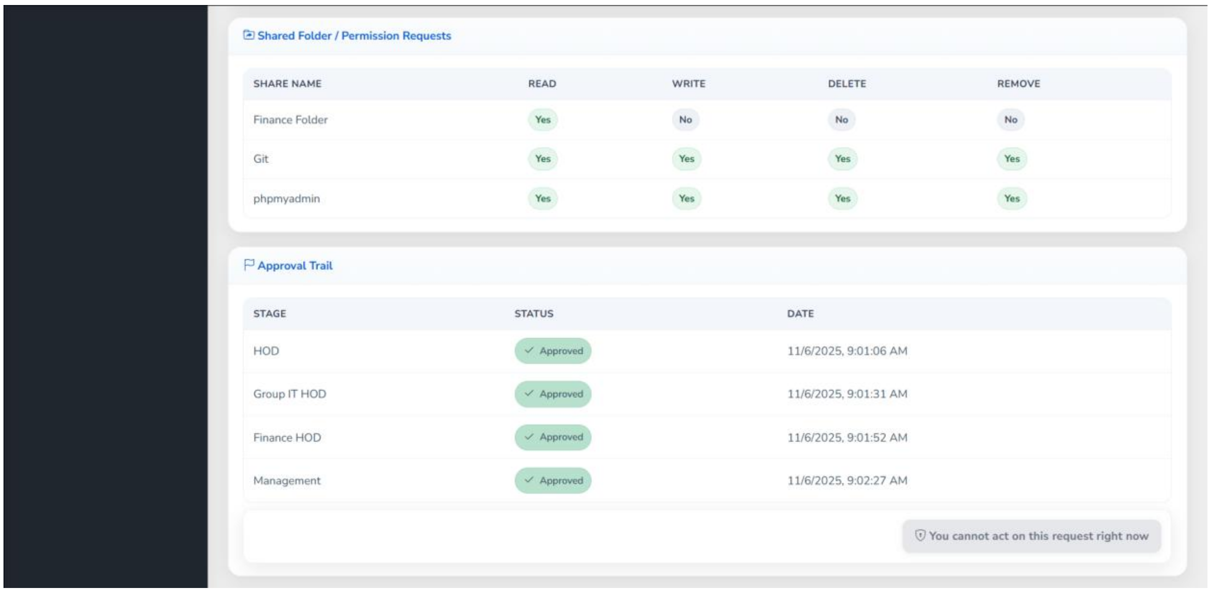
PURPOSE	PROPOSED ADDRESS
	harris.sham

Operating System Requirements

REQUIREMENT
Windows 11

Software Requested

BUCKET	NAME	OTHER	NOTES
General	MS Word		
General	MS Excel		
General	MS Outlook		
General	MS PowerPoint		
General	MS Access		
General	MS Project		
General	Acrobat Standard		



Shared Folder / Permission Requests

SHARE NAME	READ	WRITE	DELETE	REMOVE
Finance Folder	Yes	No	No	No
Git	Yes	Yes	Yes	Yes
phpmyadmin	Yes	Yes	Yes	Yes

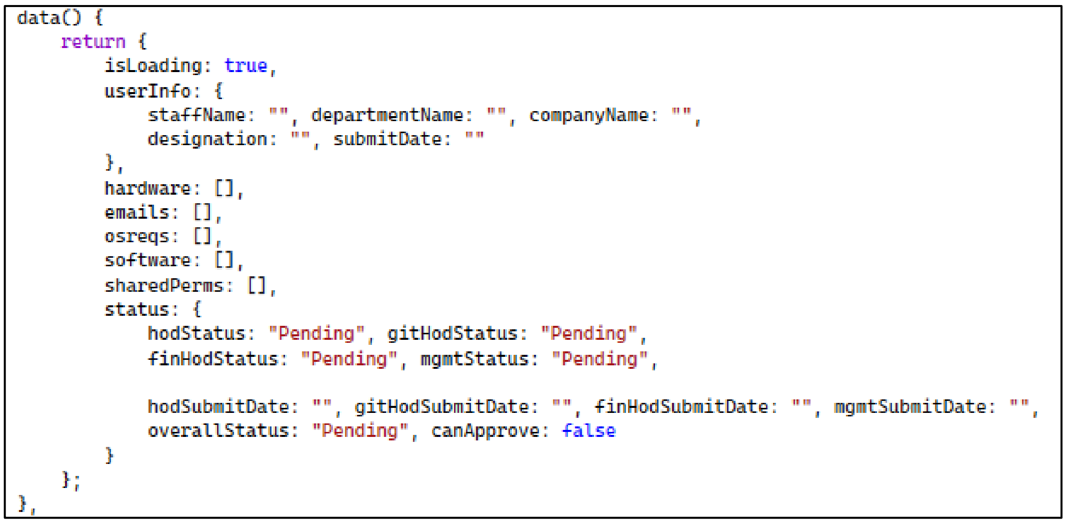
Approval Trail

STAGE	STATUS	DATE
HOD	✓ Approved	11/6/2025, 9:01:06 AM
Group IT HOD	✓ Approved	11/6/2025, 9:01:31 AM
Finance HOD	✓ Approved	11/6/2025, 9:01:52 AM
Management	✓ Approved	11/6/2025, 9:02:27 AM

This file (RequestReview.cshtml) is the read-only review and approval screen. It pulls a full snapshot of the request and shows each section clearly, along with stage chips and a sticky action bar for approvers.

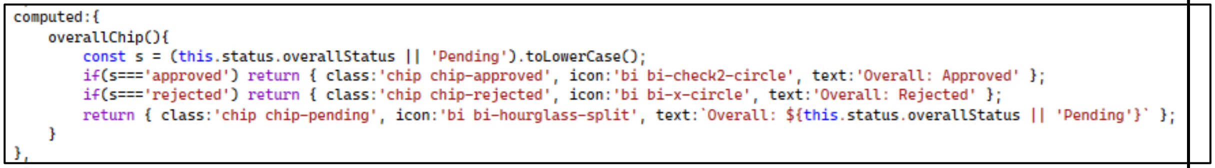
- ✓ **<div id="reviewApp">…</div>**
  - **Purpose:** Root container for the Vue app so the cards, tables, and action buttons are reactive.
- ✓ **const reviewApp = Vue.createApp({...})**
  - **Purpose:** Starts the Vue instance and wires up loading state, derived chips, API calls, and lifecycle.

- ✓ **data()**
  - **isLoading**: Controls skeletons and disables the action bar until the request is fully loaded. The page feels responsive even when the payload is large.
  - **userInfo { staffName, departmentName, companyName, designation, submitDate }**: Snapshot of who submitted and when. Values are normalized from the API so the view does not care about PascalCase vs camelCase.
  - **hardware, emails, osreqs, software, sharedPerms**: Each array feeds one card. They are shaped to match table columns, with safe defaults for missing fields so empty states render cleanly.
  - **status { per-stage statuses, per-stage dates, overallStatus, canApprove }**: Central state for the approval trail and the sticky action bar. canApprove is a server truth flag and can be overridden locally when the overall status makes actions irrelevant.



```
data() {
    return {
        isLoading: true,
        userInfo: {
            staffName: "", departmentName: "", companyName: "",
            designation: "", submitDate: ""
        },
        hardware: [],
        emails: [],
        osreqs: [],
        software: [],
        sharedPerms: [],
        status: {
            hodStatus: "Pending", gitHodStatus: "Pending",
            finHodStatus: "Pending", mgmtStatus: "Pending",

            hodSubmitDate: "", gitHodSubmitDate: "", finHodSubmitDate: "", mgmtSubmitDate: "",
            overallStatus: "Pending", canApprove: false
        }
    };
},
```

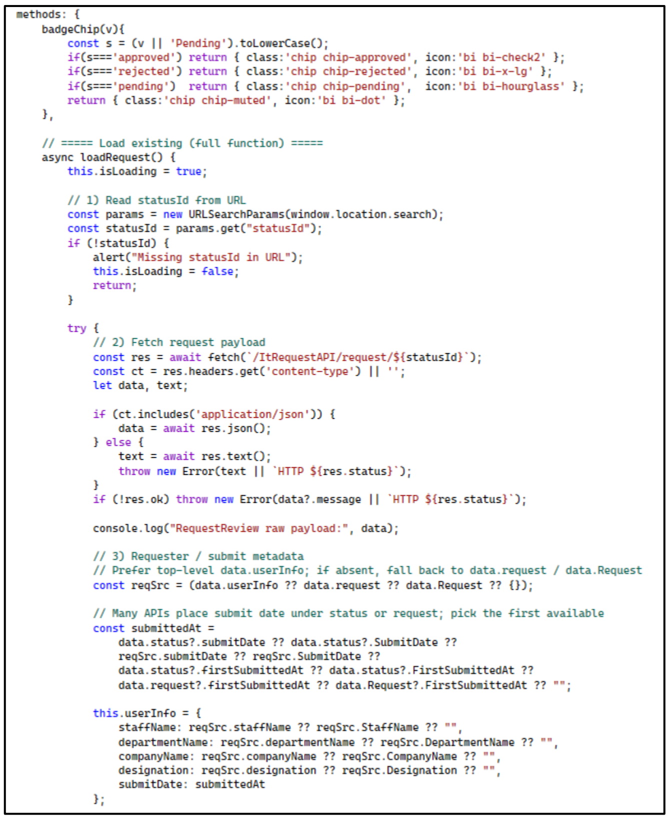
- ✓ **computed**
  - **overallChip**: Produces a small object with class, icon, and text for the big chip in the header. It reacts to status.overallStatus and returns Approved, Rejected, or a friendly Pending variant.



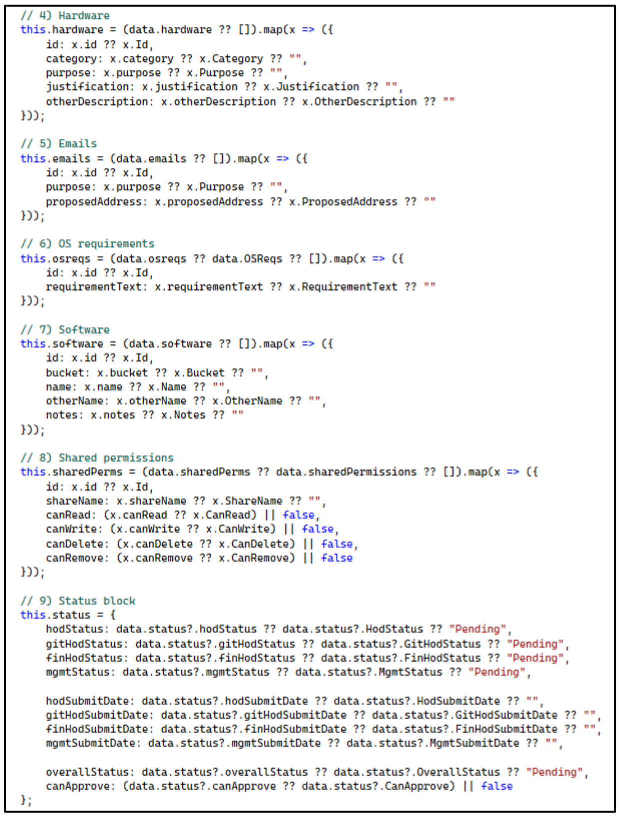
```
computed:{
    overallChip(){
        const s = (this.status.overallStatus || 'Pending').toLowerCase();
        if(s==='approved') return { class:'chip chip-approved', icon:'bi bi-check2-circle', text:'Overall: Approved' };
        if(s==='rejected') return { class:'chip chip-rejected', icon:'bi bi-x-circle', text:'Overall: Rejected' };
        return { class:'chip chip-pending', icon:'bi bi-hourglass-split', text:`Overall: ${this.status.overallStatus || 'Pending'}` };
    }
},
```

- ✓ **methods**

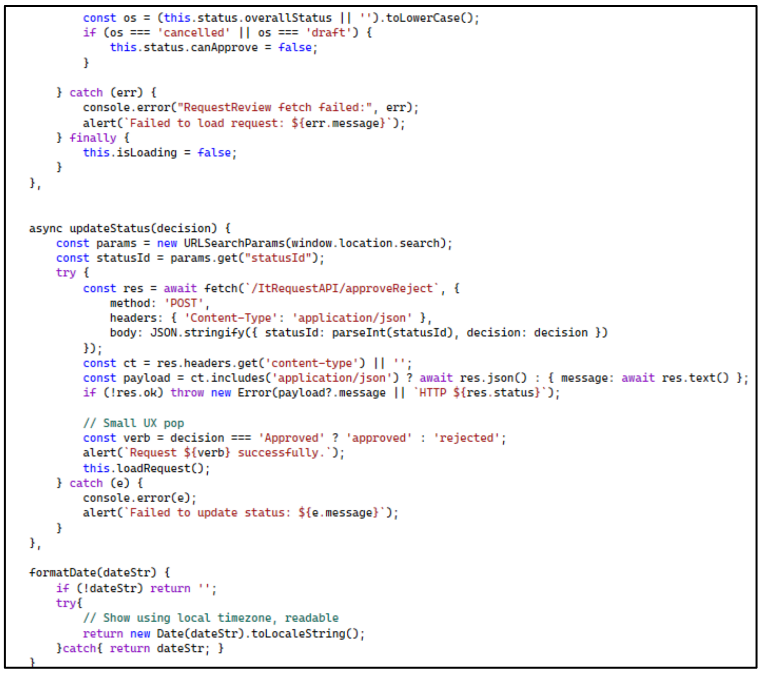
- **badgeChip(v)**: Maps any per-stage status into a tiny chip model for the trail table. Approved shows a green check, Rejected shows a red cross, Pending shows an amber hourglass, and everything else falls back to a muted dot.
- **loadRequest()**: End-to-end loader. It reads statusId from the query string, fetches /ItRequestAPI/request/{statusId}, and hydrates all sections.
  - Requester block: prefers a userInfo payload, otherwise falls back to request fields and picks the best available submit timestamp.
  - Each section: maps incoming arrays to a stable shape and fills gaps so tables never break on missing fields.
  - Status block: normalizes stage statuses and dates, sets an overallStatus, and respects canApprove from the server. If the overall is Cancelled or Draft, it switches off actions locally so the buttons cannot be used by mistake. A single try/finally flips isLoading, which drives skeletons and chip rendering.
- **updateStatus(decision)**: Sends the approver's choice to /ItRequestAPI/approveReject with the current statusId. It shows a small success alert, then reloads the page state by calling loadRequest() so badges and chips reflect the new decision immediately.
- **formatDate(dateStr)**: Best effort formatter that uses the browser locale. If the backend sends a non-date string, it returns the raw content so the UI does not show "Invalid Date."



```
methods: {
    badgeChip(v){
        const s = (v || 'Pending').toLowerCase();
        if(s==='approved') return { class:'chip chip-approved', icon:'bi bi-check2' };
        if(s==='rejected') return { class:'chip chip-rejected', icon:'bi bi-x-lg' };
        if(s==='pending')  return { class:'chip chip-pending',  icon:'bi bi-hourglass' };
        return { class:'chip chip-muted', icon:'bi bi-dot' };
    },

    // ===== Load existing (full function) =====
    async loadRequest() {
        this.isLoading = true;

        // 1) Read statusId from URL
        const params = new URLSearchParams(window.location.search);
        const statusId = params.get("statusId");
        if (!statusId) {
            alert("Missing statusId in URL");
            this.isLoading = false;
            return;
        }

        try {
            // 2) Fetch request payload
            const res = await fetch(`/ItRequestAPI/request/${statusId}`);
            const ct = res.headers.get('content-type') || '';
            let data, text;

            if (ct.includes('application/json')) {
                data = await res.json();
            } else {
                text = await res.text();
                throw new Error(text || `HTTP ${res.status}`);
            }
            if (!res.ok) throw new Error(data?.message || `HTTP ${res.status}`);

            console.log("RequestReview raw payload:", data);

            // 3) Requester / submit metadata
            // Prefer top-level data.userInfo; if absent, fall back to data.request / data.Request
            const reqSrc = (data.userInfo ?? data.request ?? data.Request ?? {});

            // Many APIs place submit date under status or request; pick the first available
            const submittedAt =
                data.status?.submitDate ?? data.status?.SubmitDate ??
                reqSrc.submitDate ?? reqSrc.SubmitDate ??
                data.status?.firstSubmittedAt ?? data.status?.FirstSubmittedAt ??
                data.request?.firstSubmittedAt ?? data.Request?.FirstSubmittedAt ?? "";

            this.userInfo = {
                staffName: reqSrc.staffName ?? reqSrc.StaffName ?? "",
                departmentName: reqSrc.departmentName ?? reqSrc.DepartmentName ?? "",
                companyName: reqSrc.companyName ?? reqSrc.CompanyName ?? "",
                designation: reqSrc.designation ?? reqSrc.Designation ?? "",
                submitDate: submittedAt
            };
```

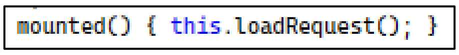


```
// 4) Hardware
this.hardware = (data.hardware ?? []).map(x => ({
    id: x.id ?? x.Id,
    category: x.category ?? x.Category ?? "",
    purpose: x.purpose ?? x.Purpose ?? "",
    justification: x.justification ?? x.Justification ?? "",
    otherDescription: x.otherDescription ?? x.OtherDescription ?? ""
}));

// 5) Emails
this.emails = (data.emails ?? []).map(x => ({
    id: x.id ?? x.Id,
    purpose: x.purpose ?? x.Purpose ?? "",
    proposedAddress: x.proposedAddress ?? x.ProposedAddress ?? ""
}));

// 6) OS requirements
this.osreqs = (data.osreqs ?? data.OSReqs ?? []).map(x => ({
    id: x.id ?? x.Id,
    requirementText: x.requirementText ?? x.RequirementText ?? ""
}));

// 7) Software
this.software = (data.software ?? []).map(x => ({
    id: x.id ?? x.Id,
    bucket: x.bucket ?? x.Bucket ?? "",
    name: x.name ?? x.Name ?? "",
    otherName: x.otherName ?? x.OtherName ?? "",
    notes: x.notes ?? x.Notes ?? ""
}));

// 8) Shared permissions
this.sharedPerms = (data.sharedPerms ?? data.sharedPermissions ?? []).map(x => ({
    id: x.id ?? x.Id,
    shareName: x.shareName ?? x.ShareName ?? "",
    canRead: (x.canRead ?? x.CanRead) || false,
    canWrite: (x.canWrite ?? x.CanWrite) || false,
    canDelete: (x.canDelete ?? x.CanDelete) || false,
    canRemove: (x.canRemove ?? x.CanRemove) || false
}));

// 9) Status block
this.status = {
    hodStatus: data.status?.hodStatus ?? data.status?.HodStatus ?? "Pending",
    gitHodStatus: data.status?.gitHodStatus ?? data.status?.GitHodStatus ?? "Pending",
    finHodStatus: data.status?.finHodStatus ?? data.status?.FinHodStatus ?? "Pending",
    mgmtStatus: data.status?.mgmtStatus ?? data.status?.MgmtStatus ?? "Pending",

    hodSubmitDate: data.status?.hodSubmitDate ?? data.status?.HodSubmitDate ?? "",
    gitHodSubmitDate: data.status?.gitHodSubmitDate ?? data.status?.GitHodSubmitDate ?? "",
    finHodSubmitDate: data.status?.finHodSubmitDate ?? data.status?.FinHodSubmitDate ?? "",
    mgmtSubmitDate: data.status?.mgmtSubmitDate ?? data.status?.MgmtSubmitDate ?? "",

    overallStatus: data.status?.overallStatus ?? data.status?.OverallStatus ?? "Pending",
    canApprove: (data.status?.canApprove ?? data.status?.CanApprove) || false
};
```



```
        const os = (this.status.overallStatus || '').toLowerCase();
        if (os === 'cancelled' || os === 'draft') {
            this.status.canApprove = false;
        }

    } catch (err) {
        console.error("RequestReview fetch failed:", err);
        alert(`Failed to load request: ${err.message}`);
    } finally {
        this.isLoading = false;
    }
},


async updateStatus(decision) {
    const params = new URLSearchParams(window.location.search);
    const statusId = params.get("statusId");
    try {
        const res = await fetch(`/ItRequestAPI/approveReject`, {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({ statusId: parseInt(statusId), decision: decision })
        });
        const ct = res.headers.get('content-type') || '';
        const payload = ct.includes('application/json') ? await res.json() : { message: await res.text() };
        if (!res.ok) throw new Error(payload?.message || `HTTP ${res.status}`);

        // Small UX pop
        const verb = decision === 'Approved' ? 'approved' : 'rejected';
        alert(`Request ${verb} successfully.`);
        this.loadRequest();
    } catch (e) {
        console.error(e);
        alert(`Failed to update status: ${e.message}`);
    }
},

formatDate(dateStr) {
    if (!dateStr) return '';
    try{
        // Show using local timezone, readable
        return new Date(dateStr).toLocaleString();
    }catch{ return dateStr; }
}
```

- ✓ **mounted()**
    - **mounted()**: Calls loadRequest() right away so the screen starts with skeletons and transitions to real content as soon as the payload arrives.



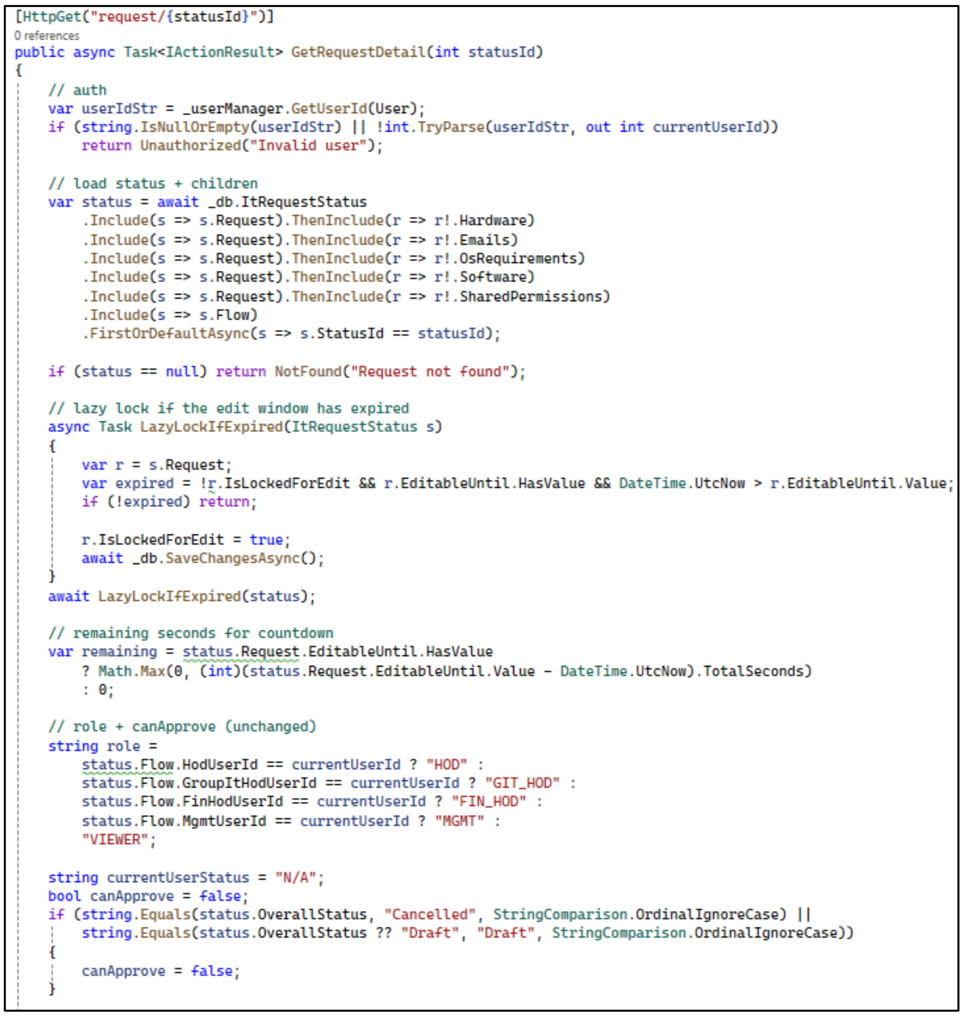
```
mounted() { this.loadRequest(); }
```

- ITRequestAPI (RequestReview.cshtml)

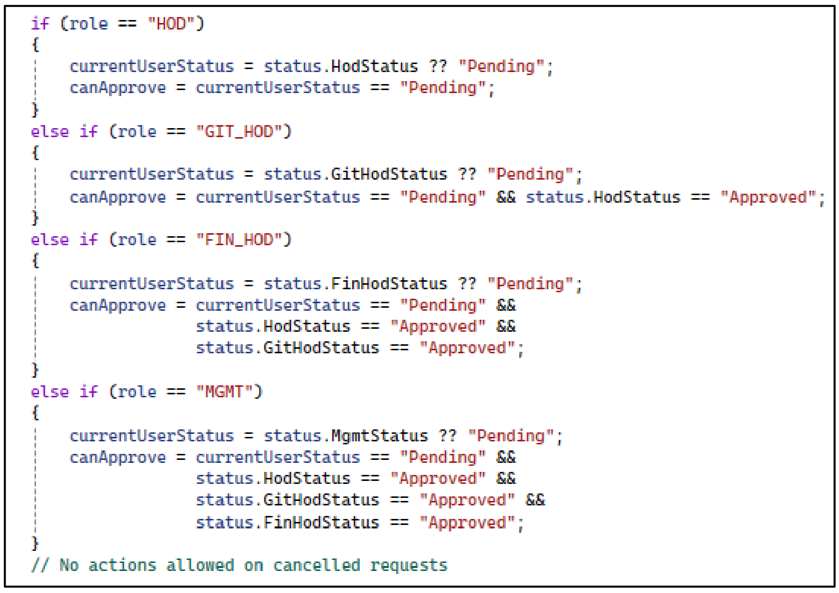
    - **[HttpGet("request/{statusId}")]**

- **Purpose:**
  This endpoint loads everything the Request Review page needs the full request details and its approval status. Basically, when you open an existing request to view or update it, this is the API that supplies all the data.
- **Key functionalities:**
    - **User validation:**
      It first checks who's currently logged in and makes sure the user ID is valid. If not, it immediately returns an unauthorized message.

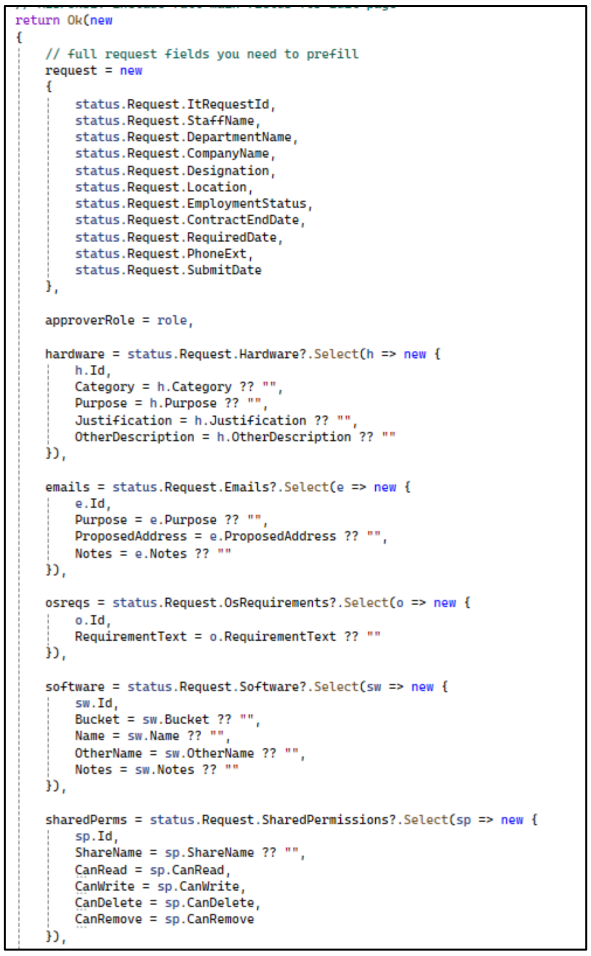
- **Load request and all related data:**
  It fetches the full ItRequestStatus record by statusId, including its related tables:
  - Main request info (Section A)
  - Hardware, Emails, OS Requirements, Software, Shared Permissions
  - Approval Flow (to check roles and statuses)
    If the record doesn't exist, it returns a "Request not found" error.
- **Determine user's role and approval rights:**
  Based on the approval flow, the system checks whether the current user is the HOD, Group IT HOD, Finance HOD, or Management.
  Then it determines two things:
  - currentUserStatus — what stage the user is at (e.g., "Pending", "Approved", "Rejected")
  - canApprove — whether the user can take action right now.
    For example:
  - HOD can act if their status is still Pending.
  - Group IT HOD can only act after HOD has approved.
  - Finance HOD follows after both HOD and Group IT HOD approvals.
  - Management only steps in after everyone else has approved.



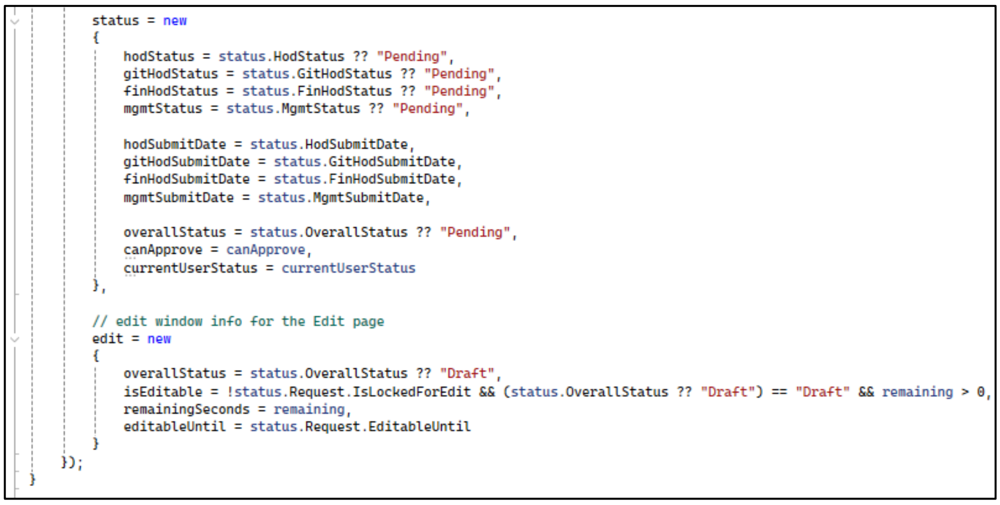
```
[HttpGet("request/{statusId}")]
0 references
public async Task<IActionResult> GetRequestDetail(int statusId)
{
    // auth
    var userIdStr = _userManager.GetUserId(User);
    if (string.IsNullOrEmpty(userIdStr) || !int.TryParse(userIdStr, out int currentUserId))
        return Unauthorized("Invalid user");

    // load status + children
    var status = await _db.ItRequestStatus
        .Include(s => s.Request).ThenInclude(r => r!.Hardware)
        .Include(s => s.Request).ThenInclude(r => r!.Emails)
        .Include(s => s.Request).ThenInclude(r => r!.OsRequirements)
        .Include(s => s.Request).ThenInclude(r => r!.Software)
        .Include(s => s.Request).ThenInclude(r => r!.SharedPermissions)
        .Include(s => s.Flow)
        .FirstOrDefaultAsync(s => s.StatusId == statusId);

    if (status == null) return NotFound("Request not found");

    // lazy lock if the edit window has expired
    async Task LazyLockIfExpired(ItRequestStatus s)
    {
        var r = s.Request;
        var expired = !r.IsLockedForEdit && r.EditableUntil.HasValue && DateTime.UtcNow > r.EditableUntil.Value;
        if (!expired) return;

        r.IsLockedForEdit = true;
        await _db.SaveChangesAsync();
    }
    await LazyLockIfExpired(status);

    // remaining seconds for countdown
    var remaining = status.Request.EditableUntil.HasValue
        ? Math.Max(0, (int)(status.Request.EditableUntil.Value - DateTime.UtcNow).TotalSeconds)
        : 0;

    // role + canApprove (unchanged)
    string role =
        status.Flow.HodUserId == currentUserId ? "HOD" :
        status.Flow.GroupItHodUserId == currentUserId ? "GIT_HOD" :
        status.Flow.FinHodUserId == currentUserId ? "FIN_HOD" :
        status.Flow.MgmtUserId == currentUserId ? "MGMT" :
        "VIEWER";

    string currentUserStatus = "N/A";
    bool canApprove = false;
    if (string.Equals(status.OverallStatus, "Cancelled", StringComparison.OrdinalIgnoreCase) ||
        string.Equals(status.OverallStatus ?? "Draft", "Draft", StringComparison.OrdinalIgnoreCase))
    {
        canApprove = false;
    }
```



```
if (role == "HOD")
{
    currentUserStatus = status.HodStatus ?? "Pending";
    canApprove = currentUserStatus == "Pending";
}
else if (role == "GIT_HOD")
{
    currentUserStatus = status.GitHodStatus ?? "Pending";
    canApprove = currentUserStatus == "Pending" && status.HodStatus == "Approved";
}
else if (role == "FIN_HOD")
{
    currentUserStatus = status.FinHodStatus ?? "Pending";
    canApprove = currentUserStatus == "Pending" &&
                 status.HodStatus == "Approved" &&
                 status.GitHodStatus == "Approved";
}
else if (role == "MGMT")
{
    currentUserStatus = status.MgmtStatus ?? "Pending";
    canApprove = currentUserStatus == "Pending" &&
                 status.HodStatus == "Approved" &&
                 status.GitHodStatus == "Approved" &&
                 status.FinHodStatus == "Approved";
}
// No actions allowed on cancelled requests
```



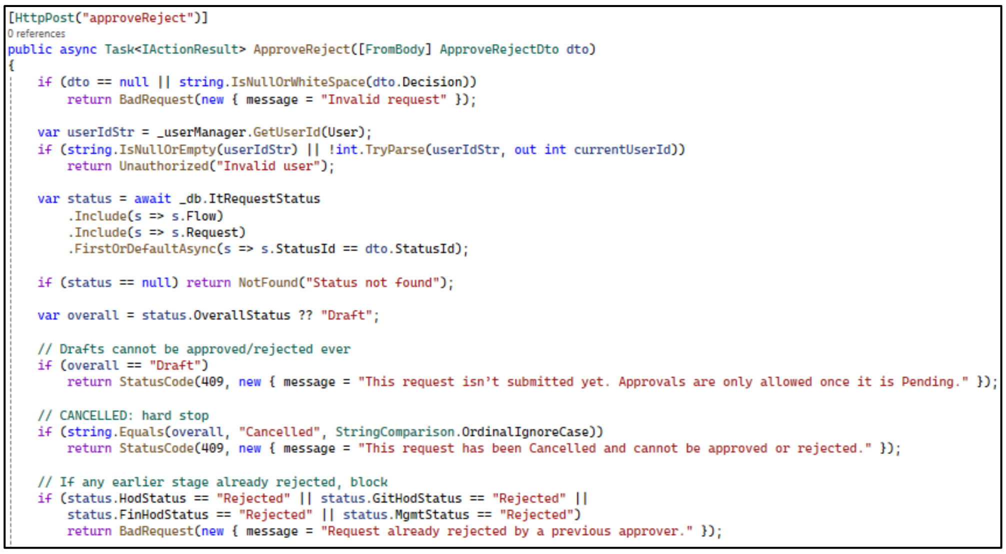
```
return Ok(new
{
    // full request fields you need to prefill
    request = new
    {
        status.Request.ItRequestId,
        status.Request.StaffName,
        status.Request.DepartmentName,
        status.Request.CompanyName,
        status.Request.Designation,
        status.Request.Location,
        status.Request.EmploymentStatus,
        status.Request.ContractEndDate,
        status.Request.RequiredDate,
        status.Request.PhoneExt,
        status.Request.SubmitDate
    },

    approverRole = role,

    hardware = status.Request.Hardware?.Select(h => new {
        h.Id,
        Category = h.Category ?? "",
        Purpose = h.Purpose ?? "",
        Justification = h.Justification ?? "",
        OtherDescription = h.OtherDescription ?? ""
    }),

    emails = status.Request.Emails?.Select(e => new {
        e.Id,
        Purpose = e.Purpose ?? "",
        ProposedAddress = e.ProposedAddress ?? "",
        Notes = e.Notes ?? ""
    }),

    osreqs = status.Request.OsRequirements?.Select(o => new {
        o.Id,
        RequirementText = o.RequirementText ?? ""
    }),

    software = status.Request.Software?.Select(sw => new {
        sw.Id,
        Bucket = sw.Bucket ?? "",
        Name = sw.Name ?? "",
        OtherName = sw.OtherName ?? "",
        Notes = sw.Notes ?? ""
    }),

    sharedPerms = status.Request.SharedPermissions?.Select(sp => new {
        sp.Id,
        ShareName = sp.ShareName ?? "",
        CanRead = sp.CanRead,
        CanWrite = sp.CanWrite,
        CanDelete = sp.CanDelete,
        CanRemove = sp.CanRemove
    }),
```



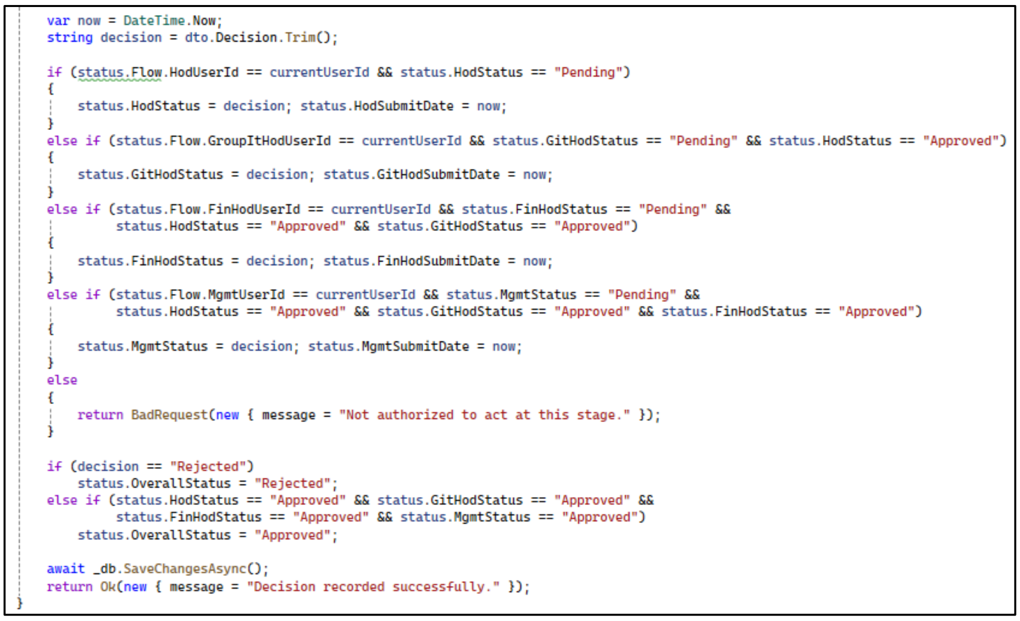
```
        status = new
        {
            hodStatus = status.HodStatus ?? "Pending",
            gitHodStatus = status.GitHodStatus ?? "Pending",
            finHodStatus = status.FinHodStatus ?? "Pending",
            mgmtStatus = status.MgmtStatus ?? "Pending",

            hodSubmitDate = status.HodSubmitDate,
            gitHodSubmitDate = status.GitHodSubmitDate,
            finHodSubmitDate = status.FinHodSubmitDate,
            mgmtSubmitDate = status.MgmtSubmitDate,

            overallStatus = status.OverallStatus ?? "Pending",
            canApprove = canApprove,
            currentUserStatus = currentUserStatus
        },

        // edit window info for the Edit page
        edit = new
        {
            overallStatus = status.OverallStatus ?? "Draft",
            isEditable = !status.Request.IsLockedForEdit && (status.OverallStatus ?? "Draft") == "Draft" && remaining > 0,
            remainingSeconds = remaining,
            editableUntil = status.Request.EditableUntil
        }
    });
}
```

- **[HttpPost("approveReject")]**
- **Purpose:** Records the approver's decision (Approve/Reject) for one IT request, enforcing the correct stage order.

- **Key functionality:**
    - **Auth & input:** Reads currentUserId from Identity; rejects empty payload or missing Decision.
    - **Load record:** Gets ItRequestStatus (with Flow and Request) by StatusId.
    - **Hard guards:**
        - Blocks if OverallStatus is Draft (not submitted) or Cancelled.
        - Blocks if any prior step already **Rejected**.
    - **Stage-by-stage routing:**
        - **HOD** may act only when HodStatus == "Pending".
        - **GIT_HOD** only after HOD Approved and GitHodStatus == "Pending".
        - **FIN_HOD** only after HOD & GIT_HOD Approved and FinHodStatus == "Pending".
        - **MGMT** only after HOD, GIT_HOD, FIN_HOD Approved and MgmtStatus == "Pending".
        - If none match, returns “Not authorized to act at this stage.”
    - **Overall status:**
        - If decision is **Rejected** → OverallStatus = "Rejected".
        - If all four steps are **Approved** → OverallStatus = "Approved".
    - **Audit fields:** Sets the corresponding *SubmitDate = now on the step that acted.

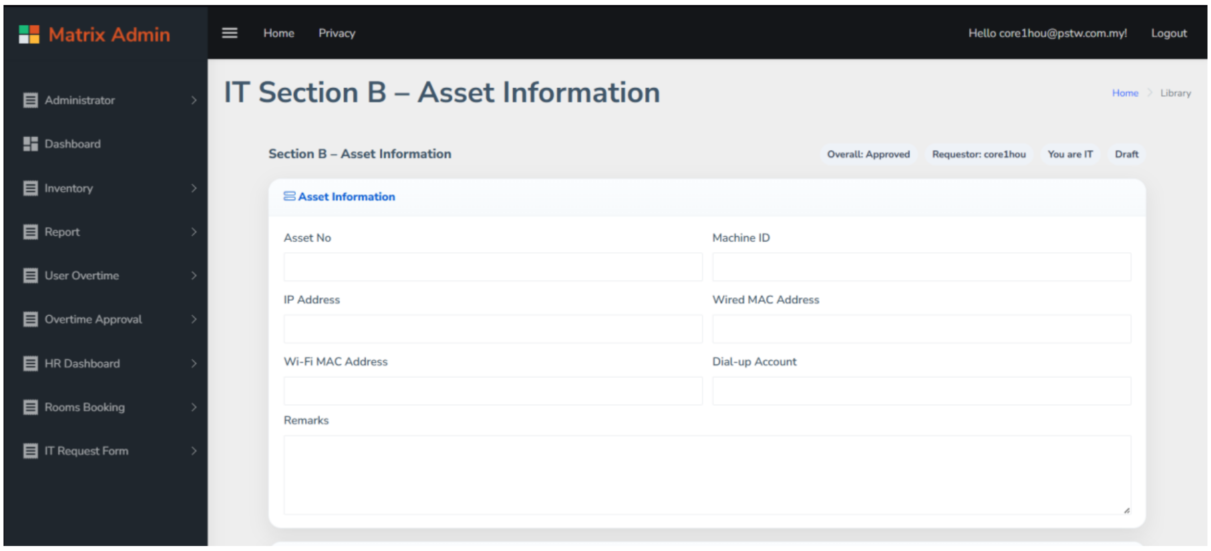


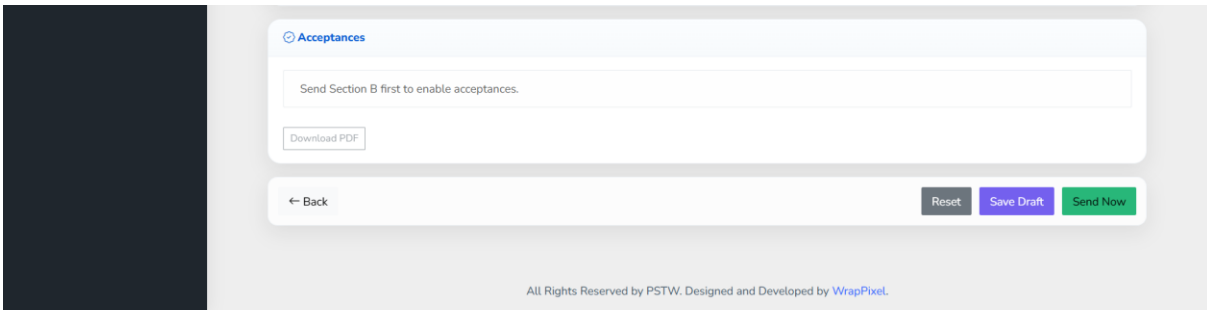
```
[HttpPost("approveReject")]
0 references
public async Task<IActionResult> ApproveReject([FromBody] ApproveRejectDto dto)
{
    if (dto == null || string.IsNullOrWhiteSpace(dto.Decision))
        return BadRequest(new { message = "Invalid request" });

    var userIdStr = _userManager.GetUserId(User);
    if (string.IsNullOrEmpty(userIdStr) || !int.TryParse(userIdStr, out int currentUserId))
        return Unauthorized("Invalid user");

    var status = await _db.ItRequestStatus
        .Include(s => s.Flow)
        .Include(s => s.Request)
        .FirstOrDefaultAsync(s => s.StatusId == dto.StatusId);

    if (status == null) return NotFound("Status not found");

    var overall = status.OverallStatus ?? "Draft";

    // Drafts cannot be approved/rejected ever
    if (overall == "Draft")
        return StatusCode(409, new { message = "This request isn't submitted yet. Approvals are only allowed once it is Pending." });

    // CANCELLED: hard stop
    if (string.Equals(overall, "Cancelled", StringComparison.OrdinalIgnoreCase))
        return StatusCode(409, new { message = "This request has been Cancelled and cannot be approved or rejected." });

    // If any earlier stage already rejected, block
    if (status.HodStatus == "Rejected" || status.GitHodStatus == "Rejected" ||
        status.FinHodStatus == "Rejected" || status.MgmtStatus == "Rejected")
        return BadRequest(new { message = "Request already rejected by a previous approver." });
```



```
    var now = DateTime.Now;
    string decision = dto.Decision.Trim();

    if (status.Flow.HodUserId == currentUserId && status.HodStatus == "Pending")
    {
        status.HodStatus = decision; status.HodSubmitDate = now;
    }
    else if (status.Flow.GroupItHodUserId == currentUserId && status.GitHodStatus == "Pending" && status.HodStatus == "Approved")
    {
        status.GitHodStatus = decision; status.GitHodSubmitDate = now;
    }
    else if (status.Flow.FinHodUserId == currentUserId && status.FinHodStatus == "Pending" &&
             status.HodStatus == "Approved" && status.GitHodStatus == "Approved")
    {
        status.FinHodStatus = decision; status.FinHodSubmitDate = now;
    }
    else if (status.Flow.MgmtUserId == currentUserId && status.MgmtStatus == "Pending" &&
             status.HodStatus == "Approved" && status.GitHodStatus == "Approved" && status.FinHodStatus == "Approved")
    {
        status.MgmtStatus = decision; status.MgmtSubmitDate = now;
    }
    else
    {
        return BadRequest(new { message = "Not authorized to act at this stage." });
    }

    if (decision == "Rejected")
        status.OverallStatus = "Rejected";
    else if (status.HodStatus == "Approved" && status.GitHodStatus == "Approved" &&
             status.FinHodStatus == "Approved" && status.MgmtStatus == "Approved")
        status.OverallStatus = "Approved";

    await _db.SaveChangesAsync();
    return Ok(new { message = "Decision recorded successfully." });
}
```

- SectionB.cshtml

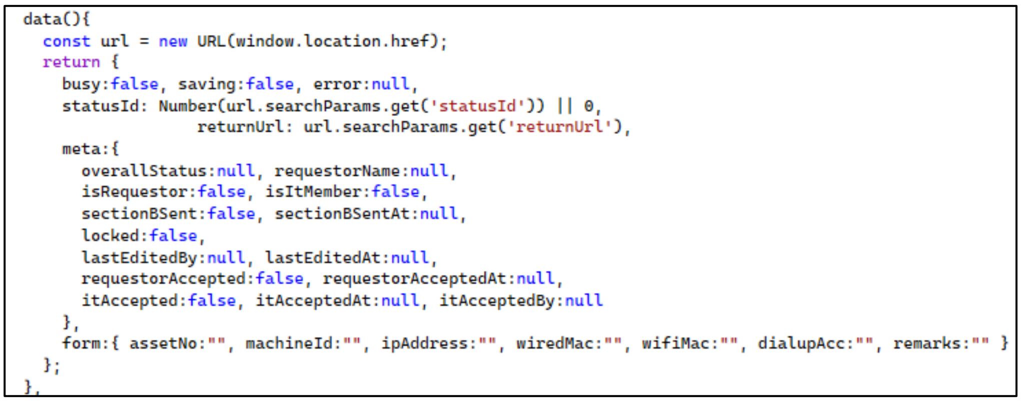



Acceptances
Send Section B first to enable acceptances.
Download PDF
← Back
Reset
Save Draft
Send Now
All Rights Reserved by PSTW. Designed and Developed by WrapPixel.

This file (SectionB.cshtml) is where IT fills in asset details after the main request is approved, then sends it for dual acceptance (requestor and IT). The page shows your role, whether Section B is a draft or sent, and locks fields once sent.

- ✓ **<div id="bApp">…</div>**
  - **Purpose:** Root container for the Vue app so the cards, inputs, and buttons stay reactive.
- ✓ **const bApp = Vue.createApp({...})**
  - **Purpose:** Boots the Vue instance and wires up form state, meta flags, API calls, and lifecycle.
- ✓ **data()**
  - **busy, saving, error**: Control spinners and alerts during load and save actions. saving is dedicated to the Save Draft button so users get clear feedback.
  - **statusId, returnUrl**: Parsed from the URL at startup. statusId targets the current request and returnUrl (if present and safe) lets the Back button return you to the exact view you came from.

- **meta { overallStatus, requestorName, isRequestor, isItMember, sectionBSent, locked, lastEditedBy, lastEditedAt, requestorAccepted, requestorAcceptedAt, itAccepted, itAcceptedAt, itAcceptedBy }**: A compact snapshot of Section B and who can act on it. overallStatus must be Approved for the editor to unlock, isItMember gates editing, sectionBSent and locked flip the form into read-only for acceptance, and the acceptance flags drive the badges and the PDF button.
- **form { assetNo, machineId, ipAddress, wiredMac, wifiMac, dialupAcc, remarks }**: The actual asset fields IT records. All fields are disabled for non-IT users and once locked is true after sending.

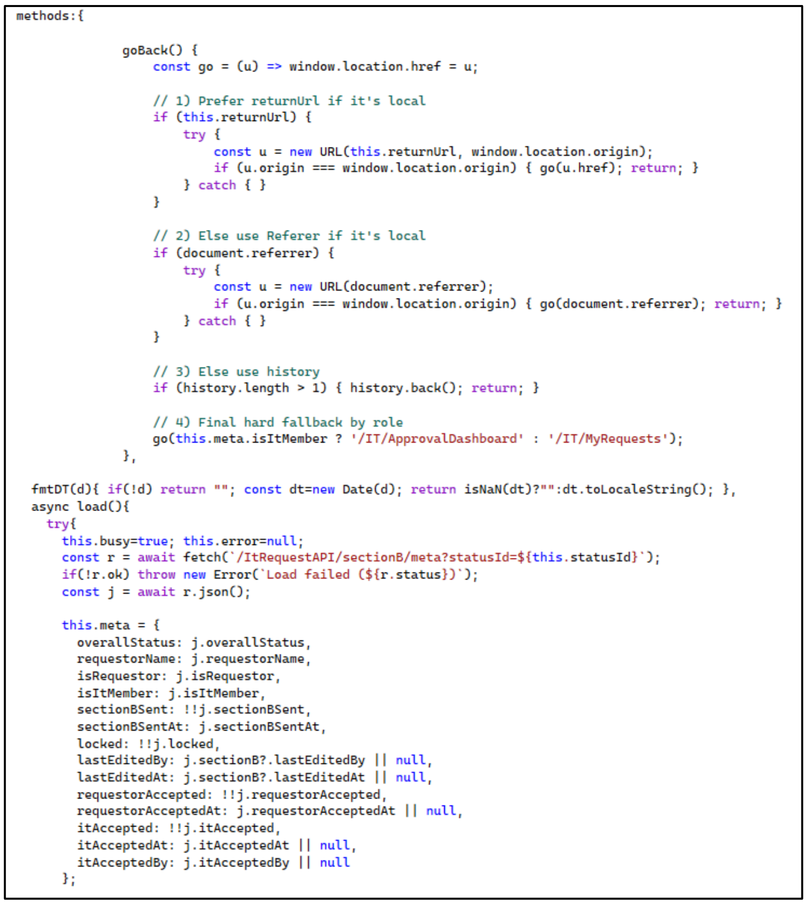


```
data(){
  const url = new URL(window.location.href);
  return {
    busy:false, saving:false, error:null,
    statusId: Number(url.searchParams.get('statusId')) || 0,
              returnUrl: url.searchParams.get('returnUrl'),
    meta:{
      overallStatus:null, requestorName:null,
      isRequestor:false, isItMember:false,
      sectionBSent:false, sectionBSentAt:null,
      locked:false,
      lastEditedBy:null, lastEditedAt:null,
      requestorAccepted:false, requestorAcceptedAt:null,
      itAccepted:false, itAcceptedAt:null, itAcceptedBy:null
    },
    form:{ assetNo:"", machineId:"", ipAddress:"", wiredMac:"", wifiMac:"", dialupAcc:"", remarks:"" }
  };
},
```

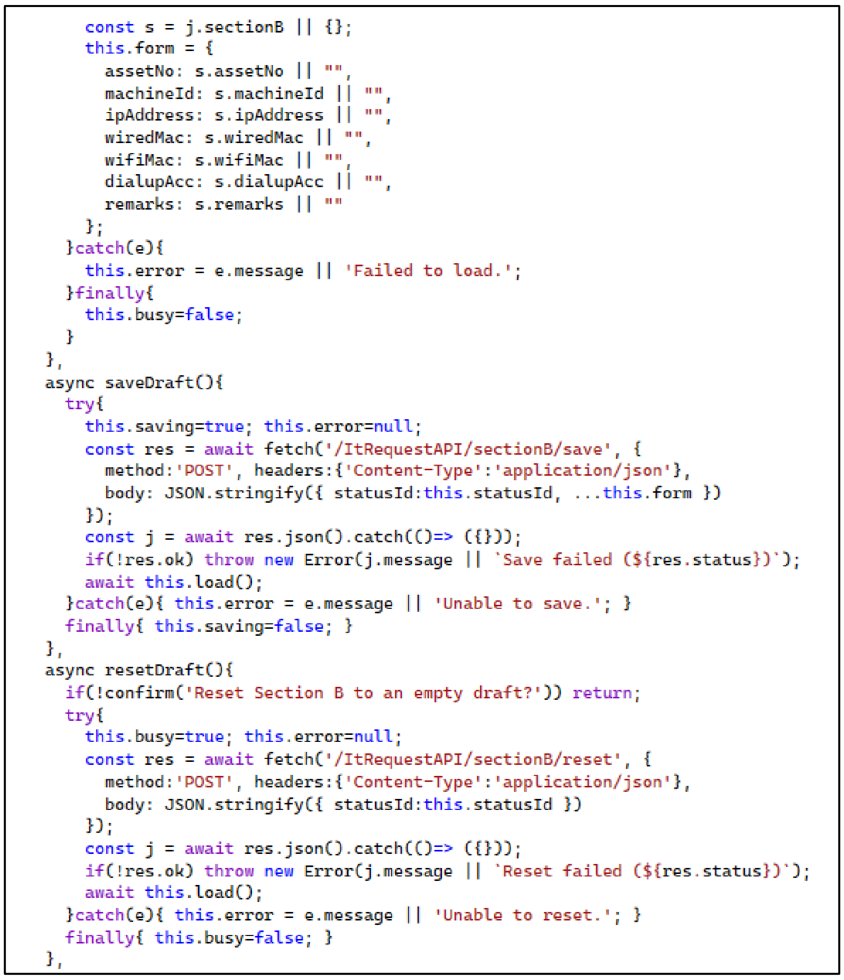
✓ **methods**

- **goBack()**: Navigates back in a safe order: trusted returnUrl if it is same-origin, then the browser referrer if local, then history back if available, and finally a role-based fallback route. This avoids sending users to unknown sites.
- **fmtDT(d)**: Formats timestamps using the browser locale with a graceful empty string if parsing fails. Keeps the header "Last edited" and acceptance dates readable.
- **load()**: Fetches /ItRequestAPI/sectionB/meta?statusId=..., hydrates meta, and fills the form from any existing draft. It also normalizes booleans and dates so the UI renders consistent badges and lock state.
- **saveDraft()**: Posts the current form to /ItRequestAPI/sectionB/save. On success it reloads the meta so the "Last edited" pill reflects the latest editor and time.

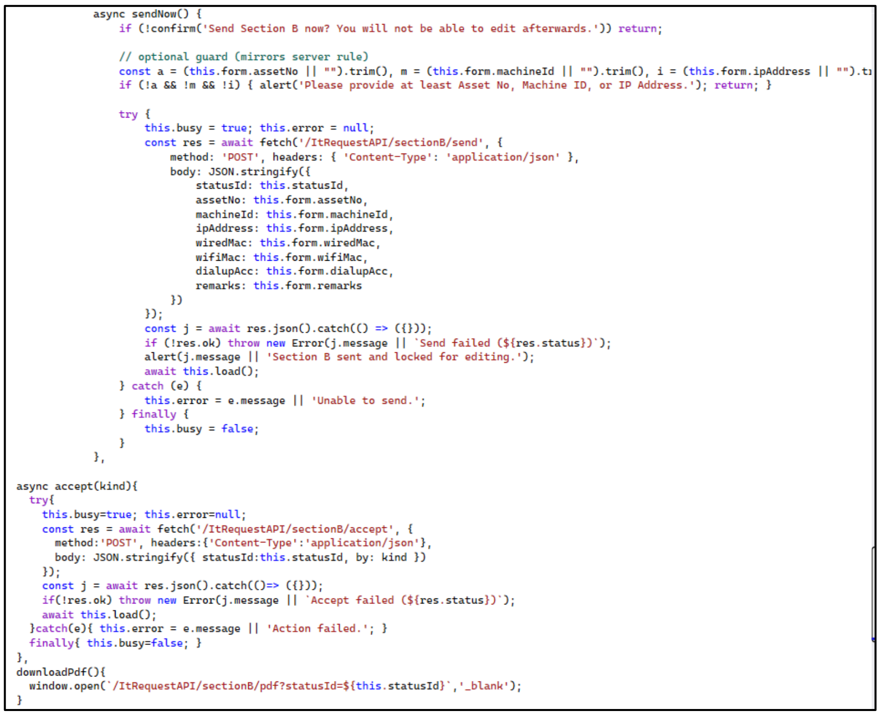
- **resetDraft()**: Confirms, then posts to /ItRequestAPI/sectionB/reset to clear all Section B fields back to a blank draft. Reloads the page state so the form and meta are in sync.
- **sendNow()**: Confirms intent, does a light guard that at least one identifier is present (Asset No or Machine ID or IP Address), then posts to /ItRequestAPI/sectionB/send. A successful send locks the form, flips sectionBSent to true, and enables acceptances.
- **accept(kind)**: Records an acceptance by either the Requestor or IT using /ItRequestAPI/sectionB/accept with by: 'REQUESTOR' or by: 'IT'. After saving it reloads only the meta and form so badges update immediately. Buttons disable themselves when already accepted.
- **downloadPdf():** Opens the Section B PDF in a new tab once both acceptances are done. Keeps your place on the page.



```
methods:{

              goBack() {
                  const go = (u) => window.location.href = u;

                  // 1) Prefer returnUrl if it's local
                  if (this.returnUrl) {
                      try {
                          const u = new URL(this.returnUrl, window.location.origin);
                          if (u.origin === window.location.origin) { go(u.href); return; }
                      } catch { }
                  }

                  // 2) Else use Referer if it's local
                  if (document.referrer) {
                      try {
                          const u = new URL(document.referrer);
                          if (u.origin === window.location.origin) { go(document.referrer); return; }
                      } catch { }
                  }

                  // 3) Else use history
                  if (history.length > 1) { history.back(); return; }

                  // 4) Final hard fallback by role
                  go(this.meta.isItMember ? '/IT/ApprovalDashboard' : '/IT/MyRequests');
              },

  fmtDT(d){ if(!d) return ""; const dt=new Date(d); return isNaN(dt)?"":dt.toLocaleString(); },
  async load(){
    try{
      this.busy=true; this.error=null;
      const r = await fetch(`/ItRequestAPI/sectionB/meta?statusId=${this.statusId}`);
      if(!r.ok) throw new Error(`Load failed (${r.status})`);
      const j = await r.json();

      this.meta = {
        overallStatus: j.overallStatus,
        requestorName: j.requestorName,
        isRequestor: j.isRequestor,
        isItMember: j.isItMember,
        sectionBSent: !!j.sectionBSent,
        sectionBSentAt: j.sectionBSentAt,
        locked: !!j.locked,
        lastEditedBy: j.sectionB?.lastEditedBy || null,
        lastEditedAt: j.sectionB?.lastEditedAt || null,
        requestorAccepted: !!j.requestorAccepted,
        requestorAcceptedAt: j.requestorAcceptedAt || null,
        itAccepted: !!j.itAccepted,
        itAcceptedAt: j.itAcceptedAt || null,
        itAcceptedBy: j.itAcceptedBy || null
      };
```



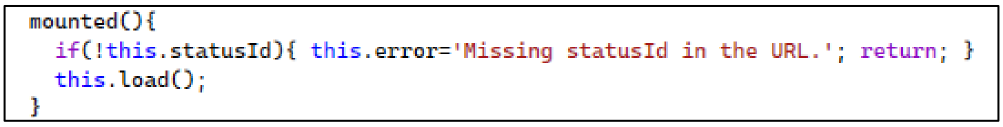
```
        const s = j.sectionB || {};
        this.form = {
          assetNo: s.assetNo || "",
          machineId: s.machineId || "",
          ipAddress: s.ipAddress || "",
          wiredMac: s.wiredMac || "",
          wifiMac: s.wifiMac || "",
          dialupAcc: s.dialupAcc || "",
          remarks: s.remarks || ""
        };
      }catch(e){
        this.error = e.message || 'Failed to load.';
      }finally{
        this.busy=false;
      }
    },
    async saveDraft(){
      try{
        this.saving=true; this.error=null;
        const res = await fetch('/ItRequestAPI/sectionB/save', {
          method:'POST', headers:{'Content-Type':'application/json'},
          body: JSON.stringify({ statusId:this.statusId, ...this.form })
        });
        const j = await res.json().catch(()=> ({}));
        if(!res.ok) throw new Error(j.message || `Save failed (${res.status})`);
        await this.load();
      }catch(e){ this.error = e.message || 'Unable to save.'; }
      finally{ this.saving=false; }
    },
    async resetDraft(){
      if(!confirm('Reset Section B to an empty draft?')) return;
      try{
        this.busy=true; this.error=null;
        const res = await fetch('/ItRequestAPI/sectionB/reset', {
          method:'POST', headers:{'Content-Type':'application/json'},
          body: JSON.stringify({ statusId:this.statusId })
        });
        const j = await res.json().catch(()=> ({}));
        if(!res.ok) throw new Error(j.message || `Reset failed (${res.status})`);
        await this.load();
      }catch(e){ this.error = e.message || 'Unable to reset.'; }
      finally{ this.busy=false; }
    },
```



```
            async sendNow() {
                if (!confirm('Send Section B now? You will not be able to edit afterwards.')) return;

                // optional guard (mirrors server rule)
                const a = (this.form.assetNo || "").trim(), m = (this.form.machineId || "").trim(), i = (this.form.ipAddress || "").t
                if (!a && !m && !i) { alert('Please provide at least Asset No, Machine ID, or IP Address.'); return; }

                try {
                    this.busy = true; this.error = null;
                    const res = await fetch('/ItRequestAPI/sectionB/send', {
                        method: 'POST', headers: { 'Content-Type': 'application/json' },
                        body: JSON.stringify({
                            statusId: this.statusId,
                            assetNo: this.form.assetNo,
                            machineId: this.form.machineId,
                            ipAddress: this.form.ipAddress,
                            wiredMac: this.form.wiredMac,
                            wifiMac: this.form.wifiMac,
                            dialupAcc: this.form.dialupAcc,
                            remarks: this.form.remarks
                        })
                    });
                    const j = await res.json().catch(() => ({}));
                    if (!res.ok) throw new Error(j.message || `Send failed (${res.status})`);
                    alert(j.message || 'Section B sent and locked for editing.');
                    await this.load();
                } catch (e) {
                    this.error = e.message || 'Unable to send.';
                } finally {
                    this.busy = false;
                }
            },

async accept(kind){
  try{
    this.busy=true; this.error=null;
    const res = await fetch('/ItRequestAPI/sectionB/accept', {
      method:'POST', headers:{'Content-Type':'application/json'},
      body: JSON.stringify({ statusId:this.statusId, by: kind })
    });
    const j = await res.json().catch(()=> ({}));
    if(!res.ok) throw new Error(j.message || `Accept failed (${res.status})`);
    await this.load();
  }catch(e){ this.error = e.message || 'Action failed.'; }
  finally{ this.busy=false; }
},
downloadPdf(){
  window.open(`/ItRequestAPI/sectionB/pdf?statusId=${this.statusId}`,'_blank');
}
```

✓ **mounted**

- **mounted()**: Validates that statusId exists, then calls load() right away. The page shows a gate warning until the overall status is Approved. After approval, IT can edit, save, and send, followed by requestor and IT acceptances.

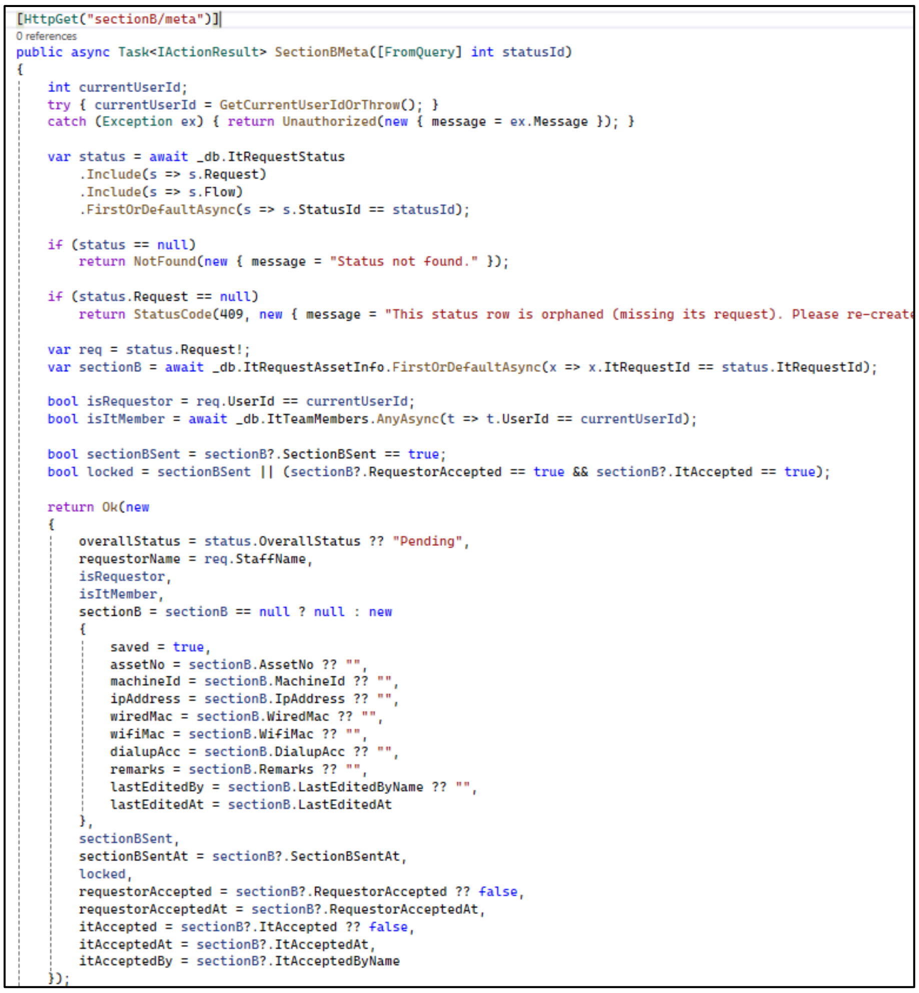


```
mounted(){
  if(!this.statusId){ this.error='Missing statusId in the URL.'; return; }
  this.load();
}
```

- ITRequestAPI (SectionB.cshtml)

- **[HttpGet("sectionB/meta")]** — **Used by:** SectionB.cshtml
- **Purpose:**
  Load the Section B header info and permissions for a given request. The page uses this to know who the viewer is (requestor or IT), whether Section B was sent, whether the form is locked, and to prefill any saved asset details.
- **Key functionalities:**
  - Checks who is logged in and gets their user id. If that fails, it returns Unauthorized with a clear message.
  - Fetches the ItRequestStatus by statusId together with its Request and Flow. If there is no status row it returns NotFound. If the status exists but the linked request is missing, it returns a 409 explaining the orphaned row.
  - Looks up the Section B row in ItRequestAssetInfo for this request. This may be null if nobody has started Section B yet.
  - Computes role flags:
    - isRequestor if the current user owns the request
    - isItMember if the user appears in the IT team members table
  - Determines Section B state:
    - sectionBSent is true when the user pressed “Send” in Section B
    - locked becomes true if Section B has been sent, or if both sides have accepted (RequestorAccepted and ItAccepted)

- Shapes a compact payload for the page:
    - Top level: overallStatus, requestorName, role flags, sectionBSent, timestamps, locked, individual acceptance flags and names
    - Nested sectionB object (only when it exists) with assetNo, machineId, ipAddress, MACs, dial-up account, remarks, and last edited metadata
- The page can use locked to disable inputs, isRequestor and isItMember to show the right buttons, and the sectionB fields to prefill the form if it was previously saved.



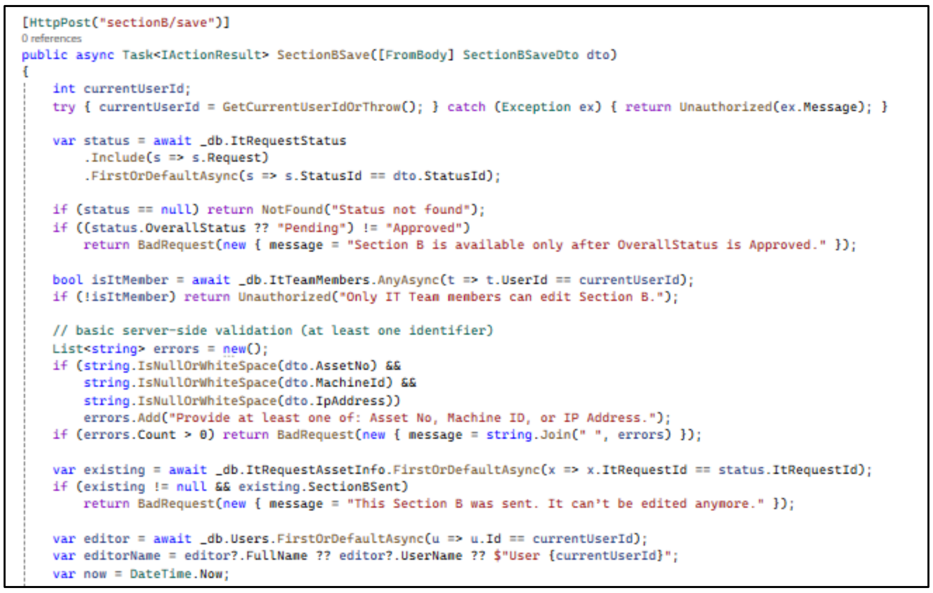
```
[HttpGet("sectionB/meta")]
0 references
public async Task<IActionResult> SectionBMeta([FromQuery] int statusId)
{
    int currentUserId;
    try { currentUserId = GetCurrentUserIdOrThrow(); }
    catch (Exception ex) { return Unauthorized(new { message = ex.Message }); }

    var status = await _db.ItRequestStatus
        .Include(s => s.Request)
        .Include(s => s.Flow)
        .FirstOrDefaultAsync(s => s.StatusId == statusId);

    if (status == null)
        return NotFound(new { message = "Status not found." });

    if (status.Request == null)
        return StatusCode(409, new { message = "This status row is orphaned (missing its request). Please re-create

    var req = status.Request!;
    var sectionB = await _db.ItRequestAssetInfo.FirstOrDefaultAsync(x => x.ItRequestId == status.ItRequestId);

    bool isRequestor = req.UserId == currentUserId;
    bool isItMember = await _db.ItTeamMembers.AnyAsync(t => t.UserId == currentUserId);

    bool sectionBSent = sectionB?.SectionBSent == true;
    bool locked = sectionBSent || (sectionB?.RequestorAccepted == true && sectionB?.ItAccepted == true);

    return Ok(new
    {
        overallStatus = status.OverallStatus ?? "Pending",
        requestorName = req.StaffName,
        isRequestor,
        isItMember,
        sectionB = sectionB == null ? null : new
        {
            saved = true,
            assetNo = sectionB.AssetNo ?? "",
            machineId = sectionB.MachineId ?? "",
            ipAddress = sectionB.IpAddress ?? "",
            wiredMac = sectionB.WiredMac ?? "",
            wifiMac = sectionB.WifiMac ?? "",
            dialupAcc = sectionB.DialupAcc ?? "",
            remarks = sectionB.Remarks ?? "",
            lastEditedBy = sectionB.LastEditedByName ?? "",
            lastEditedAt = sectionB.LastEditedAt
        },
        sectionBSent,
        sectionBSentAt = sectionB?.SectionBSentAt,
        locked,
        requestorAccepted = sectionB?.RequestorAccepted ?? false,
        requestorAcceptedAt = sectionB?.RequestorAcceptedAt,
        itAccepted = sectionB?.ItAccepted ?? false,
        itAcceptedAt = sectionB?.ItAcceptedAt,
        itAcceptedBy = sectionB?.ItAcceptedByName
    });
```

- **[HttpPost("sectionB/save")] — Used by:** SectionB.cshtml (Save Draft)

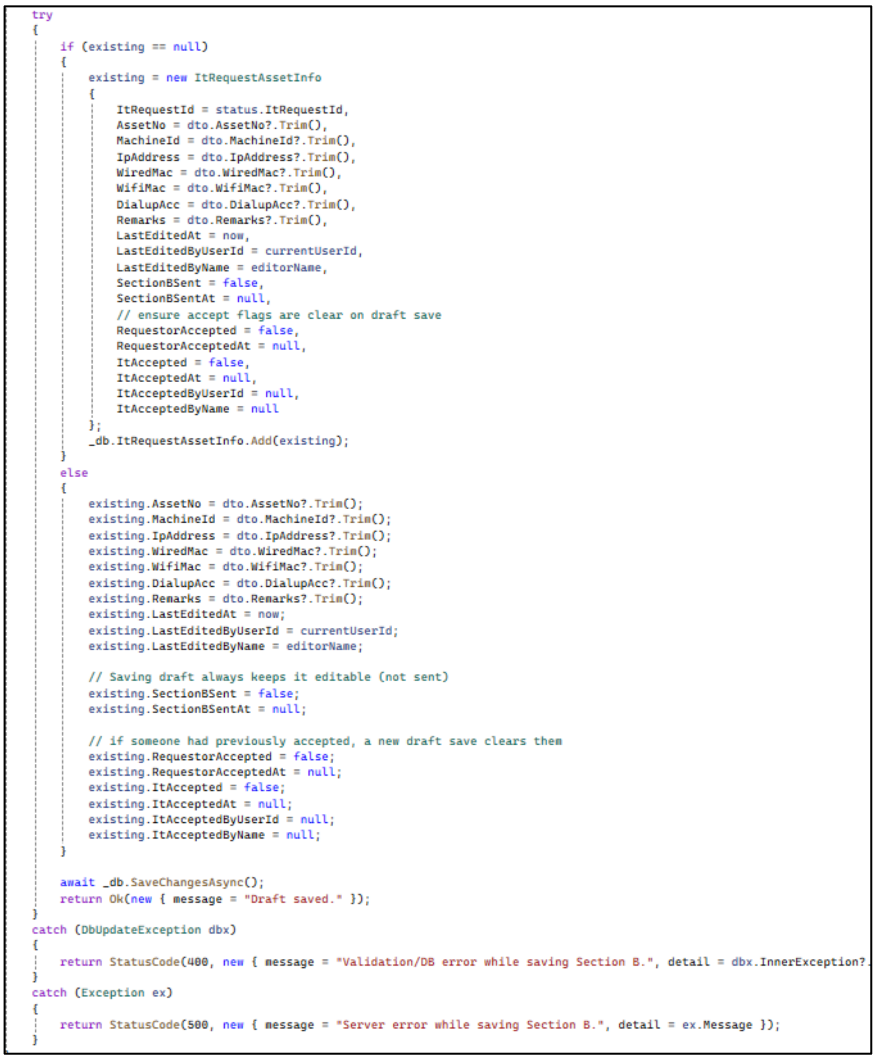
- **Purpose:**

  Save or update Section B as a draft. This is for IT team members to record asset details after the request is fully approved at the Section A level. Saving a draft keeps Section B editable and resets any acceptances so both sides can review again.

- **Key functionalities:**
  - Authenticates the caller and confirms they are an IT team member. If not, it blocks the save.
  - Loads the status by StatusId and checks that OverallStatus is **Approved**. Section B is only available after the main request clears all approvals.
  - Validates the payload. At least one identifier must be provided: Asset No, Machine ID, or IP Address. If all three are blank, it returns a friendly validation message.
  - Fetches the existing ItRequestAssetInfo row for this request.
    - If a Section B already exists and has been sent, the endpoint refuses edits because sent records are locked.
    - If there is no record yet, it creates a new one and fills in all fields, including editor name and timestamp.
    - If there is a record, it updates all fields and stamps LastEditedBy and LastEditedAt.
  - Saves as a draft. It always sets SectionBSent = false and clears any previous acceptances: RequestorAccepted, ItAccepted, and their timestamps and by-name fields. This ensures both the requestor and IT will re-acknowledge after changes.
  - On success, returns a simple { message: "Draft saved." }. On database or validation issues, returns clear error messages so the UI can show a toast.



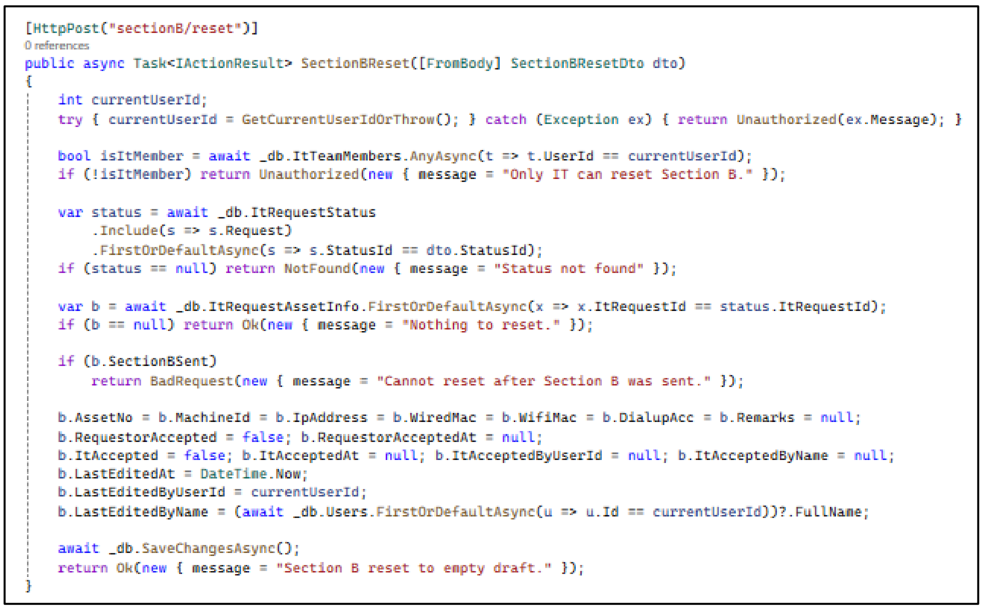
```
[HttpPost("sectionB/save")]
0 references
public async Task<IActionResult> SectionBSave([FromBody] SectionBSaveDto dto)
{
    int currentUserId;
    try { currentUserId = GetCurrentUserIdOrThrow(); } catch (Exception ex) { return Unauthorized(ex.Message); }

    var status = await _db.ItRequestStatus
        .Include(s => s.Request)
        .FirstOrDefaultAsync(s => s.StatusId == dto.StatusId);

    if (status == null) return NotFound("Status not found");
    if ((status.OverallStatus ?? "Pending") != "Approved")
        return BadRequest(new { message = "Section B is available only after OverallStatus is Approved." });

    bool isItMember = await _db.ItTeamMembers.AnyAsync(t => t.UserId == currentUserId);
    if (!isItMember) return Unauthorized("Only IT Team members can edit Section B.");

    // basic server-side validation (at least one identifier)
    List<string> errors = new();
    if (string.IsNullOrWhiteSpace(dto.AssetNo) &&
        string.IsNullOrWhiteSpace(dto.MachineId) &&
        string.IsNullOrWhiteSpace(dto.IpAddress))
        errors.Add("Provide at least one of: Asset No, Machine ID, or IP Address.");
    if (errors.Count > 0) return BadRequest(new { message = string.Join(" ", errors) });

    var existing = await _db.ItRequestAssetInfo.FirstOrDefaultAsync(x => x.ItRequestId == status.ItRequestId);
    if (existing != null && existing.SectionBSent)
        return BadRequest(new { message = "This Section B was sent. It can't be edited anymore." });

    var editor = await _db.Users.FirstOrDefaultAsync(u => u.Id == currentUserId);
    var editorName = editor?.FullName ?? editor?.UserName ?? $"User {currentUserId}";
    var now = DateTime.Now;
```



```
    try
    {
        if (existing == null)
        {
            existing = new ItRequestAssetInfo
            {
                ItRequestId = status.ItRequestId,
                AssetNo = dto.AssetNo?.Trim(),
                MachineId = dto.MachineId?.Trim(),
                IpAddress = dto.IpAddress?.Trim(),
                WiredMac = dto.WiredMac?.Trim(),
                WifiMac = dto.WifiMac?.Trim(),
                DialupAcc = dto.DialupAcc?.Trim(),
                Remarks = dto.Remarks?.Trim(),
                LastEditedAt = now,
                LastEditedByUserId = currentUserId,
                LastEditedByName = editorName,
                SectionBSent = false,
                SectionBSentAt = null,
                // ensure accept flags are clear on draft save
                RequestorAccepted = false,
                RequestorAcceptedAt = null,
                ItAccepted = false,
                ItAcceptedAt = null,
                ItAcceptedByUserId = null,
                ItAcceptedByName = null
            };
            _db.ItRequestAssetInfo.Add(existing);
        }
        else
        {
            existing.AssetNo = dto.AssetNo?.Trim();
            existing.MachineId = dto.MachineId?.Trim();
            existing.IpAddress = dto.IpAddress?.Trim();
            existing.WiredMac = dto.WiredMac?.Trim();
            existing.WifiMac = dto.WifiMac?.Trim();
            existing.DialupAcc = dto.DialupAcc?.Trim();
            existing.Remarks = dto.Remarks?.Trim();
            existing.LastEditedAt = now;
            existing.LastEditedByUserId = currentUserId;
            existing.LastEditedByName = editorName;

            // Saving draft always keeps it editable (not sent)
            existing.SectionBSent = false;
            existing.SectionBSentAt = null;

            // if someone had previously accepted, a new draft save clears them
            existing.RequestorAccepted = false;
            existing.RequestorAcceptedAt = null;
            existing.ItAccepted = false;
            existing.ItAcceptedAt = null;
            existing.ItAcceptedByUserId = null;
            existing.ItAcceptedByName = null;
        }

        await _db.SaveChangesAsync();
        return Ok(new { message = "Draft saved." });
    }
    catch (DbUpdateException dbx)
    {
        return StatusCode(400, new { message = "Validation/DB error while saving Section B.", detail = dbx.InnerException?.
    }
    catch (Exception ex)
    {
        return StatusCode(500, new { message = "Server error while saving Section B.", detail = ex.Message });
    }
```

- **[HttpPost("sectionB/reset")] Used by:** SectionB.cshtml (Reset button)

- **Purpose:**
  Clears the current Section B draft back to a blank state. This is an IT-only action and only works while Section B hasn't been sent yet.
- **Key functionalities:**
  - Verifies the logged-in user and confirms they are part of the IT team. Non-IT users are blocked.
  - Loads the status by StatusId and then looks up the matching Section B row.
    - If there is no Section B row yet, it just returns "Nothing to reset."
    - If Section B was already sent, it refuses to reset because sent records are locked.
  - Wipes all editable fields in Section B: asset number, machine ID, IP address, wired and Wi-Fi MAC, dial-up account, and remarks.
  - Clears both acceptance states so the requestor and IT will need to re-acknowledge after new data is entered.
  - Updates the "last edited" metadata with the current IT user and time, saves, and returns a simple success message.

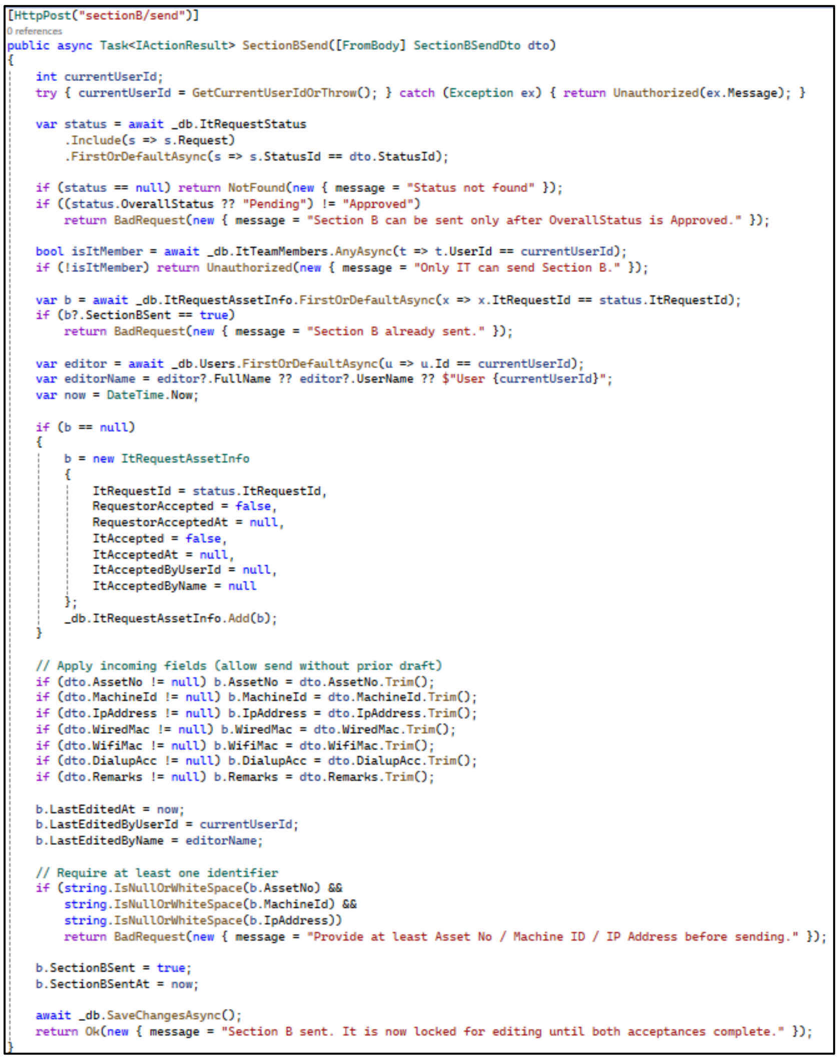


```
[HttpPost("sectionB/reset")]
0 references
public async Task<IActionResult> SectionBReset([FromBody] SectionBResetDto dto)
{
    int currentUserId;
    try { currentUserId = GetCurrentUserIdOrThrow(); } catch (Exception ex) { return Unauthorized(ex.Message); }

    bool isItMember = await _db.ItTeamMembers.AnyAsync(t => t.UserId == currentUserId);
    if (!isItMember) return Unauthorized(new { message = "Only IT can reset Section B." });

    var status = await _db.ItRequestStatus
        .Include(s => s.Request)
        .FirstOrDefaultAsync(s => s.StatusId == dto.StatusId);
    if (status == null) return NotFound(new { message = "Status not found" });

    var b = await _db.ItRequestAssetInfo.FirstOrDefaultAsync(x => x.ItRequestId == status.ItRequestId);
    if (b == null) return Ok(new { message = "Nothing to reset." });

    if (b.SectionBSent)
        return BadRequest(new { message = "Cannot reset after Section B was sent." });

    b.AssetNo = b.MachineId = b.IpAddress = b.WiredMac = b.WifiMac = b.DialupAcc = b.Remarks = null;
    b.RequestorAccepted = false; b.RequestorAcceptedAt = null;
    b.ItAccepted = false; b.ItAcceptedAt = null; b.ItAcceptedByUserId = null; b.ItAcceptedByName = null;
    b.LastEditedAt = DateTime.Now;
    b.LastEditedByUserId = currentUserId;
    b.LastEditedByName = (await _db.Users.FirstOrDefaultAsync(u => u.Id == currentUserId))?.FullName;

    await _db.SaveChangesAsync();
    return Ok(new { message = "Section B reset to empty draft." });
}
```

- **[HttpPost("sectionB/send")] — Used by:** SectionB.cshtml (Send button)

- **Purpose:**
  Sends Section B to the requestor for acknowledgement. This locks the Section B draft and marks it as “sent,” but final completion still depends on both sides accepting later.

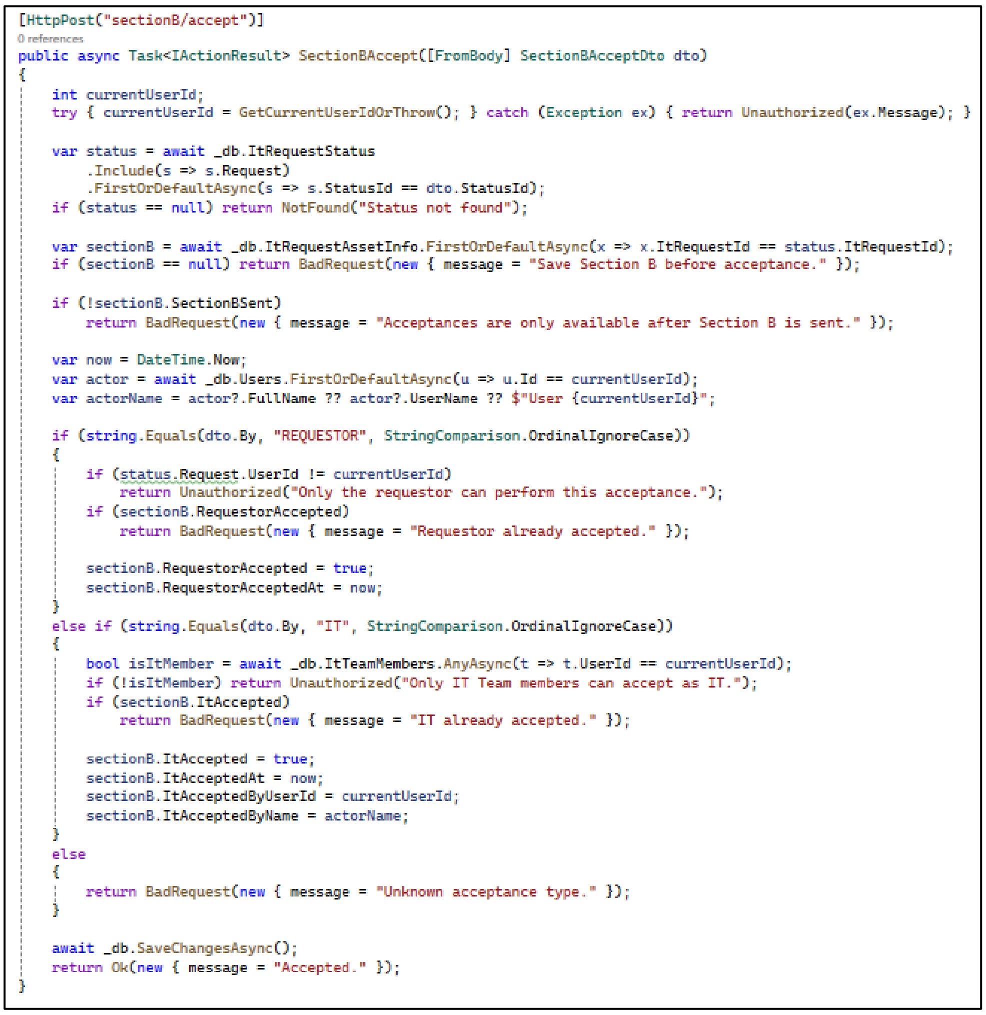
- **Key functionalities:**
  - Confirms who’s logged in and checks that they’re part of the IT team. Non-IT users can’t send Section B.
  - Loads the status by StatusId and makes sure the main request’s OverallStatus is **Approved**. Section B can only be sent after Section A has cleared all approvals.
  - Finds or creates the ItRequestAssetInfo row. If there isn’t a draft yet, it creates one on the fly so IT can send in a single step.
  - Applies the incoming fields from the payload (asset number, machine ID, IP, MACs, dial-up, remarks), trims them, and stamps editor name and time.
  - Validates that there is at least one identifier present: **Asset No**, **Machine ID**, or **IP Address**. If all three are missing, it stops and tells the user what to fix.
  - Refuses to resend if Section B is already marked as sent.
  - Marks SectionBSent = true and sets SectionBSentAt to now, effectively locking the Section B content for edits until the acceptance steps are completed.
  - Returns a short success message so the UI can update the banner/state immediately.



```
[HttpPost("sectionB/send")]
0 references
public async Task<IActionResult> SectionBSend([FromBody] SectionBSendDto dto)
{
    int currentUserId;
    try { currentUserId = GetCurrentUserIdOrThrow(); } catch (Exception ex) { return Unauthorized(ex.Message); }

    var status = await _db.ItRequestStatus
        .Include(s => s.Request)
        .FirstOrDefaultAsync(s => s.StatusId == dto.StatusId);

    if (status == null) return NotFound(new { message = "Status not found" });
    if ((status.OverallStatus ?? "Pending") != "Approved")
        return BadRequest(new { message = "Section B can be sent only after OverallStatus is Approved." });

    bool isItMember = await _db.ItTeamMembers.AnyAsync(t => t.UserId == currentUserId);
    if (!isItMember) return Unauthorized(new { message = "Only IT can send Section B." });

    var b = await _db.ItRequestAssetInfo.FirstOrDefaultAsync(x => x.ItRequestId == status.ItRequestId);
    if (b?.SectionBSent == true)
        return BadRequest(new { message = "Section B already sent." });

    var editor = await _db.Users.FirstOrDefaultAsync(u => u.Id == currentUserId);
    var editorName = editor?.FullName ?? editor?.UserName ?? $"User {currentUserId}";
    var now = DateTime.Now;

    if (b == null)
    {
        b = new ItRequestAssetInfo
        {
            ItRequestId = status.ItRequestId,
            RequestorAccepted = false,
            RequestorAcceptedAt = null,
            ItAccepted = false,
            ItAcceptedAt = null,
            ItAcceptedByUserId = null,
            ItAcceptedByName = null
        };
        _db.ItRequestAssetInfo.Add(b);
    }

    // Apply incoming fields (allow send without prior draft)
    if (dto.AssetNo != null) b.AssetNo = dto.AssetNo.Trim();
    if (dto.MachineId != null) b.MachineId = dto.MachineId.Trim();
    if (dto.IpAddress != null) b.IpAddress = dto.IpAddress.Trim();
    if (dto.WiredMac != null) b.WiredMac = dto.WiredMac.Trim();
    if (dto.WifiMac != null) b.WifiMac = dto.WifiMac.Trim();
    if (dto.DialupAcc != null) b.DialupAcc = dto.DialupAcc.Trim();
    if (dto.Remarks != null) b.Remarks = dto.Remarks.Trim();

    b.LastEditedAt = now;
    b.LastEditedByUserId = currentUserId;
    b.LastEditedByName = editorName;

    // Require at least one identifier
    if (string.IsNullOrWhiteSpace(b.AssetNo) &&
        string.IsNullOrWhiteSpace(b.MachineId) &&
        string.IsNullOrWhiteSpace(b.IpAddress))
        return BadRequest(new { message = "Provide at least Asset No / Machine ID / IP Address before sending." });

    b.SectionBSent = true;
    b.SectionBSentAt = now;

    await _db.SaveChangesAsync();
    return Ok(new { message = "Section B sent. It is now locked for editing until both acceptances complete." });
}
```

- **[HttpPost("sectionB/accept")]** (Used by: SectionB.cshtml, Accept buttons)
- **Purpose:**
  Records an acceptance for Section B, either by the requestor or by IT, after Section B has been sent.
- **Key functionalities:**
  - Verifies the logged-in user, loads the status by StatusId, and ensures a Section B row exists. If Section B hasn't been sent yet, it blocks acceptance.
  - Determines who is accepting based on dto.By:

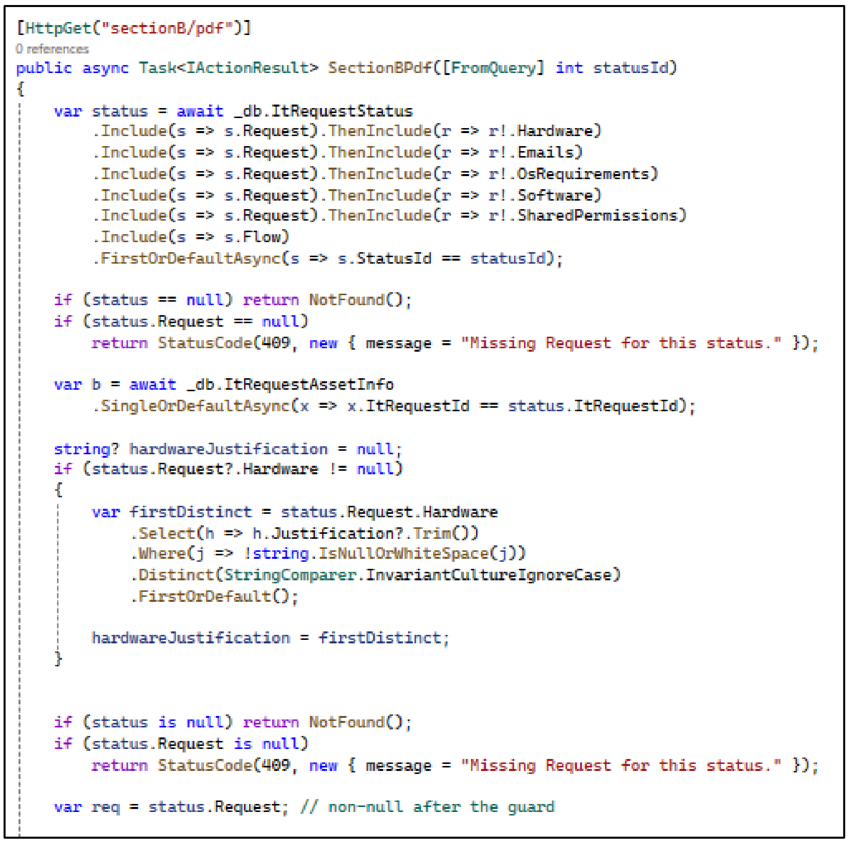
        - ▪ REQUESTOR: only the original requester can accept. If already accepted, returns a friendly “already accepted” message. Otherwise stamps RequestorAccepted = true and the time.
        - ▪ IT: only users in ItTeamMembers can accept as IT. If already accepted, it stops. Otherwise sets ItAccepted = true, stamps the time, and saves ItAcceptedByUserId/ItAcceptedByName.
    - Rejects any other dto.By value with “Unknown acceptance type.”
    - On success, saves and returns { message: "Accepted." } so the UI can refresh badges and buttons.



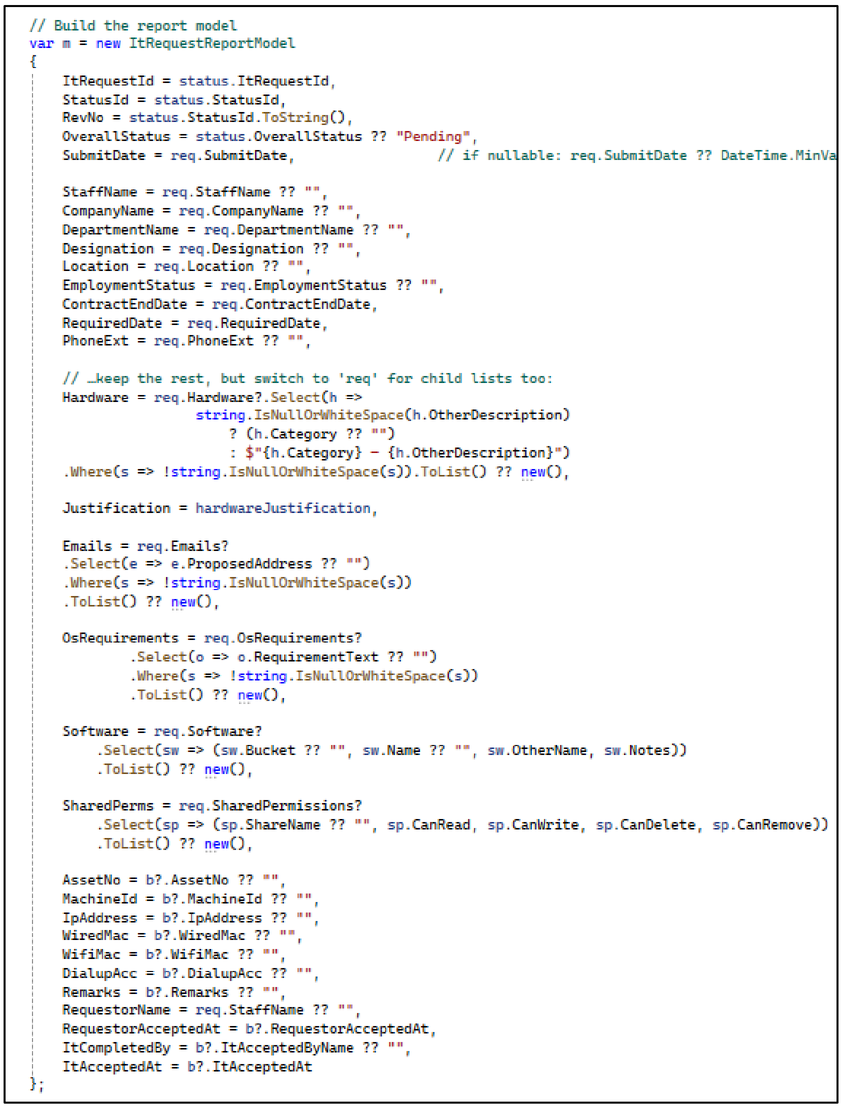
```
[HttpPost("sectionB/accept")]
0 references
public async Task<IActionResult> SectionBAccept([FromBody] SectionBAcceptDto dto)
{
    int currentUserId;
    try { currentUserId = GetCurrentUserIdOrThrow(); } catch (Exception ex) { return Unauthorized(ex.Message); }

    var status = await _db.ItRequestStatus
        .Include(s => s.Request)
        .FirstOrDefaultAsync(s => s.StatusId == dto.StatusId);
    if (status == null) return NotFound("Status not found");

    var sectionB = await _db.ItRequestAssetInfo.FirstOrDefaultAsync(x => x.ItRequestId == status.ItRequestId);
    if (sectionB == null) return BadRequest(new { message = "Save Section B before acceptance." });

    if (!sectionB.SectionBSent)
        return BadRequest(new { message = "Acceptances are only available after Section B is sent." });

    var now = DateTime.Now;
    var actor = await _db.Users.FirstOrDefaultAsync(u => u.Id == currentUserId);
    var actorName = actor?.FullName ?? actor?.UserName ?? $"User {currentUserId}";

    if (string.Equals(dto.By, "REQUESTOR", StringComparison.OrdinalIgnoreCase))
    {
        if (status.Request.UserId != currentUserId)
            return Unauthorized("Only the requestor can perform this acceptance.");
        if (sectionB.RequestorAccepted)
            return BadRequest(new { message = "Requestor already accepted." });

        sectionB.RequestorAccepted = true;
        sectionB.RequestorAcceptedAt = now;
    }
    else if (string.Equals(dto.By, "IT", StringComparison.OrdinalIgnoreCase))
    {
        bool isItMember = await _db.ItTeamMembers.AnyAsync(t => t.UserId == currentUserId);
        if (!isItMember) return Unauthorized("Only IT Team members can accept as IT.");
        if (sectionB.ItAccepted)
            return BadRequest(new { message = "IT already accepted." });

        sectionB.ItAccepted = true;
        sectionB.ItAcceptedAt = now;
        sectionB.ItAcceptedByUserId = currentUserId;
        sectionB.ItAcceptedByName = actorName;
    }
    else
    {
        return BadRequest(new { message = "Unknown acceptance type." });
    }

    await _db.SaveChangesAsync();
    return Ok(new { message = "Accepted." });
}
```

- **[HttpGet("sectionB/pdf")] — Used by:** SectionB.cshtml (Download/Print PDF button)

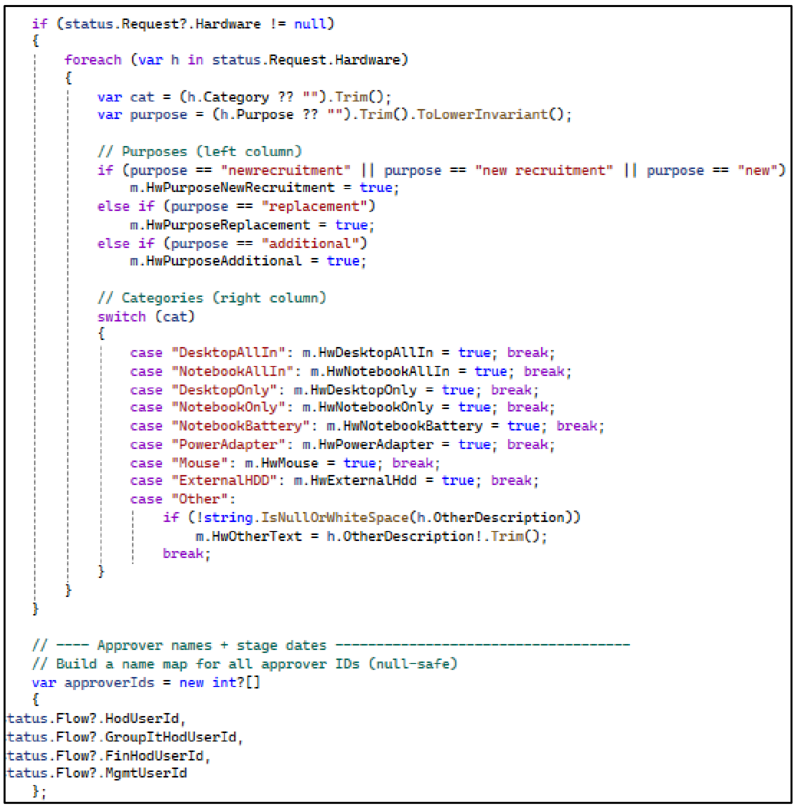
- **Purpose:**
  Produces the Section B PDF for a request that has fully cleared Section A approvals and has both acceptances in Section B. It pulls the requester snapshot, all Section A selections, Section B asset info, and approver names and dates, then renders a single PDF file.
- **Key functionalities:**
  - Loads the status by statusId together with the full Request graph (hardware, emails, OS requirements, software, shared permissions) and the approval flow. Returns 404 if the status does not exist and 409 if the status row has no linked request.
  - Looks up the Section B row (ItRequestAssetInfo).
  - Picks one hardware justification to print once in the PDF. It takes the first non-empty distinct justification from the Hardware list so the same sentence does not repeat across multiple hardware lines.
  - Builds the ItRequestReportModel:
    - Copies the core requester fields and Section A lists.
    - Flattens Hardware into printable strings and sets purpose/category flags (e.g., HwPurposeNewRecruitment, HwNotebookAllIn, HwOtherText) for the PDF checkboxes.
    - Adds Section B asset fields like Asset No, Machine ID, IP, MACs, Dial-up, Remarks, plus acceptance stamps and who accepted.
  - Resolves approver display names. Collects unique approver IDs from the flow, queries Users, and maps names into the model. Also passes the per-stage submit dates (HOD, Group IT, Finance, Management) for the signature/date area.
  - Enforces the final gate. If both RequestorAccepted and ItAccepted are not true, it returns 400 with "PDF available after both acceptances."
  - Calls ItRequestPdfService().Generate(model) and returns the PDF as application/pdf so the browser can display or download it depending on user settings.



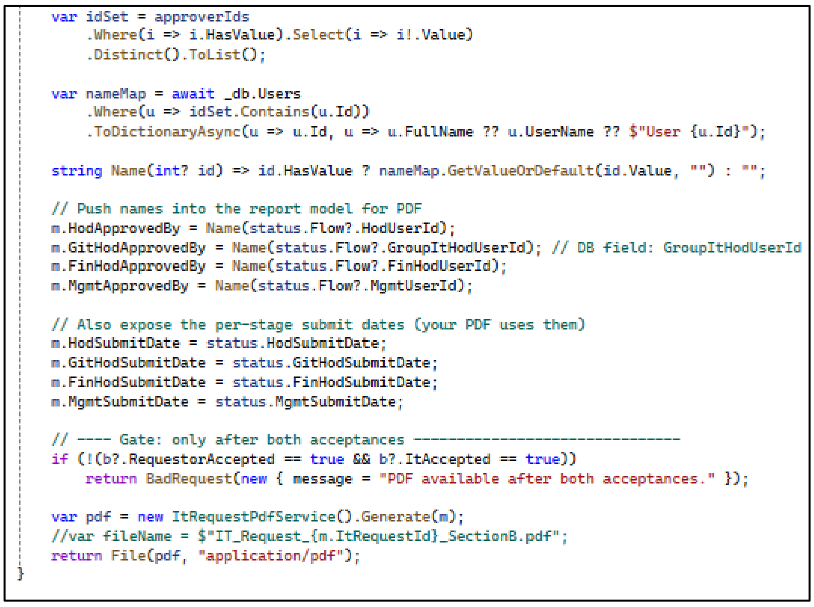
```
[HttpGet("sectionB/pdf")]
0 references
public async Task<IActionResult> SectionBPdf([FromQuery] int statusId)
{
    var status = await _db.ItRequestStatus
        .Include(s => s.Request).ThenInclude(r => r!.Hardware)
        .Include(s => s.Request).ThenInclude(r => r!.Emails)
        .Include(s => s.Request).ThenInclude(r => r!.OsRequirements)
        .Include(s => s.Request).ThenInclude(r => r!.Software)
        .Include(s => s.Request).ThenInclude(r => r!.SharedPermissions)
        .Include(s => s.Flow)
        .FirstOrDefaultAsync(s => s.StatusId == statusId);

    if (status == null) return NotFound();
    if (status.Request == null)
        return StatusCode(409, new { message = "Missing Request for this status." });

    var b = await _db.ItRequestAssetInfo
        .SingleOrDefaultAsync(x => x.ItRequestId == status.ItRequestId);

    string? hardwareJustification = null;
    if (status.Request?.Hardware != null)
    {
        var firstDistinct = status.Request.Hardware
            .Select(h => h.Justification?.Trim())
            .Where(j => !string.IsNullOrWhiteSpace(j))
            .Distinct(StringComparer.InvariantCultureIgnoreCase)
            .FirstOrDefault();

        hardwareJustification = firstDistinct;
    }


    if (status is null) return NotFound();
    if (status.Request is null)
        return StatusCode(409, new { message = "Missing Request for this status." });

    var req = status.Request; // non-null after the guard
```



```
// Build the report model
var m = new ItRequestReportModel
{
    ItRequestId = status.ItRequestId,
    StatusId = status.StatusId,
    RevNo = status.StatusId.ToString(),
    OverallStatus = status.OverallStatus ?? "Pending",
    SubmitDate = req.SubmitDate,                 // if nullable: req.SubmitDate ?? DateTime.MinVa

    StaffName = req.StaffName ?? "",
    CompanyName = req.CompanyName ?? "",
    DepartmentName = req.DepartmentName ?? "",
    Designation = req.Designation ?? "",
    Location = req.Location ?? "",
    EmploymentStatus = req.EmploymentStatus ?? "",
    ContractEndDate = req.ContractEndDate,
    RequiredDate = req.RequiredDate,
    PhoneExt = req.PhoneExt ?? "",

    // …keep the rest, but switch to 'req' for child lists too:
    Hardware = req.Hardware?.Select(h =>
                    string.IsNullOrWhiteSpace(h.OtherDescription)
                        ? (h.Category ?? "")
                        : $"{h.Category} – {h.OtherDescription}")
    .Where(s => !string.IsNullOrWhiteSpace(s)).ToList() ?? new(),

    Justification = hardwareJustification,

    Emails = req.Emails?
    .Select(e => e.ProposedAddress ?? "")
    .Where(s => !string.IsNullOrWhiteSpace(s))
    .ToList() ?? new(),

    OsRequirements = req.OsRequirements?
            .Select(o => o.RequirementText ?? "")
            .Where(s => !string.IsNullOrWhiteSpace(s))
            .ToList() ?? new(),

    Software = req.Software?
        .Select(sw => (sw.Bucket ?? "", sw.Name ?? "", sw.OtherName, sw.Notes))
        .ToList() ?? new(),

    SharedPerms = req.SharedPermissions?
        .Select(sp => (sp.ShareName ?? "", sp.CanRead, sp.CanWrite, sp.CanDelete, sp.CanRemove))
        .ToList() ?? new(),

    AssetNo = b?.AssetNo ?? "",
    MachineId = b?.MachineId ?? "",
    IpAddress = b?.IpAddress ?? "",
    WiredMac = b?.WiredMac ?? "",
    WifiMac = b?.WifiMac ?? "",
    DialupAcc = b?.DialupAcc ?? "",
    Remarks = b?.Remarks ?? "",
    RequestorName = req.StaffName ?? "",
    RequestorAcceptedAt = b?.RequestorAcceptedAt,
    ItCompletedBy = b?.ItAcceptedByName ?? "",
    ItAcceptedAt = b?.ItAcceptedAt
};
```

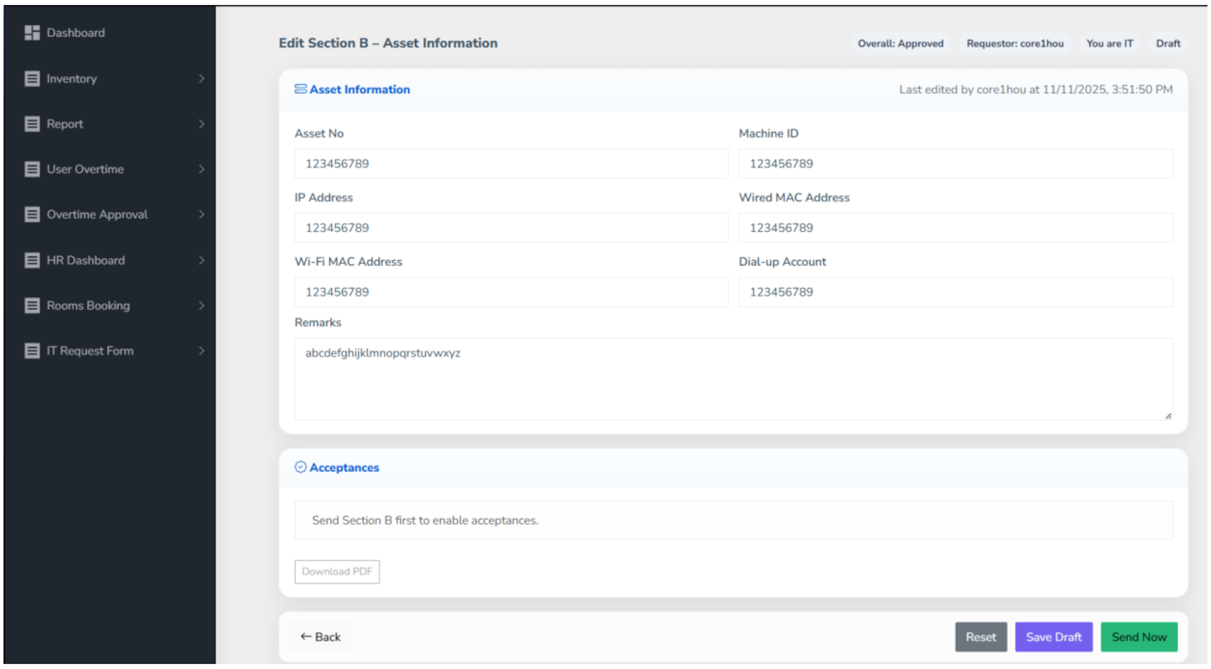


```
if (status.Request?.Hardware != null)
{
    foreach (var h in status.Request.Hardware)
    {
        var cat = (h.Category ?? "").Trim();
        var purpose = (h.Purpose ?? "").Trim().ToLowerInvariant();

        // Purposes (left column)
        if (purpose == "newrecruitment" || purpose == "new recruitment" || purpose == "new")
            m.HwPurposeNewRecruitment = true;
        else if (purpose == "replacement")
            m.HwPurposeReplacement = true;
        else if (purpose == "additional")
            m.HwPurposeAdditional = true;

        // Categories (right column)
        switch (cat)
        {
            case "DesktopAllIn": m.HwDesktopAllIn = true; break;
            case "NotebookAllIn": m.HwNotebookAllIn = true; break;
            case "DesktopOnly": m.HwDesktopOnly = true; break;
            case "NotebookOnly": m.HwNotebookOnly = true; break;
            case "NotebookBattery": m.HwNotebookBattery = true; break;
            case "PowerAdapter": m.HwPowerAdapter = true; break;
            case "Mouse": m.HwMouse = true; break;
            case "ExternalHDD": m.HwExternalHdd = true; break;
            case "Other":
                if (!string.IsNullOrWhiteSpace(h.OtherDescription))
                    m.HwOtherText = h.OtherDescription!.Trim();
                break;
        }
    }
}

// ---- Approver names + stage dates -------------------------------------
// Build a name map for all approver IDs (null-safe)
var approverIds = new int?[]
{
tatus.Flow?.HodUserId,
tatus.Flow?.GroupItHodUserId,
tatus.Flow?.FinHodUserId,
tatus.Flow?.MgmtUserId
};
```



```
    var idSet = approverIds
        .Where(i => i.HasValue).Select(i => i!.Value)
        .Distinct().ToList();

    var nameMap = await _db.Users
        .Where(u => idSet.Contains(u.Id))
        .ToDictionaryAsync(u => u.Id, u => u.FullName ?? u.UserName ?? $"User {u.Id}");

    string Name(int? id) => id.HasValue ? nameMap.GetValueOrDefault(id.Value, "") : "";

    // Push names into the report model for PDF
    m.HodApprovedBy = Name(status.Flow?.HodUserId);
    m.GitHodApprovedBy = Name(status.Flow?.GroupItHodUserId); // DB field: GroupItHodUserId
    m.FinHodApprovedBy = Name(status.Flow?.FinHodUserId);
    m.MgmtApprovedBy = Name(status.Flow?.MgmtUserId);

    // Also expose the per-stage submit dates (your PDF uses them)
    m.HodSubmitDate = status.HodSubmitDate;
    m.GitHodSubmitDate = status.GitHodSubmitDate;
    m.FinHodSubmitDate = status.FinHodSubmitDate;
    m.MgmtSubmitDate = status.MgmtSubmitDate;

    // ---- Gate: only after both acceptances --------------------------------
    if (!(b?.RequestorAccepted == true && b?.ItAccepted == true))
        return BadRequest(new { message = "PDF available after both acceptances." });

    var pdf = new ItRequestPdfService().Generate(m);
    //var fileName = $"IT_Request_{m.ItRequestId}_SectionB.pdf";
    return File(pdf, "application/pdf");
}
```

- SectionBEdit.cshtml



Edit Section B – Asset Information

Overall: Approved | Requestor: core1hou | You are IT | Draft

Asset Information — Last edited by core1hou at 11/11/2025, 3:51:50 PM

Asset No: 123456789
Machine ID: 123456789
IP Address: 123456789
Wired MAC Address: 123456789
Wi-Fi MAC Address: 123456789
Dial-up Account: 123456789
Remarks: abcdefghijklmnopqrstuvwxyz

Acceptances

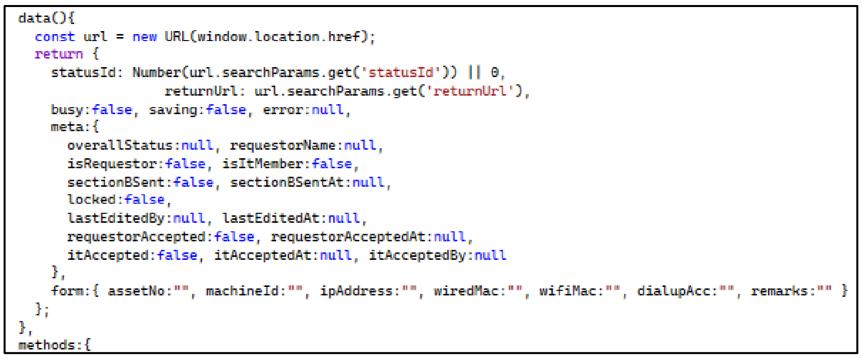
Send Section B first to enable acceptances.

This file (SectionBEdit.cshtml) is the IT-focused editor for Section B. It loads the current draft, lets IT update asset details, and provides shortcuts to send, accept, or open the requester-facing page. It respects the same lock rules as Section B and only unlocks when the overall request is Approved.

- ✓ **<div id="sbEditApp">…</div>**
  - **Purpose:** Root container for the Vue app so the cards, inputs, and buttons are reactive.
- ✓ **const sbEditApp = Vue.createApp({...})**
  - **Purpose:** Creates the Vue instance and wires up meta flags, form state, API calls, and lifecycle.
- ✓ **data()**
  - **statusId, returnUrl**: Parsed from the URL at startup. statusId identifies the request being edited. returnUrl is used by the Back button to return users to their previous view when it is safe to do so.
  - **busy, saving, error**: Control UX while loading and saving. busy disables actions during network calls, saving is dedicated to the Save Draft button, and error shows a clear alert when something fails.
  - **meta { overallStatus, requestorName, isRequestor, isItMember, sectionBSent, locked, lastEditedBy, lastEditedAt, requestorAccepted,**

requestorAcceptedAt, itAccepted, itAcceptedAt, itAcceptedBy }: A compact state object that drives the page. Editing unlocks only when overallStatus is Approved and the viewer is IT. sectionBSent and locked flip the form to read-only for the acceptance phase. The acceptance flags power the summary badges and PDF availability.

- **form { assetNo, machineId, ipAddress, wiredMac, wifiMac, dialupAcc, remarks** }: The asset fields IT maintains. These are disabled for non-IT users and when locked is true after sending.



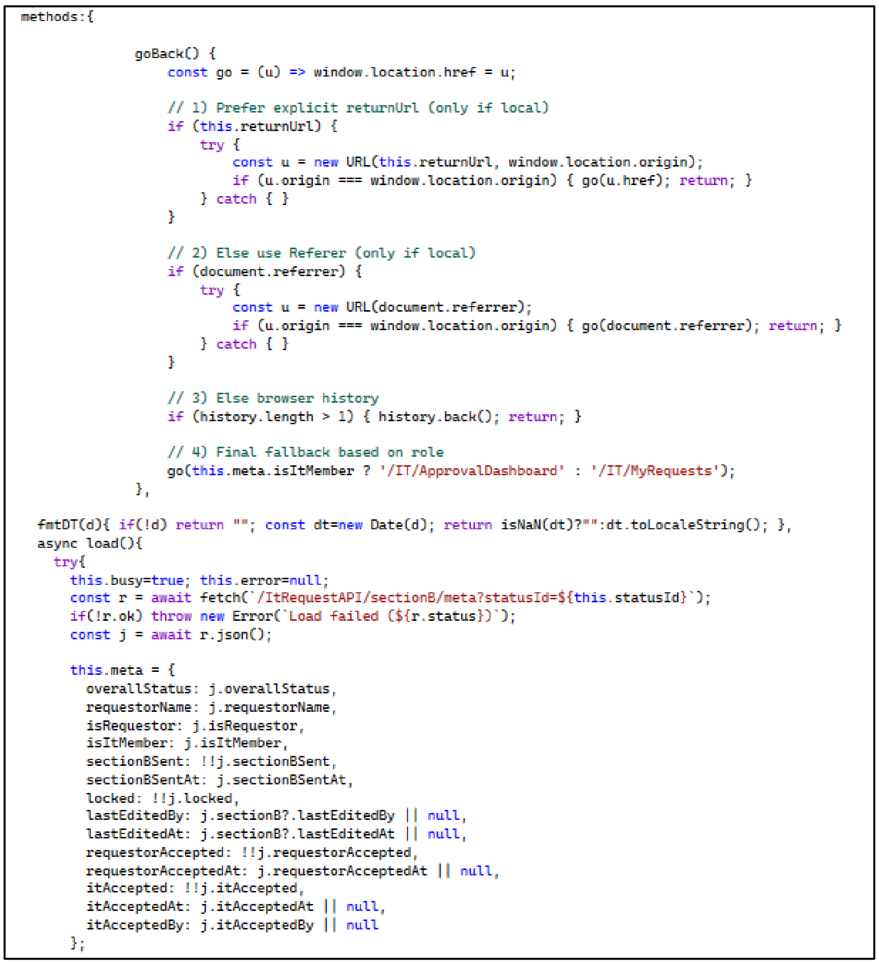
```
data(){
  const url = new URL(window.location.href);
  return {
    statusId: Number(url.searchParams.get('statusId')) || 0,
                returnUrl: url.searchParams.get('returnUrl'),
    busy:false, saving:false, error:null,
    meta:{
      overallStatus:null, requestorName:null,
      isRequestor:false, isItMember:false,
      sectionBSent:false, sectionBSentAt:null,
      locked:false,
      lastEditedBy:null, lastEditedAt:null,
      requestorAccepted:false, requestorAcceptedAt:null,
      itAccepted:false, itAcceptedAt:null, itAcceptedBy:null
    },
    form:{ assetNo:"", machineId:"", ipAddress:"", wiredMac:"", wifiMac:"", dialupAcc:"", remarks:"" }
  };
},
methods:{
```

✓ **methods**

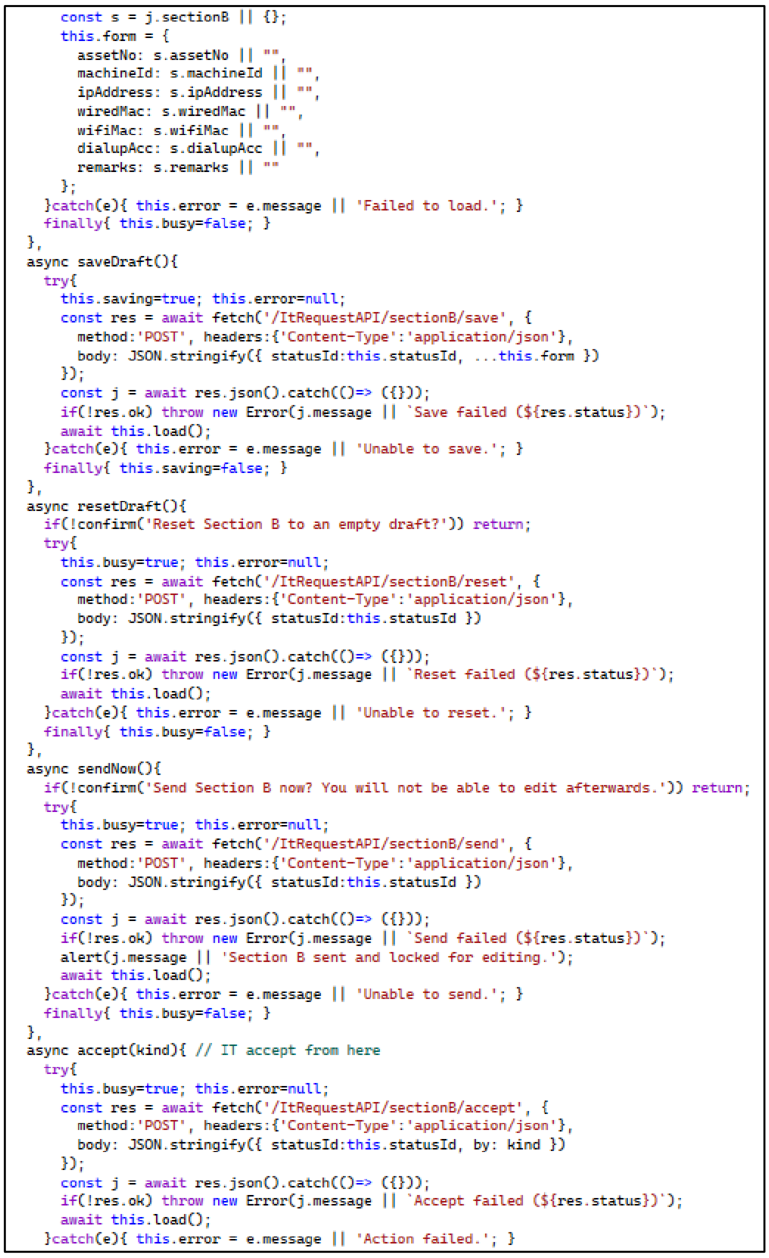
- **goBack()**: Navigates safely and predictably. It prefers a same-origin returnUrl, then a same-origin referrer, then browser history, and finally a role-based fallback route. This avoids accidental navigation to other sites.
- **fmtDT(d)**: Formats timestamps using the browser locale and returns an empty string when parsing fails. Keeps "Last edited" and acceptance dates readable.
- **load()**: Fetches /ItRequestAPI/sectionB/meta?statusId=..., hydrates meta, and fills the form with any existing draft values. Normalizes booleans and dates so the UI reflects a consistent lock and acceptance state.
- **saveDraft()**: Posts the current form to /ItRequestAPI/sectionB/save. On success it reloads the meta so "Last edited by" immediately reflects the latest change.
- **resetDraft()**: Confirms, then posts to /ItRequestAPI/sectionB/reset to clear the draft back to empty fields. Reloads afterward to keep the UI in sync.
- **sendNow()**: Confirms intent and posts to /ItRequestAPI/sectionB/send. A successful send locks the editor, switches the banner to Sent, and enables the

acceptance section. This mirrors the business rule that Section B is immutable after sending.

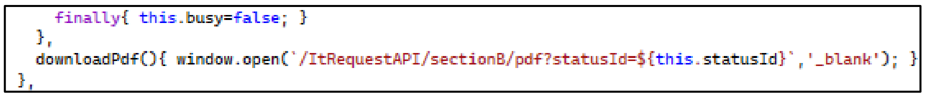
- ○ **accept(kind)**: Lets IT perform their acceptance straight from here with /ItRequestAPI/sectionB/accept and by: 'IT'. It also supports a generic kind so the same code path can be reused if needed.
- ○ **downloadPdf()**: Opens the Section B PDF in a new tab. The button activates only when both acceptances are complete.



```
methods:{

              goBack() {
                  const go = (u) => window.location.href = u;

                  // 1) Prefer explicit returnUrl (only if local)
                  if (this.returnUrl) {
                      try {
                          const u = new URL(this.returnUrl, window.location.origin);
                          if (u.origin === window.location.origin) { go(u.href); return; }
                      } catch { }
                  }

                  // 2) Else use Referer (only if local)
                  if (document.referrer) {
                      try {
                          const u = new URL(document.referrer);
                          if (u.origin === window.location.origin) { go(document.referrer); return; }
                      } catch { }
                  }

                  // 3) Else browser history
                  if (history.length > 1) { history.back(); return; }

                  // 4) Final fallback based on role
                  go(this.meta.isItMember ? '/IT/ApprovalDashboard' : '/IT/MyRequests');
              },

  fmtDT(d){ if(!d) return ""; const dt=new Date(d); return isNaN(dt)?"":dt.toLocaleString(); },
  async load(){
    try{
      this.busy=true; this.error=null;
      const r = await fetch(`/ItRequestAPI/sectionB/meta?statusId=${this.statusId}`);
      if(!r.ok) throw new Error(`Load failed (${r.status})`);
      const j = await r.json();

      this.meta = {
        overallStatus: j.overallStatus,
        requestorName: j.requestorName,
        isRequestor: j.isRequestor,
        isItMember: j.isItMember,
        sectionBSent: !!j.sectionBSent,
        sectionBSentAt: j.sectionBSentAt,
        locked: !!j.locked,
        lastEditedBy: j.sectionB?.lastEditedBy || null,
        lastEditedAt: j.sectionB?.lastEditedAt || null,
        requestorAccepted: !!j.requestorAccepted,
        requestorAcceptedAt: j.requestorAcceptedAt || null,
        itAccepted: !!j.itAccepted,
        itAcceptedAt: j.itAcceptedAt || null,
        itAcceptedBy: j.itAcceptedBy || null
      };
```



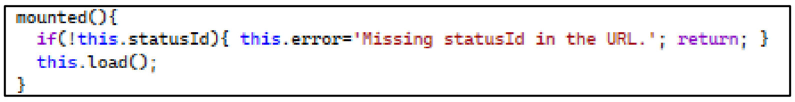
```
      const s = j.sectionB || {};
      this.form = {
        assetNo: s.assetNo || "",
        machineId: s.machineId || "",
        ipAddress: s.ipAddress || "",
        wiredMac: s.wiredMac || "",
        wifiMac: s.wifiMac || "",
        dialupAcc: s.dialupAcc || "",
        remarks: s.remarks || ""
      };
    }catch(e){ this.error = e.message || 'Failed to load.'; }
    finally{ this.busy=false; }
  },
  async saveDraft(){
    try{
      this.saving=true; this.error=null;
      const res = await fetch('/ItRequestAPI/sectionB/save', {
        method:'POST', headers:{'Content-Type':'application/json'},
        body: JSON.stringify({ statusId:this.statusId, ...this.form })
      });
      const j = await res.json().catch(()=> ({}));
      if(!res.ok) throw new Error(j.message || `Save failed (${res.status})`);
      await this.load();
    }catch(e){ this.error = e.message || 'Unable to save.'; }
    finally{ this.saving=false; }
  },
  async resetDraft(){
    if(!confirm('Reset Section B to an empty draft?')) return;
    try{
      this.busy=true; this.error=null;
      const res = await fetch('/ItRequestAPI/sectionB/reset', {
        method:'POST', headers:{'Content-Type':'application/json'},
        body: JSON.stringify({ statusId:this.statusId })
      });
      const j = await res.json().catch(()=> ({}));
      if(!res.ok) throw new Error(j.message || `Reset failed (${res.status})`);
      await this.load();
    }catch(e){ this.error = e.message || 'Unable to reset.'; }
    finally{ this.busy=false; }
  },
  async sendNow(){
    if(!confirm('Send Section B now? You will not be able to edit afterwards.')) return;
    try{
      this.busy=true; this.error=null;
      const res = await fetch('/ItRequestAPI/sectionB/send', {
        method:'POST', headers:{'Content-Type':'application/json'},
        body: JSON.stringify({ statusId:this.statusId })
      });
      const j = await res.json().catch(()=> ({}));
      if(!res.ok) throw new Error(j.message || `Send failed (${res.status})`);
      alert(j.message || 'Section B sent and locked for editing.');
      await this.load();
    }catch(e){ this.error = e.message || 'Unable to send.'; }
    finally{ this.busy=false; }
  },
  async accept(kind){ // IT accept from here
    try{
      this.busy=true; this.error=null;
      const res = await fetch('/ItRequestAPI/sectionB/accept', {
        method:'POST', headers:{'Content-Type':'application/json'},
        body: JSON.stringify({ statusId:this.statusId, by: kind })
      });
      const j = await res.json().catch(()=> ({}));
      if(!res.ok) throw new Error(j.message || `Accept failed (${res.status})`);
      await this.load();
    }catch(e){ this.error = e.message || 'Action failed.'; }
```



```
      finally{ this.busy=false; }
    },
    downloadPdf(){ window.open(`/ItRequestAPI/sectionB/pdf?statusId=${this.statusId}`,'_blank'); }
  },
```

✓ mounted

- **mounted():** Verifies statusId exists, then calls load() immediately. Until the overall status is Approved, the page shows a gate message and keeps the editor locked. Once Approved, IT can edit, save, and send, followed by acceptances.

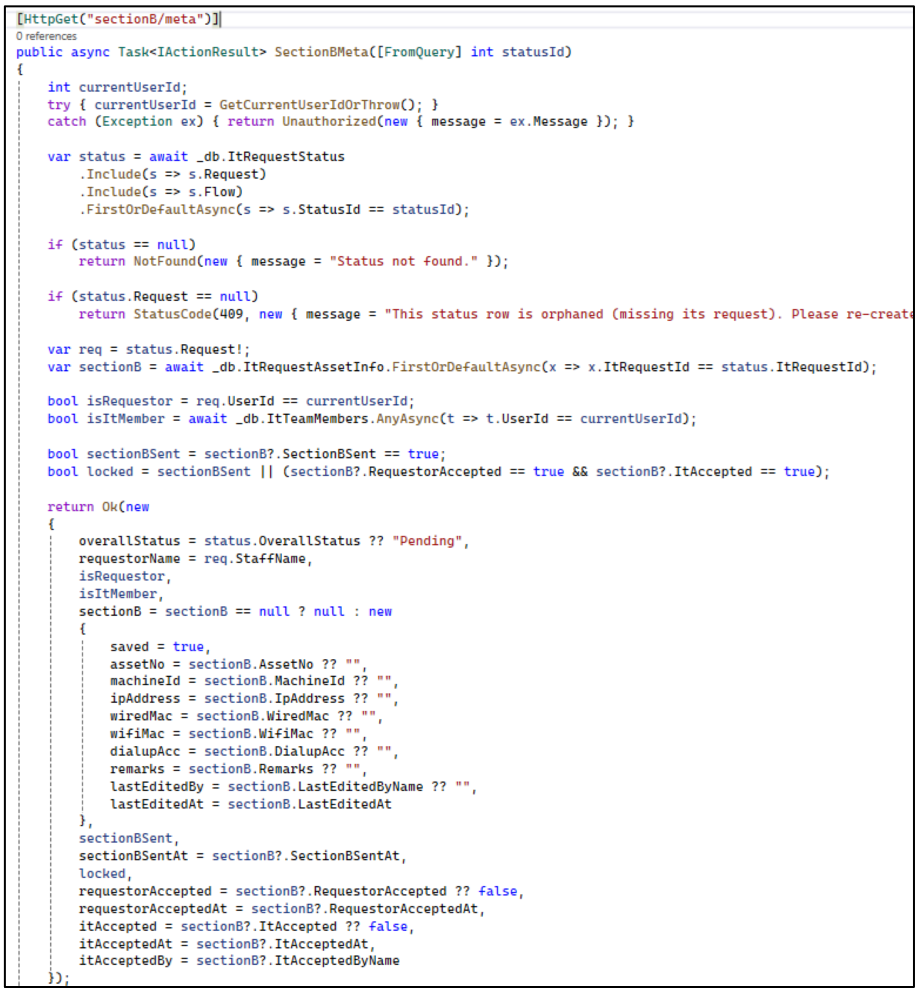


```
mounted(){
  if(!this.statusId){ this.error='Missing statusId in the URL.'; return; }
  this.load();
}
```

- ITRequestAPI (SectionBEdit.cshtml)

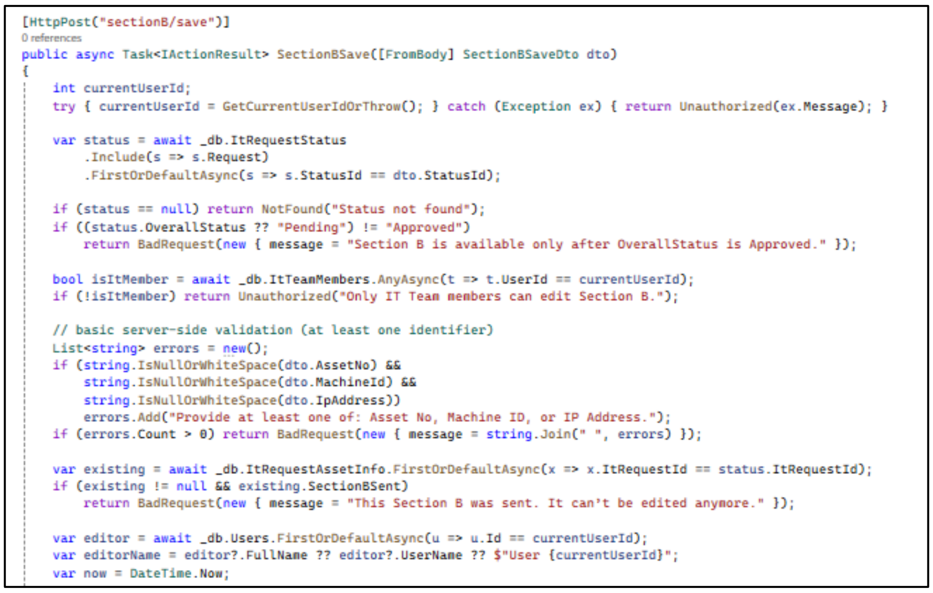
- **[HttpGet("sectionB/meta")]** (Used by: SectionBEdit.cshtml)
- **Purpose:**

  Load the header and permission info for Section B so the edit screen knows who the viewer is, whether Section B exists, if it was sent, and if it's locked.
- **Key functionalities:**
  - Authenticates the user, fetches the status with its Request and Flow.
  - Finds the Section B row (may be null if nobody started it).
  - Sets flags you'll bind to the UI: isRequestor, isItMember, sectionBSent, locked.
  - If Section B exists, returns the saved fields to prefill the edit form plus "last edited" metadata.

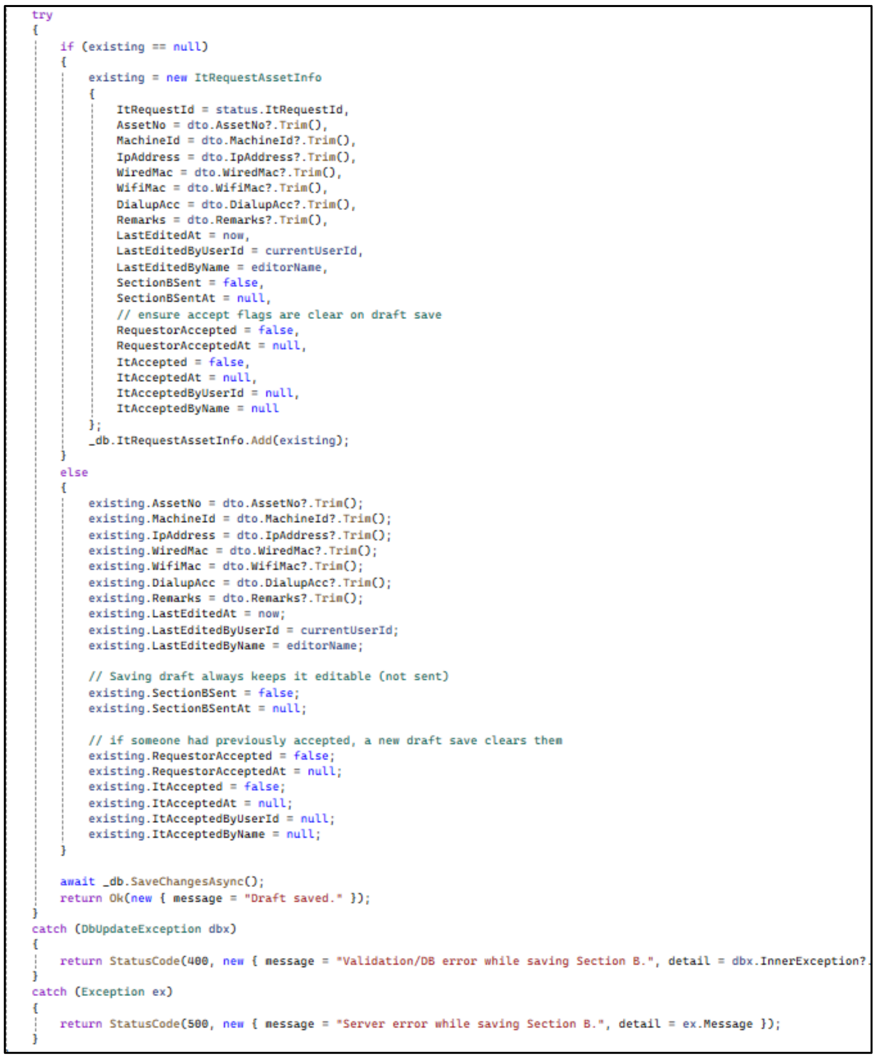


```
[HttpGet("sectionB/meta")]
0 references
public async Task<IActionResult> SectionBMeta([FromQuery] int statusId)
{
    int currentUserId;
    try { currentUserId = GetCurrentUserIdOrThrow(); }
    catch (Exception ex) { return Unauthorized(new { message = ex.Message }); }

    var status = await _db.ItRequestStatus
        .Include(s => s.Request)
        .Include(s => s.Flow)
        .FirstOrDefaultAsync(s => s.StatusId == statusId);

    if (status == null)
        return NotFound(new { message = "Status not found." });

    if (status.Request == null)
        return StatusCode(409, new { message = "This status row is orphaned (missing its request). Please re-create

    var req = status.Request!;
    var sectionB = await _db.ItRequestAssetInfo.FirstOrDefaultAsync(x => x.ItRequestId == status.ItRequestId);

    bool isRequestor = req.UserId == currentUserId;
    bool isItMember = await _db.ItTeamMembers.AnyAsync(t => t.UserId == currentUserId);

    bool sectionBSent = sectionB?.SectionBSent == true;
    bool locked = sectionBSent || (sectionB?.RequestorAccepted == true && sectionB?.ItAccepted == true);

    return Ok(new
    {
        overallStatus = status.OverallStatus ?? "Pending",
        requestorName = req.StaffName,
        isRequestor,
        isItMember,
        sectionB = sectionB == null ? null : new
        {
            saved = true,
            assetNo = sectionB.AssetNo ?? "",
            machineId = sectionB.MachineId ?? "",
            ipAddress = sectionB.IpAddress ?? "",
            wiredMac = sectionB.WiredMac ?? "",
            wifiMac = sectionB.WifiMac ?? "",
            dialupAcc = sectionB.DialupAcc ?? "",
            remarks = sectionB.Remarks ?? "",
            lastEditedBy = sectionB.LastEditedByName ?? "",
            lastEditedAt = sectionB.LastEditedAt
        },
        sectionBSent,
        sectionBSentAt = sectionB?.SectionBSentAt,
        locked,
        requestorAccepted = sectionB?.RequestorAccepted ?? false,
        requestorAcceptedAt = sectionB?.RequestorAcceptedAt,
        itAccepted = sectionB?.ItAccepted ?? false,
        itAcceptedAt = sectionB?.ItAcceptedAt,
        itAcceptedBy = sectionB?.ItAcceptedByName
    });
```

- **[HttpPost("sectionB/save")]** (Used by: SectionBEdit.cshtml, Save Draft)
- **Purpose:**
  Let IT save or update Section B as a draft after Section A is fully approved. Draft saves keep the form editable and clear any prior acceptances so both sides will recheck later.
- **Key functionalities:**
  - Ensures the caller is an IT team member and the request's OverallStatus is Approved.
  - Validates that at least one identifier is present (Asset No or Machine ID or IP Address).
  - Creates the Section B row if missing, or updates it if present.
  - Always keeps it in draft state: sets SectionBSent = false and resets RequestorAccepted and ItAccepted fields.
  - Returns a simple "Draft saved." for the toast.

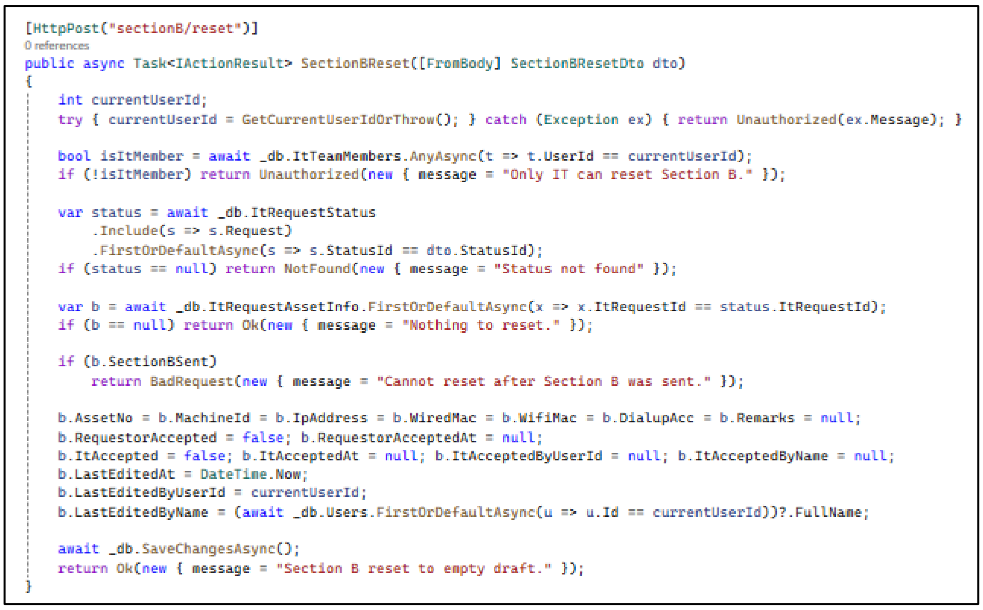


```
[HttpPost("sectionB/save")]
0 references
public async Task<IActionResult> SectionBSave([FromBody] SectionBSaveDto dto)
{
    int currentUserId;
    try { currentUserId = GetCurrentUserIdOrThrow(); } catch (Exception ex) { return Unauthorized(ex.Message); }

    var status = await _db.ItRequestStatus
        .Include(s => s.Request)
        .FirstOrDefaultAsync(s => s.StatusId == dto.StatusId);

    if (status == null) return NotFound("Status not found");
    if ((status.OverallStatus ?? "Pending") != "Approved")
        return BadRequest(new { message = "Section B is available only after OverallStatus is Approved." });

    bool isItMember = await _db.ItTeamMembers.AnyAsync(t => t.UserId == currentUserId);
    if (!isItMember) return Unauthorized("Only IT Team members can edit Section B.");

    // basic server-side validation (at least one identifier)
    List<string> errors = new();
    if (string.IsNullOrWhiteSpace(dto.AssetNo) &&
        string.IsNullOrWhiteSpace(dto.MachineId) &&
        string.IsNullOrWhiteSpace(dto.IpAddress))
        errors.Add("Provide at least one of: Asset No, Machine ID, or IP Address.");
    if (errors.Count > 0) return BadRequest(new { message = string.Join(" ", errors) });

    var existing = await _db.ItRequestAssetInfo.FirstOrDefaultAsync(x => x.ItRequestId == status.ItRequestId);
    if (existing != null && existing.SectionBSent)
        return BadRequest(new { message = "This Section B was sent. It can't be edited anymore." });

    var editor = await _db.Users.FirstOrDefaultAsync(u => u.Id == currentUserId);
    var editorName = editor?.FullName ?? editor?.UserName ?? $"User {currentUserId}";
    var now = DateTime.Now;
```



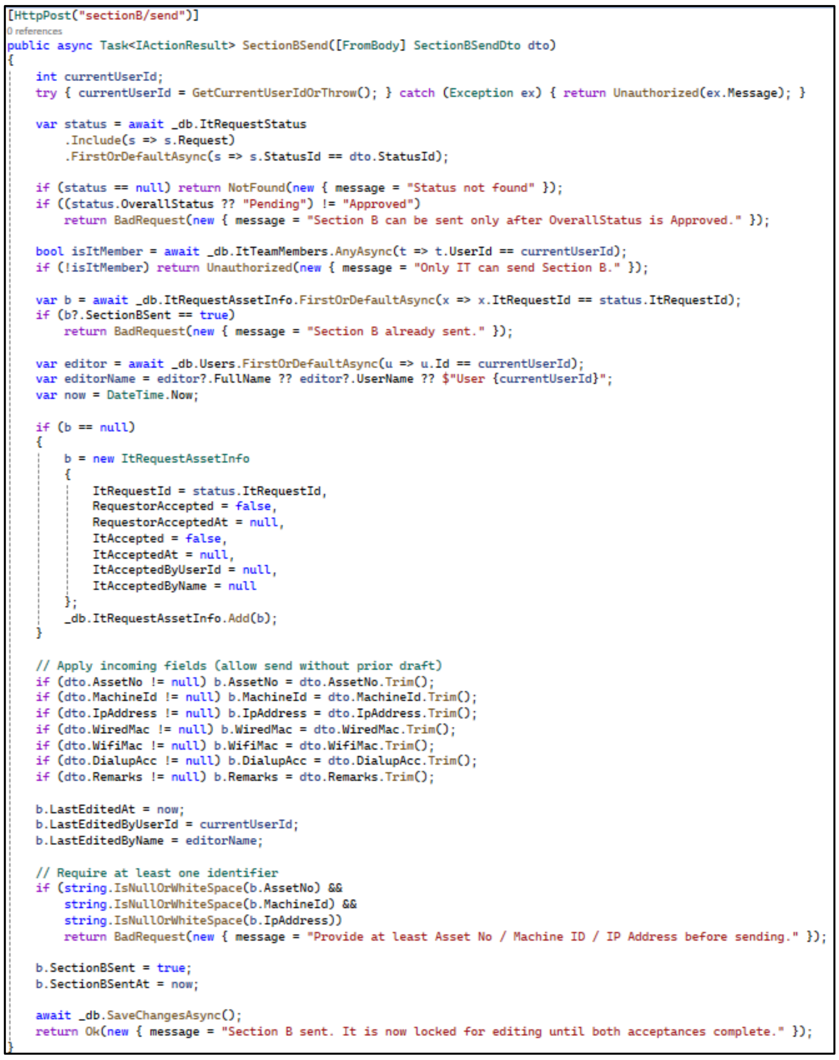
```
try
{
    if (existing == null)
    {
        existing = new ItRequestAssetInfo
        {
            ItRequestId = status.ItRequestId,
            AssetNo = dto.AssetNo?.Trim(),
            MachineId = dto.MachineId?.Trim(),
            IpAddress = dto.IpAddress?.Trim(),
            WiredMac = dto.WiredMac?.Trim(),
            WifiMac = dto.WifiMac?.Trim(),
            DialupAcc = dto.DialupAcc?.Trim(),
            Remarks = dto.Remarks?.Trim(),
            LastEditedAt = now,
            LastEditedByUserId = currentUserId,
            LastEditedByName = editorName,
            SectionBSent = false,
            SectionBSentAt = null,
            // ensure accept flags are clear on draft save
            RequestorAccepted = false,
            RequestorAcceptedAt = null,
            ItAccepted = false,
            ItAcceptedAt = null,
            ItAcceptedByUserId = null,
            ItAcceptedByName = null
        };
        _db.ItRequestAssetInfo.Add(existing);
    }
    else
    {
        existing.AssetNo = dto.AssetNo?.Trim();
        existing.MachineId = dto.MachineId?.Trim();
        existing.IpAddress = dto.IpAddress?.Trim();
        existing.WiredMac = dto.WiredMac?.Trim();
        existing.WifiMac = dto.WifiMac?.Trim();
        existing.DialupAcc = dto.DialupAcc?.Trim();
        existing.Remarks = dto.Remarks?.Trim();
        existing.LastEditedAt = now;
        existing.LastEditedByUserId = currentUserId;
        existing.LastEditedByName = editorName;

        // Saving draft always keeps it editable (not sent)
        existing.SectionBSent = false;
        existing.SectionBSentAt = null;

        // if someone had previously accepted, a new draft save clears them
        existing.RequestorAccepted = false;
        existing.RequestorAcceptedAt = null;
        existing.ItAccepted = false;
        existing.ItAcceptedAt = null;
        existing.ItAcceptedByUserId = null;
        existing.ItAcceptedByName = null;
    }

    await _db.SaveChangesAsync();
    return Ok(new { message = "Draft saved." });
}
catch (DbUpdateException dbx)
{
    return StatusCode(400, new { message = "Validation/DB error while saving Section B.", detail = dbx.InnerException?.
}
catch (Exception ex)
{
    return StatusCode(500, new { message = "Server error while saving Section B.", detail = ex.Message });
}
```

- **[HttpPost("sectionB/reset")]** (Used by: SectionBEdit.cshtml, Reset button)
- **Purpose:**
  Clear the current Section B draft back to empty. IT-only, and only if it hasn't been sent yet.
- **Key functionalities:**
  - Verifies the user is in IT.
  - If no Section B row exists, returns "Nothing to reset."
  - If Section B was already sent, blocks the reset.
  - Wipes all editable fields and clears both acceptance flags, then updates "last edited by" info.



```
[HttpPost("sectionB/reset")]
0 references
public async Task<IActionResult> SectionBReset([FromBody] SectionBResetDto dto)
{
    int currentUserId;
    try { currentUserId = GetCurrentUserIdOrThrow(); } catch (Exception ex) { return Unauthorized(ex.Message); }

    bool isItMember = await _db.ItTeamMembers.AnyAsync(t => t.UserId == currentUserId);
    if (!isItMember) return Unauthorized(new { message = "Only IT can reset Section B." });

    var status = await _db.ItRequestStatus
        .Include(s => s.Request)
        .FirstOrDefaultAsync(s => s.StatusId == dto.StatusId);
    if (status == null) return NotFound(new { message = "Status not found" });

    var b = await _db.ItRequestAssetInfo.FirstOrDefaultAsync(x => x.ItRequestId == status.ItRequestId);
    if (b == null) return Ok(new { message = "Nothing to reset." });

    if (b.SectionBSent)
        return BadRequest(new { message = "Cannot reset after Section B was sent." });

    b.AssetNo = b.MachineId = b.IpAddress = b.WiredMac = b.WifiMac = b.DialupAcc = b.Remarks = null;
    b.RequestorAccepted = false; b.RequestorAcceptedAt = null;
    b.ItAccepted = false; b.ItAcceptedAt = null; b.ItAcceptedByUserId = null; b.ItAcceptedByName = null;
    b.LastEditedAt = DateTime.Now;
    b.LastEditedByUserId = currentUserId;
    b.LastEditedByName = (await _db.Users.FirstOrDefaultAsync(u => u.Id == currentUserId))?.FullName;

    await _db.SaveChangesAsync();
    return Ok(new { message = "Section B reset to empty draft." });
}
```

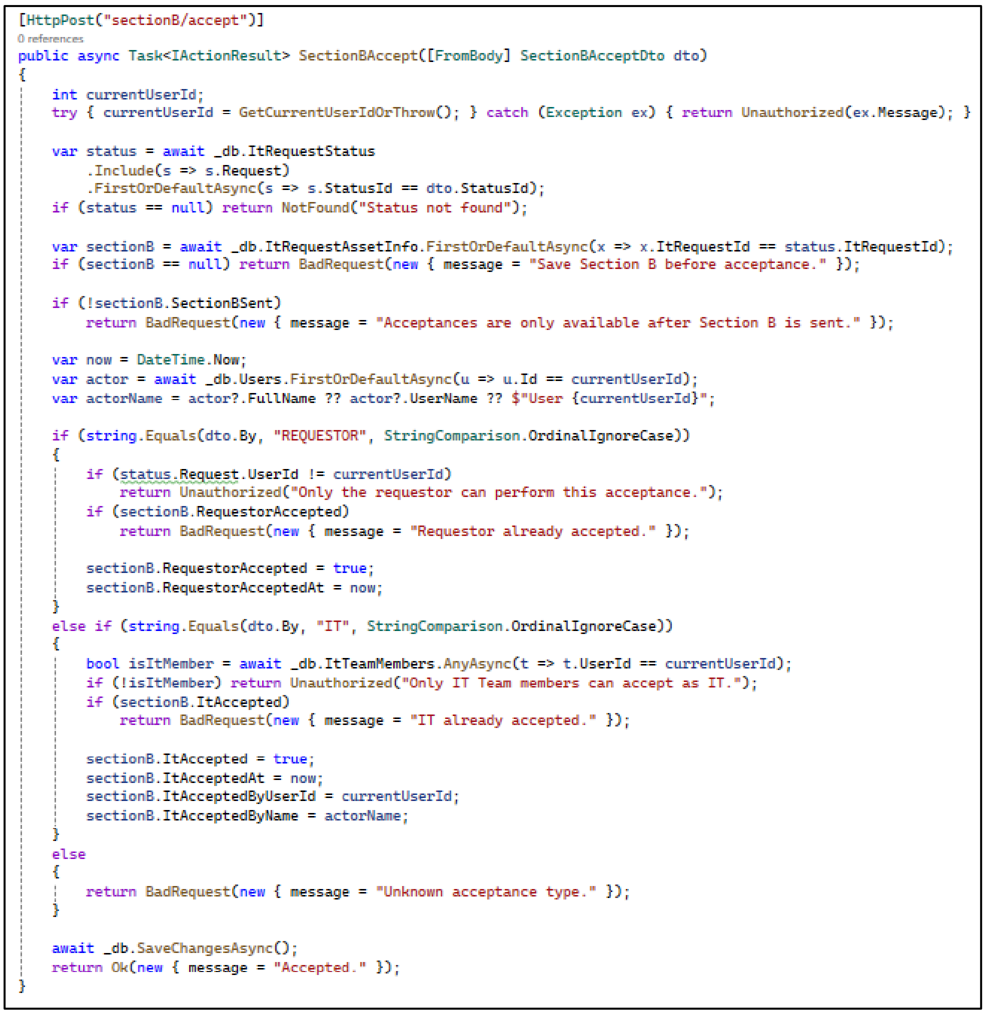
- **[HttpPost("sectionB/send")]** (Used by: SectionBEdit.cshtml, Send button)
- **Purpose:**
  Mark Section B as "sent" so the requestor can acknowledge. This locks Section B content until both acceptances are done.
- **Key functionalities:**
  - Requires IT role and OverallStatus = Approved.
  - Finds or creates the Section B row and applies any incoming fields from the form.
  - Validates that at least one identifier is present (Asset No or Machine ID or IP Address).
  - Refuses if it was already sent. Otherwise stamps SectionBSent = true and time, and locks edits.



```
[HttpPost("sectionB/send")]
0 references
public async Task<IActionResult> SectionBSend([FromBody] SectionBSendDto dto)
{
    int currentUserId;
    try { currentUserId = GetCurrentUserIdOrThrow(); } catch (Exception ex) { return Unauthorized(ex.Message); }

    var status = await _db.ItRequestStatus
        .Include(s => s.Request)
        .FirstOrDefaultAsync(s => s.StatusId == dto.StatusId);

    if (status == null) return NotFound(new { message = "Status not found" });
    if ((status.OverallStatus ?? "Pending") != "Approved")
        return BadRequest(new { message = "Section B can be sent only after OverallStatus is Approved." });

    bool isItMember = await _db.ItTeamMembers.AnyAsync(t => t.UserId == currentUserId);
    if (!isItMember) return Unauthorized(new { message = "Only IT can send Section B." });

    var b = await _db.ItRequestAssetInfo.FirstOrDefaultAsync(x => x.ItRequestId == status.ItRequestId);
    if (b?.SectionBSent == true)
        return BadRequest(new { message = "Section B already sent." });

    var editor = await _db.Users.FirstOrDefaultAsync(u => u.Id == currentUserId);
    var editorName = editor?.FullName ?? editor?.UserName ?? $"User {currentUserId}";
    var now = DateTime.Now;

    if (b == null)
    {
        b = new ItRequestAssetInfo
        {
            ItRequestId = status.ItRequestId,
            RequestorAccepted = false,
            RequestorAcceptedAt = null,
            ItAccepted = false,
            ItAcceptedAt = null,
            ItAcceptedByUserId = null,
            ItAcceptedByName = null
        };
        _db.ItRequestAssetInfo.Add(b);
    }

    // Apply incoming fields (allow send without prior draft)
    if (dto.AssetNo != null) b.AssetNo = dto.AssetNo.Trim();
    if (dto.MachineId != null) b.MachineId = dto.MachineId.Trim();
    if (dto.IpAddress != null) b.IpAddress = dto.IpAddress.Trim();
    if (dto.WiredMac != null) b.WiredMac = dto.WiredMac.Trim();
    if (dto.WifiMac != null) b.WifiMac = dto.WifiMac.Trim();
    if (dto.DialupAcc != null) b.DialupAcc = dto.DialupAcc.Trim();
    if (dto.Remarks != null) b.Remarks = dto.Remarks.Trim();

    b.LastEditedAt = now;
    b.LastEditedByUserId = currentUserId;
    b.LastEditedByName = editorName;

    // Require at least one identifier
    if (string.IsNullOrWhiteSpace(b.AssetNo) &&
        string.IsNullOrWhiteSpace(b.MachineId) &&
        string.IsNullOrWhiteSpace(b.IpAddress))
        return BadRequest(new { message = "Provide at least Asset No / Machine ID / IP Address before sending." });

    b.SectionBSent = true;
    b.SectionBSentAt = now;

    await _db.SaveChangesAsync();
    return Ok(new { message = "Section B sent. It is now locked for editing until both acceptances complete." });
}
```

- **[HttpPost("sectionB/accept")]** (Used by: SectionBEdit.cshtml, Accept buttons)
- **Purpose:**

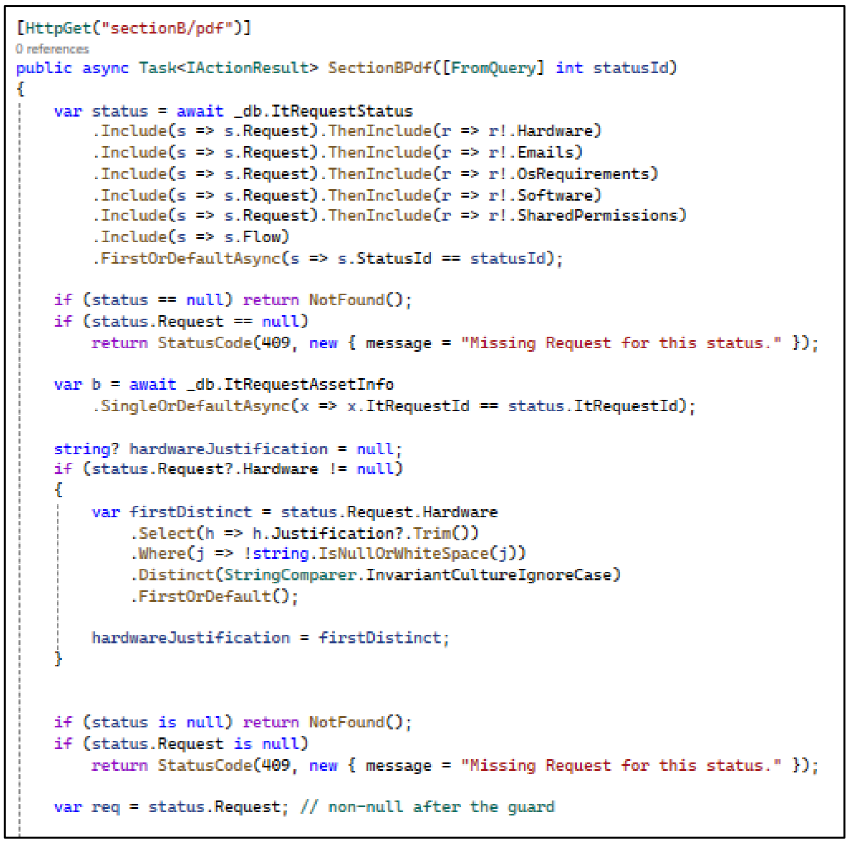
  Record the acceptance for Section B by either the requestor or IT after it has been sent.
- **Key functionalities:**
  - Ensures a Section B row exists and has been sent.
  - By = "REQUESTOR": only the original requester can accept.
  - By = "IT": only IT team members can accept as IT; also records who accepted and when.
  - Blocks duplicate acceptances and unknown By values.



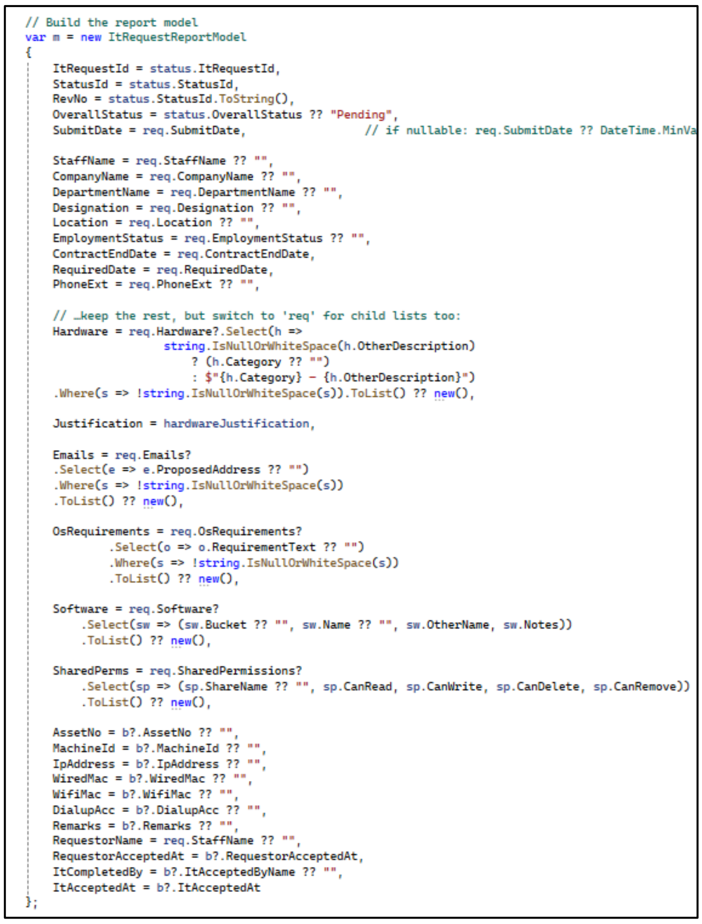
```
[HttpPost("sectionB/accept")]
0 references
public async Task<IActionResult> SectionBAccept([FromBody] SectionBAcceptDto dto)
{
    int currentUserId;
    try { currentUserId = GetCurrentUserIdOrThrow(); } catch (Exception ex) { return Unauthorized(ex.Message); }

    var status = await _db.ItRequestStatus
        .Include(s => s.Request)
        .FirstOrDefaultAsync(s => s.StatusId == dto.StatusId);
    if (status == null) return NotFound("Status not found");

    var sectionB = await _db.ItRequestAssetInfo.FirstOrDefaultAsync(x => x.ItRequestId == status.ItRequestId);
    if (sectionB == null) return BadRequest(new { message = "Save Section B before acceptance." });

    if (!sectionB.SectionBSent)
        return BadRequest(new { message = "Acceptances are only available after Section B is sent." });

    var now = DateTime.Now;
    var actor = await _db.Users.FirstOrDefaultAsync(u => u.Id == currentUserId);
    var actorName = actor?.FullName ?? actor?.UserName ?? $"User {currentUserId}";

    if (string.Equals(dto.By, "REQUESTOR", StringComparison.OrdinalIgnoreCase))
    {
        if (status.Request.UserId != currentUserId)
            return Unauthorized("Only the requestor can perform this acceptance.");
        if (sectionB.RequestorAccepted)
            return BadRequest(new { message = "Requestor already accepted." });

        sectionB.RequestorAccepted = true;
        sectionB.RequestorAcceptedAt = now;
    }
    else if (string.Equals(dto.By, "IT", StringComparison.OrdinalIgnoreCase))
    {
        bool isItMember = await _db.ItTeamMembers.AnyAsync(t => t.UserId == currentUserId);
        if (!isItMember) return Unauthorized("Only IT Team members can accept as IT.");
        if (sectionB.ItAccepted)
            return BadRequest(new { message = "IT already accepted." });

        sectionB.ItAccepted = true;
        sectionB.ItAcceptedAt = now;
        sectionB.ItAcceptedByUserId = currentUserId;
        sectionB.ItAcceptedByName = actorName;
    }
    else
    {
        return BadRequest(new { message = "Unknown acceptance type." });
    }

    await _db.SaveChangesAsync();
    return Ok(new { message = "Accepted." });
}
```

- **[HttpGet("sectionB/pdf")]** (Used by: SectionBEdit.cshtml, Download/Print)
- **Purpose:**

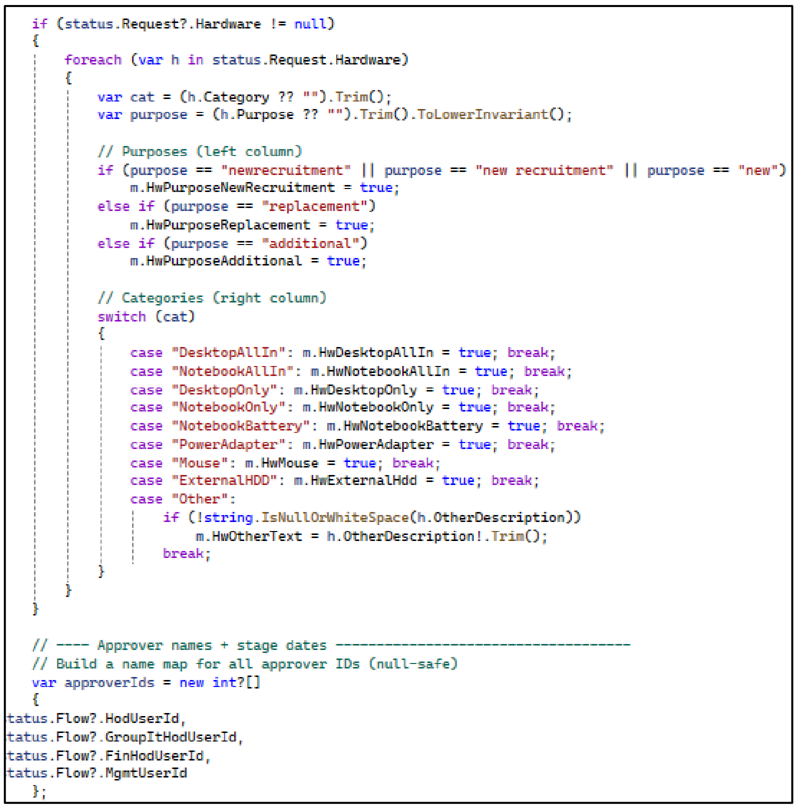
  Generate the final Section B PDF once both the requestor and IT have accepted.
- **Key functionalities:**
  - Loads the full request graph and Section B details, plus approver names and dates.
  - Picks a single hardware justification to avoid repeating the same sentence across multiple items.
  - Checks the final gate: both RequestorAccepted and ItAccepted must be true.
  - Renders and returns the PDF so the browser can display or download it.



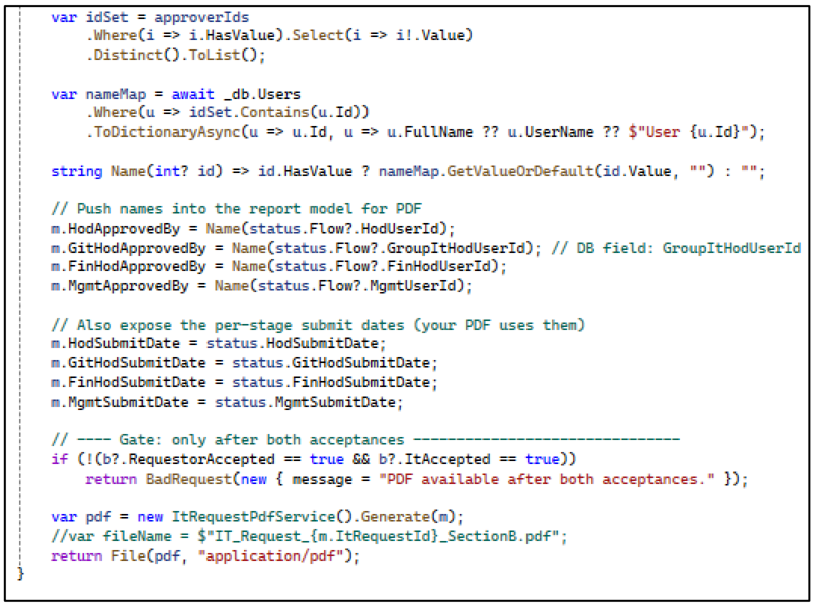
```
[HttpGet("sectionB/pdf")]
0 references
public async Task<IActionResult> SectionBPdf([FromQuery] int statusId)
{
    var status = await _db.ItRequestStatus
        .Include(s => s.Request).ThenInclude(r => r!.Hardware)
        .Include(s => s.Request).ThenInclude(r => r!.Emails)
        .Include(s => s.Request).ThenInclude(r => r!.OsRequirements)
        .Include(s => s.Request).ThenInclude(r => r!.Software)
        .Include(s => s.Request).ThenInclude(r => r!.SharedPermissions)
        .Include(s => s.Flow)
        .FirstOrDefaultAsync(s => s.StatusId == statusId);

    if (status == null) return NotFound();
    if (status.Request == null)
        return StatusCode(409, new { message = "Missing Request for this status." });

    var b = await _db.ItRequestAssetInfo
        .SingleOrDefaultAsync(x => x.ItRequestId == status.ItRequestId);

    string? hardwareJustification = null;
    if (status.Request?.Hardware != null)
    {
        var firstDistinct = status.Request.Hardware
            .Select(h => h.Justification?.Trim())
            .Where(j => !string.IsNullOrWhiteSpace(j))
            .Distinct(StringComparer.InvariantCultureIgnoreCase)
            .FirstOrDefault();

        hardwareJustification = firstDistinct;
    }


    if (status is null) return NotFound();
    if (status.Request is null)
        return StatusCode(409, new { message = "Missing Request for this status." });

    var req = status.Request; // non-null after the guard
```



```
// Build the report model
var m = new ItRequestReportModel
{
    ItRequestId = status.ItRequestId,
    StatusId = status.StatusId,
    RevNo = status.StatusId.ToString(),
    OverallStatus = status.OverallStatus ?? "Pending",
    SubmitDate = req.SubmitDate,                 // if nullable: req.SubmitDate ?? DateTime.MinVa

    StaffName = req.StaffName ?? "",
    CompanyName = req.CompanyName ?? "",
    DepartmentName = req.DepartmentName ?? "",
    Designation = req.Designation ?? "",
    Location = req.Location ?? "",
    EmploymentStatus = req.EmploymentStatus ?? "",
    ContractEndDate = req.ContractEndDate,
    RequiredDate = req.RequiredDate,
    PhoneExt = req.PhoneExt ?? "",

    // …keep the rest, but switch to 'req' for child lists too:
    Hardware = req.Hardware?.Select(h =>
                    string.IsNullOrWhiteSpace(h.OtherDescription)
                        ? (h.Category ?? "")
                        : $"{h.Category} - {h.OtherDescription}")
    .Where(s => !string.IsNullOrWhiteSpace(s)).ToList() ?? new(),

    Justification = hardwareJustification,

    Emails = req.Emails?
    .Select(e => e.ProposedAddress ?? "")
    .Where(s => !string.IsNullOrWhiteSpace(s))
    .ToList() ?? new(),

    OsRequirements = req.OsRequirements?
            .Select(o => o.RequirementText ?? "")
            .Where(s => !string.IsNullOrWhiteSpace(s))
            .ToList() ?? new(),

    Software = req.Software?
        .Select(sw => (sw.Bucket ?? "", sw.Name ?? "", sw.OtherName, sw.Notes))
        .ToList() ?? new(),

    SharedPerms = req.SharedPermissions?
        .Select(sp => (sp.ShareName ?? "", sp.CanRead, sp.CanWrite, sp.CanDelete, sp.CanRemove))
        .ToList() ?? new(),

    AssetNo = b?.AssetNo ?? "",
    MachineId = b?.MachineId ?? "",
    IpAddress = b?.IpAddress ?? "",
    WiredMac = b?.WiredMac ?? "",
    WifiMac = b?.WifiMac ?? "",
    DialupAcc = b?.DialupAcc ?? "",
    Remarks = b?.Remarks ?? "",
    RequestorName = req.StaffName ?? "",
    RequestorAcceptedAt = b?.RequestorAcceptedAt,
    ItCompletedBy = b?.ItAcceptedByName ?? "",
    ItAcceptedAt = b?.ItAcceptedAt
};
```



```
if (status.Request?.Hardware != null)
{
    foreach (var h in status.Request.Hardware)
    {
        var cat = (h.Category ?? "").Trim();
        var purpose = (h.Purpose ?? "").Trim().ToLowerInvariant();

        // Purposes (left column)
        if (purpose == "newrecruitment" || purpose == "new recruitment" || purpose == "new")
            m.HwPurposeNewRecruitment = true;
        else if (purpose == "replacement")
            m.HwPurposeReplacement = true;
        else if (purpose == "additional")
            m.HwPurposeAdditional = true;

        // Categories (right column)
        switch (cat)
        {
            case "DesktopAllIn": m.HwDesktopAllIn = true; break;
            case "NotebookAllIn": m.HwNotebookAllIn = true; break;
            case "DesktopOnly": m.HwDesktopOnly = true; break;
            case "NotebookOnly": m.HwNotebookOnly = true; break;
            case "NotebookBattery": m.HwNotebookBattery = true; break;
            case "PowerAdapter": m.HwPowerAdapter = true; break;
            case "Mouse": m.HwMouse = true; break;
            case "ExternalHDD": m.HwExternalHdd = true; break;
            case "Other":
                if (!string.IsNullOrWhiteSpace(h.OtherDescription))
                    m.HwOtherText = h.OtherDescription!.Trim();
                break;
        }
    }
}

// ---- Approver names + stage dates ------------------------------------
// Build a name map for all approver IDs (null-safe)
var approverIds = new int?[]
{
tatus.Flow?.HodUserId,
tatus.Flow?.GroupItHodUserId,
tatus.Flow?.FinHodUserId,
tatus.Flow?.MgmtUserId
};
```

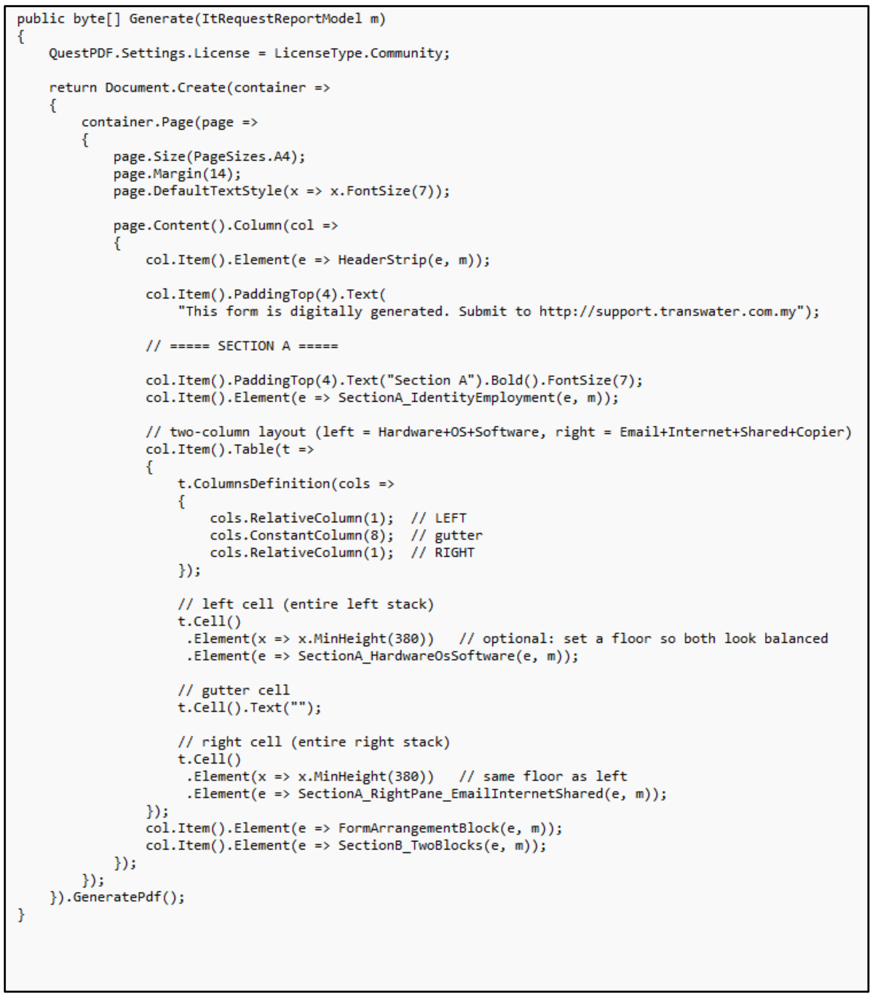


```
    var idSet = approverIds
        .Where(i => i.HasValue).Select(i => i!.Value)
        .Distinct().ToList();

    var nameMap = await _db.Users
        .Where(u => idSet.Contains(u.Id))
        .ToDictionaryAsync(u => u.Id, u => u.FullName ?? u.UserName ?? $"User {u.Id}");

    string Name(int? id) => id.HasValue ? nameMap.GetValueOrDefault(id.Value, "") : "";

    // Push names into the report model for PDF
    m.HodApprovedBy = Name(status.Flow?.HodUserId);
    m.GitHodApprovedBy = Name(status.Flow?.GroupItHodUserId); // DB field: GroupItHodUserId
    m.FinHodApprovedBy = Name(status.Flow?.FinHodUserId);
    m.MgmtApprovedBy = Name(status.Flow?.MgmtUserId);

    // Also expose the per-stage submit dates (your PDF uses them)
    m.HodSubmitDate = status.HodSubmitDate;
    m.GitHodSubmitDate = status.GitHodSubmitDate;
    m.FinHodSubmitDate = status.FinHodSubmitDate;
    m.MgmtSubmitDate = status.MgmtSubmitDate;

    // ---- Gate: only after both acceptances --------------------------------
    if (!(b?.RequestorAccepted == true && b?.ItAccepted == true))
        return BadRequest(new { message = "PDF available after both acceptances." });

    var pdf = new ItRequestPdfService().Generate(m);
    //var fileName = $"IT_Request_{m.ItRequestId}_SectionB.pdf";
    return File(pdf, "application/pdf");
}
```

SERVICES

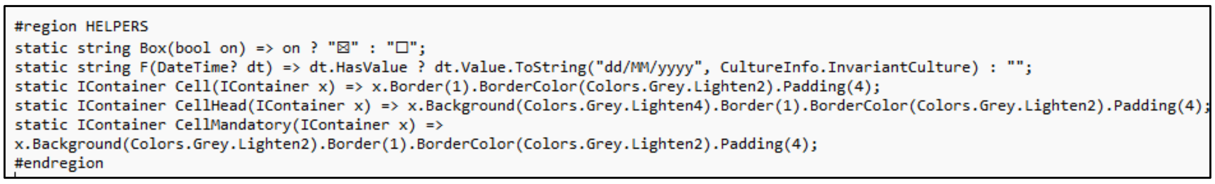
- **ItRequestPDFService.cs**
  - ➢ Generate(ItRequestReportModel m)

This is the entry point. It sets the QuestPDF license, defines an A4 page with small margins and a 7pt default text, then composes the document top-to-bottom. The layout order mirrors how a user reads the form: header strip, a small system note, Section A (identity/employment), a two-column stack for requirements, the approval flow strip, and finally Section B. The two-column table keeps the left and right panes visually balanced by giving both a minimum height.



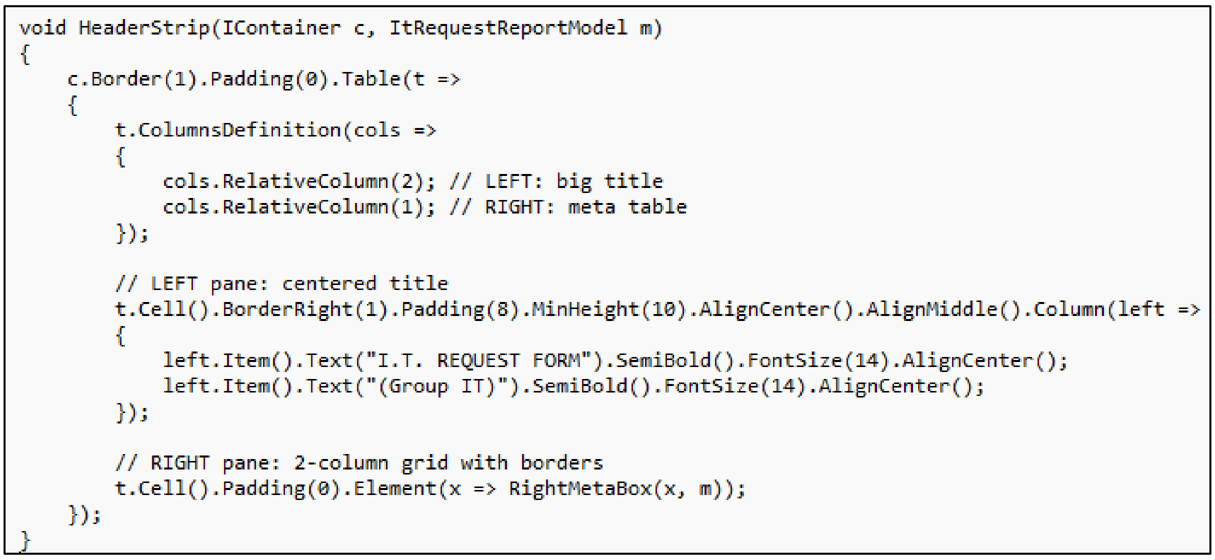
```
public byte[] Generate(ItRequestReportModel m)
{
    QuestPDF.Settings.License = LicenseType.Community;

    return Document.Create(container =>
    {
        container.Page(page =>
        {
            page.Size(PageSizes.A4);
            page.Margin(14);
            page.DefaultTextStyle(x => x.FontSize(7));

            page.Content().Column(col =>
            {
                col.Item().Element(e => HeaderStrip(e, m));

                col.Item().PaddingTop(4).Text(
                    "This form is digitally generated. Submit to http://support.transwater.com.my");

                // ===== SECTION A =====

                col.Item().PaddingTop(4).Text("Section A").Bold().FontSize(7);
                col.Item().Element(e => SectionA_IdentityEmployment(e, m));

                // two-column layout (left = Hardware+OS+Software, right = Email+Internet+Shared+Copier)
                col.Item().Table(t =>
                {
                    t.ColumnsDefinition(cols =>
                    {
                        cols.RelativeColumn(1);  // LEFT
                        cols.ConstantColumn(8);  // gutter
                        cols.RelativeColumn(1);  // RIGHT
                    });

                    // left cell (entire left stack)
                    t.Cell()
                     .Element(x => x.MinHeight(380))   // optional: set a floor so both look balanced
                     .Element(e => SectionA_HardwareOsSoftware(e, m));

                    // gutter cell
                    t.Cell().Text("");

                    // right cell (entire right stack)
                    t.Cell()
                     .Element(x => x.MinHeight(380))   // same floor as left
                     .Element(e => SectionA_RightPane_EmailInternetShared(e, m));
                });
                col.Item().Element(e => FormArrangementBlock(e, m));
                col.Item().Element(e => SectionB_TwoBlocks(e, m));
            });
        });
    }).GeneratePdf();
}
```

- **Helpers and shared styles**
  - These helpers keep the rest of the code readable. Box draws a tick or empty box, F formats dates in day-month-year, and Cell/CellHead apply consistent borders, fills, and padding so all your tables feel like one family.



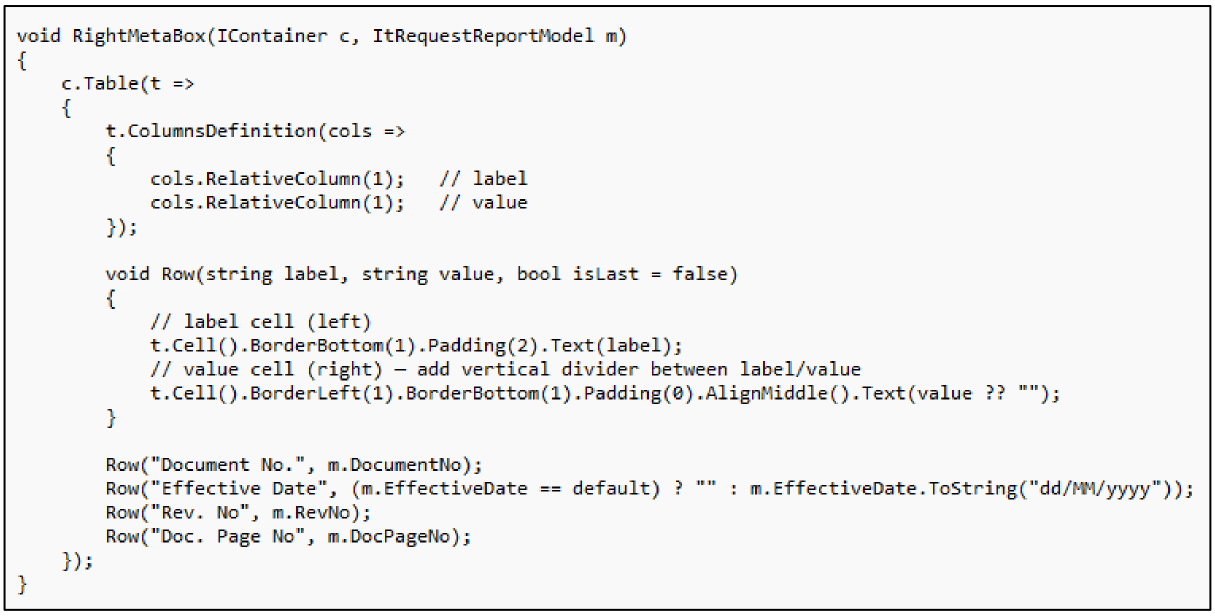
```
#region HELPERS
static string Box(bool on) => on ? "☒" : "☐";
static string F(DateTime? dt) => dt.HasValue ? dt.Value.ToString("dd/MM/yyyy", CultureInfo.InvariantCulture) : "";
static IContainer Cell(IContainer x) => x.Border(1).BorderColor(Colors.Grey.Lighten2).Padding(4);
static IContainer CellHead(IContainer x) => x.Background(Colors.Grey.Lighten4).Border(1).BorderColor(Colors.Grey.Lighten2).Padding(4);
static IContainer CellMandatory(IContainer x) =>
x.Background(Colors.Grey.Lighten2).Border(1).BorderColor(Colors.Grey.Lighten2).Padding(4);
#endregion
```

- **HeaderStrip: title on the left, meta box on the right**
    - The header creates a boxed strip that matches your screenshot style. The left is a centered two-line title. The right is a tidy two-column table for metadata like document number and revision. A vertical divider separates the two panes, which makes the header feel like a single unit.



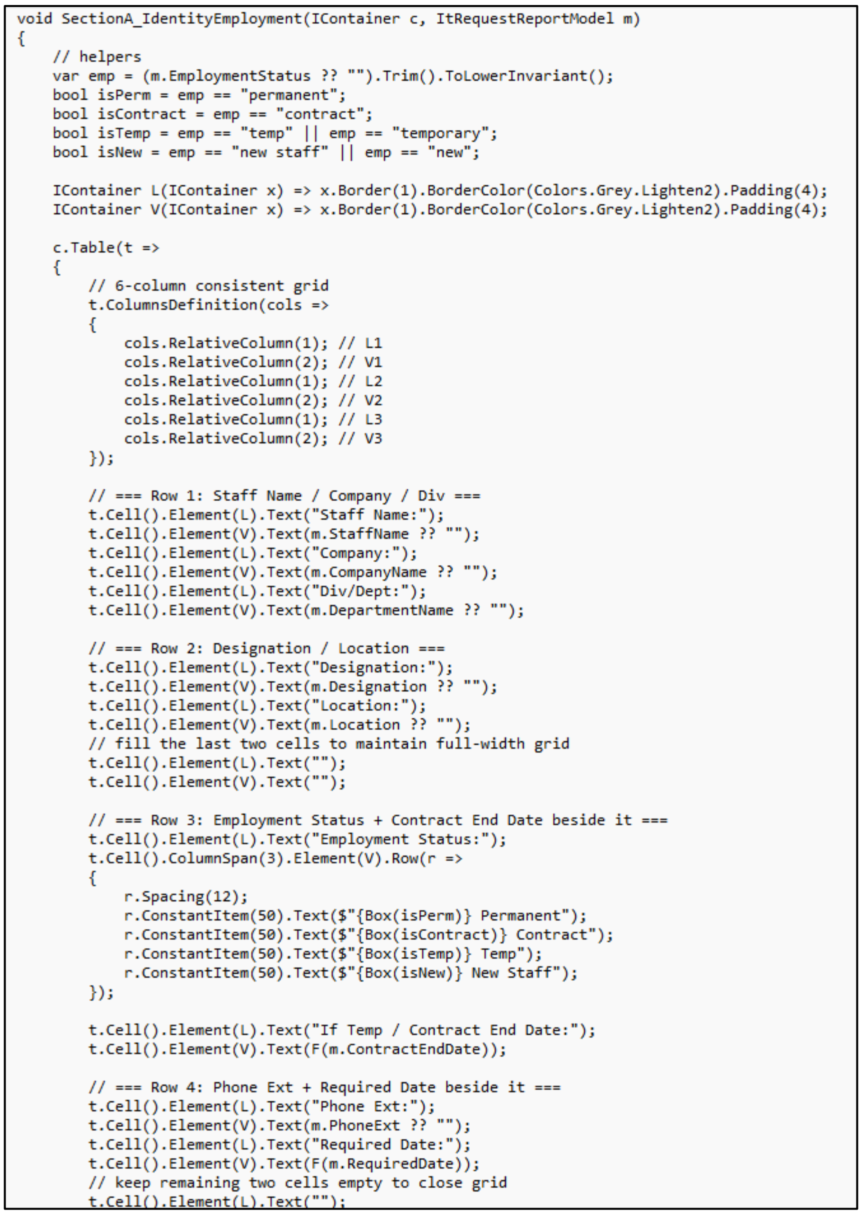
```
void HeaderStrip(IContainer c, ItRequestReportModel m)
{
    c.Border(1).Padding(0).Table(t =>
    {
        t.ColumnsDefinition(cols =>
        {
            cols.RelativeColumn(2); // LEFT: big title
            cols.RelativeColumn(1); // RIGHT: meta table
        });

        // LEFT pane: centered title
        t.Cell().BorderRight(1).Padding(8).MinHeight(10).AlignCenter().AlignMiddle().Column(left =>
        {
            left.Item().Text("I.T. REQUEST FORM").SemiBold().FontSize(14).AlignCenter();
            left.Item().Text("(Group IT)").SemiBold().FontSize(14).AlignCenter();
        });

        // RIGHT pane: 2-column grid with borders
        t.Cell().Padding(0).Element(x => RightMetaBox(x, m));
    });
}
```

- **RightMetaBox: document metadata grid**
    - This helper draws the right-hand table in the header. Each row has a label and a value with a clean vertical divider. Date formatting is applied where relevant to retain a consistent look.



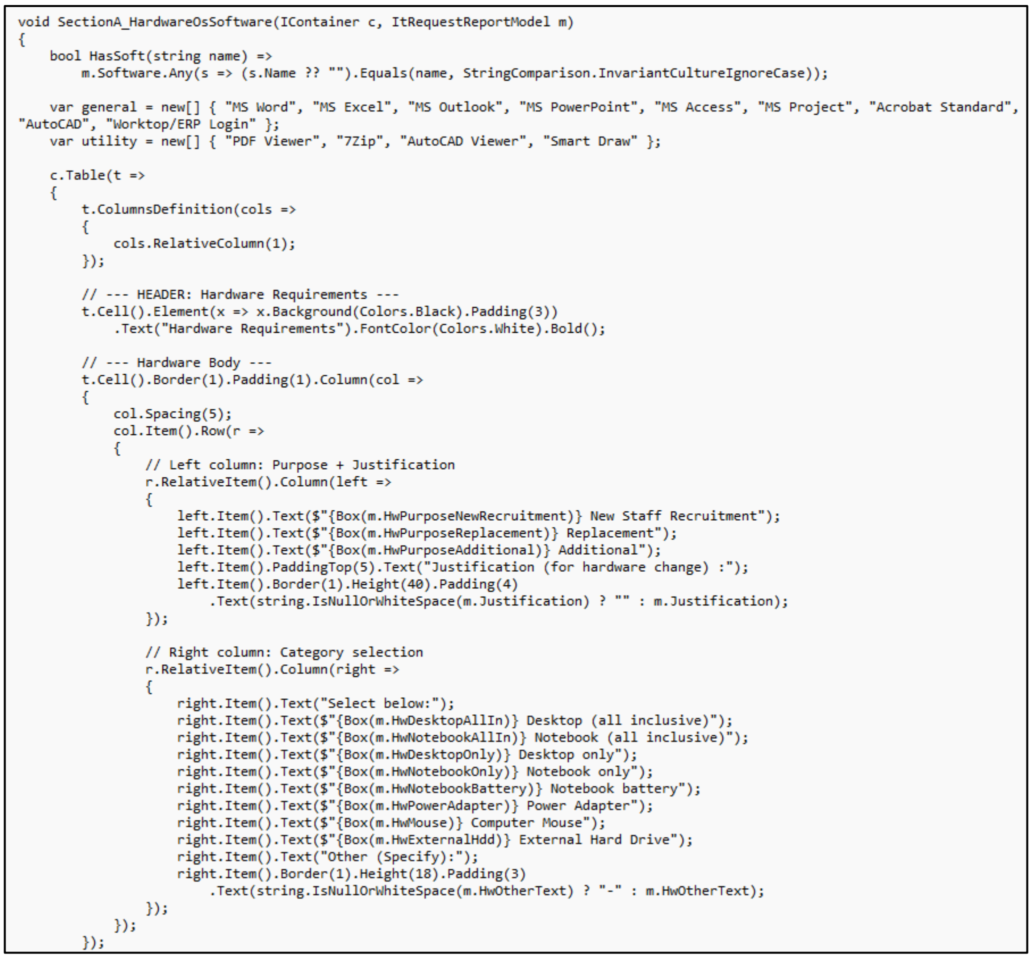
```
void RightMetaBox(IContainer c, ItRequestReportModel m)
{
    c.Table(t =>
    {
        t.ColumnsDefinition(cols =>
        {
            cols.RelativeColumn(1);   // label
            cols.RelativeColumn(1);   // value
        });

        void Row(string label, string value, bool isLast = false)
        {
            // label cell (left)
            t.Cell().BorderBottom(1).Padding(2).Text(label);
            // value cell (right) — add vertical divider between label/value
            t.Cell().BorderLeft(1).BorderBottom(1).Padding(0).AlignMiddle().Text(value ?? "");
        }

        Row("Document No.", m.DocumentNo);
        Row("Effective Date", (m.EffectiveDate == default) ? "" : m.EffectiveDate.ToString("dd/MM/yyyy"));
        Row("Rev. No", m.RevNo);
        Row("Doc. Page No", m.DocPageNo);
    });
}
```

- **Section A: Identity & Employment block**
    - This compact grid puts all identity and employment details into one six-column table so labels and values line up neatly. Employment status is presented as tick boxes, and contract end date only shows when relevant. Empty cells are deliberately filled to keep the grid aligned.

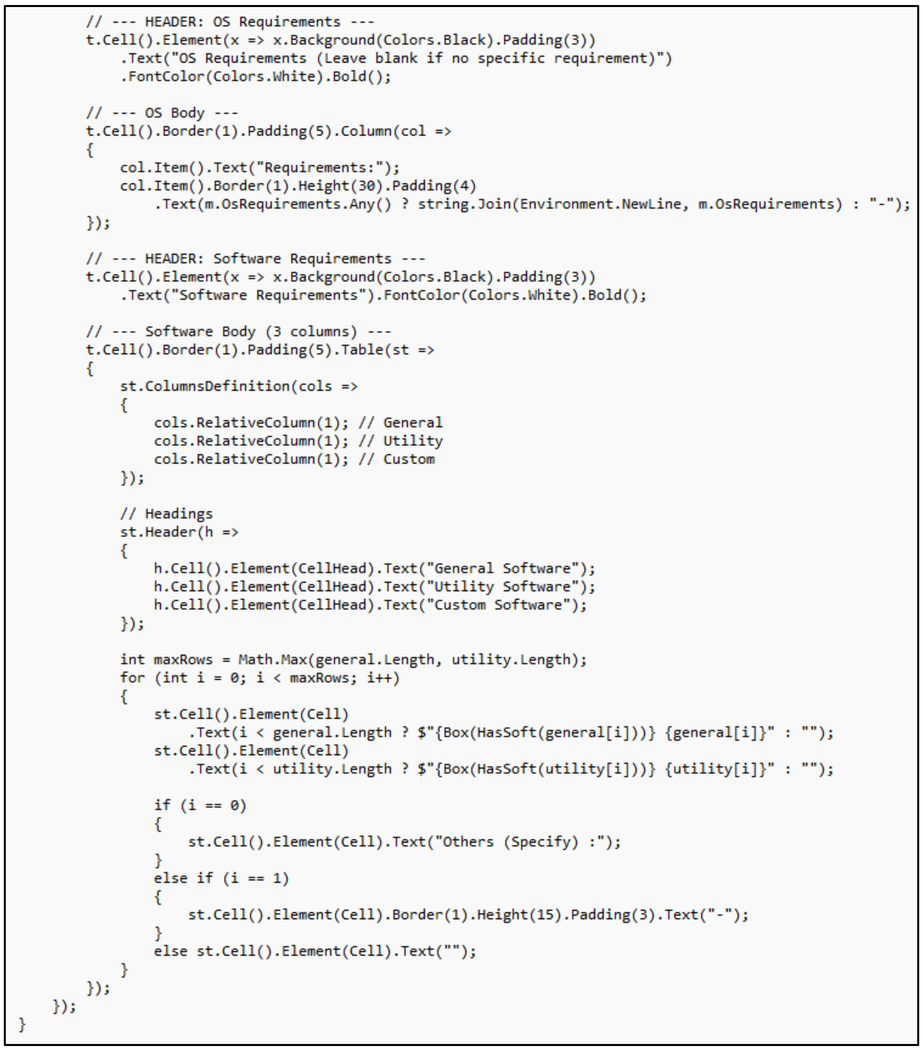


```
void SectionA_IdentityEmployment(IContainer c, ItRequestReportModel m)
{
    // helpers
    var emp = (m.EmploymentStatus ?? "").Trim().ToLowerInvariant();
    bool isPerm = emp == "permanent";
    bool isContract = emp == "contract";
    bool isTemp = emp == "temp" || emp == "temporary";
    bool isNew = emp == "new staff" || emp == "new";

    IContainer L(IContainer x) => x.Border(1).BorderColor(Colors.Grey.Lighten2).Padding(4);
    IContainer V(IContainer x) => x.Border(1).BorderColor(Colors.Grey.Lighten2).Padding(4);

    c.Table(t =>
    {
        // 6-column consistent grid
        t.ColumnsDefinition(cols =>
        {
            cols.RelativeColumn(1); // L1
            cols.RelativeColumn(2); // V1
            cols.RelativeColumn(1); // L2
            cols.RelativeColumn(2); // V2
            cols.RelativeColumn(1); // L3
            cols.RelativeColumn(2); // V3
        });

        // === Row 1: Staff Name / Company / Div ===
        t.Cell().Element(L).Text("Staff Name:");
        t.Cell().Element(V).Text(m.StaffName ?? "");
        t.Cell().Element(L).Text("Company:");
        t.Cell().Element(V).Text(m.CompanyName ?? "");
        t.Cell().Element(L).Text("Div/Dept:");
        t.Cell().Element(V).Text(m.DepartmentName ?? "");

        // === Row 2: Designation / Location ===
        t.Cell().Element(L).Text("Designation:");
        t.Cell().Element(V).Text(m.Designation ?? "");
        t.Cell().Element(L).Text("Location:");
        t.Cell().Element(V).Text(m.Location ?? "");
        // fill the last two cells to maintain full-width grid
        t.Cell().Element(L).Text("");
        t.Cell().Element(V).Text("");

        // === Row 3: Employment Status + Contract End Date beside it ===
        t.Cell().Element(L).Text("Employment Status:");
        t.Cell().ColumnSpan(3).Element(V).Row(r =>
        {
            r.Spacing(12);
            r.ConstantItem(50).Text($"{Box(isPerm)} Permanent");
            r.ConstantItem(50).Text($"{Box(isContract)} Contract");
            r.ConstantItem(50).Text($"{Box(isTemp)} Temp");
            r.ConstantItem(50).Text($"{Box(isNew)} New Staff");
        });

        t.Cell().Element(L).Text("If Temp / Contract End Date:");
        t.Cell().Element(V).Text(F(m.ContractEndDate));

        // === Row 4: Phone Ext + Required Date beside it ===
        t.Cell().Element(L).Text("Phone Ext:");
        t.Cell().Element(V).Text(m.PhoneExt ?? "");
        t.Cell().Element(L).Text("Required Date:");
        t.Cell().Element(V).Text(F(m.RequiredDate));
        // keep remaining two cells empty to close grid
        t.Cell().Element(L).Text("");
```

- **Section A (left): Hardware, OS, and Software**
    - This column groups all tech requirements in a clear progression. It starts with hardware purpose and categories, includes a justification box, follows with OS requirements as free text, then renders a three-column software matrix. The `HasSoft` helper checks user-selected software and draws ticked boxes for the common lists while leaving space for custom entries.

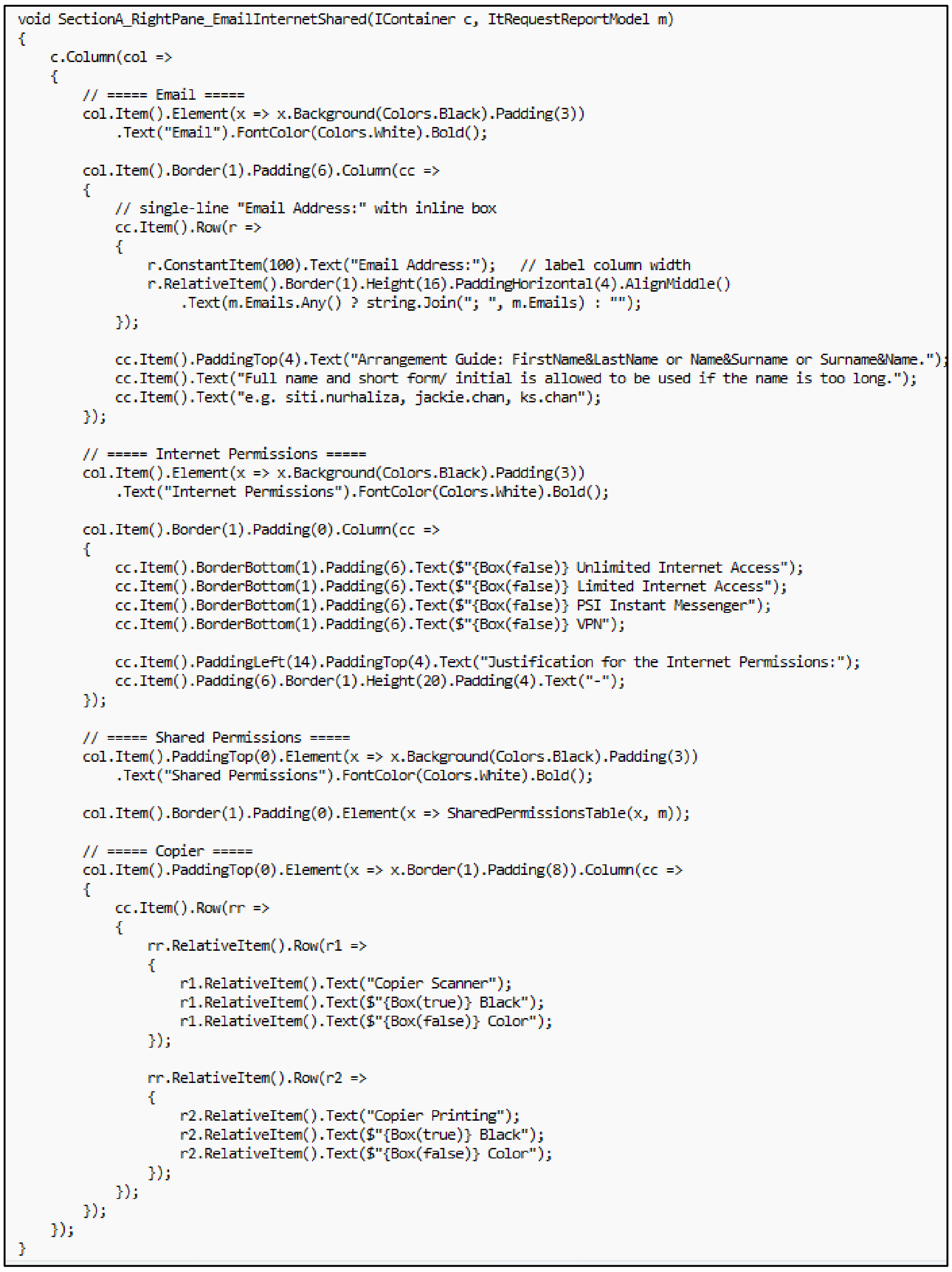


```
void SectionA_HardwareOsSoftware(IContainer c, ItRequestReportModel m)
{
    bool HasSoft(string name) =>
        m.Software.Any(s => (s.Name ?? "").Equals(name, StringComparison.InvariantCultureIgnoreCase));

    var general = new[] { "MS Word", "MS Excel", "MS Outlook", "MS PowerPoint", "MS Access", "MS Project", "Acrobat Standard",
"AutoCAD", "Worktop/ERP Login" };
    var utility = new[] { "PDF Viewer", "7Zip", "AutoCAD Viewer", "Smart Draw" };

    c.Table(t =>
    {
        t.ColumnsDefinition(cols =>
        {
            cols.RelativeColumn(1);
        });

        // --- HEADER: Hardware Requirements ---
        t.Cell().Element(x => x.Background(Colors.Black).Padding(3))
            .Text("Hardware Requirements").FontColor(Colors.White).Bold();

        // --- Hardware Body ---
        t.Cell().Border(1).Padding(1).Column(col =>
        {
            col.Spacing(5);
            col.Item().Row(r =>
            {
                // Left column: Purpose + Justification
                r.RelativeItem().Column(left =>
                {
                    left.Item().Text($"{Box(m.HwPurposeNewRecruitment)} New Staff Recruitment");
                    left.Item().Text($"{Box(m.HwPurposeReplacement)} Replacement");
                    left.Item().Text($"{Box(m.HwPurposeAdditional)} Additional");
                    left.Item().PaddingTop(5).Text("Justification (for hardware change) :");
                    left.Item().Border(1).Height(40).Padding(4)
                        .Text(string.IsNullOrWhiteSpace(m.Justification) ? "" : m.Justification);
                });

                // Right column: Category selection
                r.RelativeItem().Column(right =>
                {
                    right.Item().Text("Select below:");
                    right.Item().Text($"{Box(m.HwDesktopAllIn)} Desktop (all inclusive)");
                    right.Item().Text($"{Box(m.HwNotebookAllIn)} Notebook (all inclusive)");
                    right.Item().Text($"{Box(m.HwDesktopOnly)} Desktop only");
                    right.Item().Text($"{Box(m.HwNotebookOnly)} Notebook only");
                    right.Item().Text($"{Box(m.HwNotebookBattery)} Notebook battery");
                    right.Item().Text($"{Box(m.HwPowerAdapter)} Power Adapter");
                    right.Item().Text($"{Box(m.HwMouse)} Computer Mouse");
                    right.Item().Text($"{Box(m.HwExternalHdd)} External Hard Drive");
                    right.Item().Text("Other (Specify):");
                    right.Item().Border(1).Height(18).Padding(3)
                        .Text(string.IsNullOrWhiteSpace(m.HwOtherText) ? "-" : m.HwOtherText);
                });
            });
        });
```



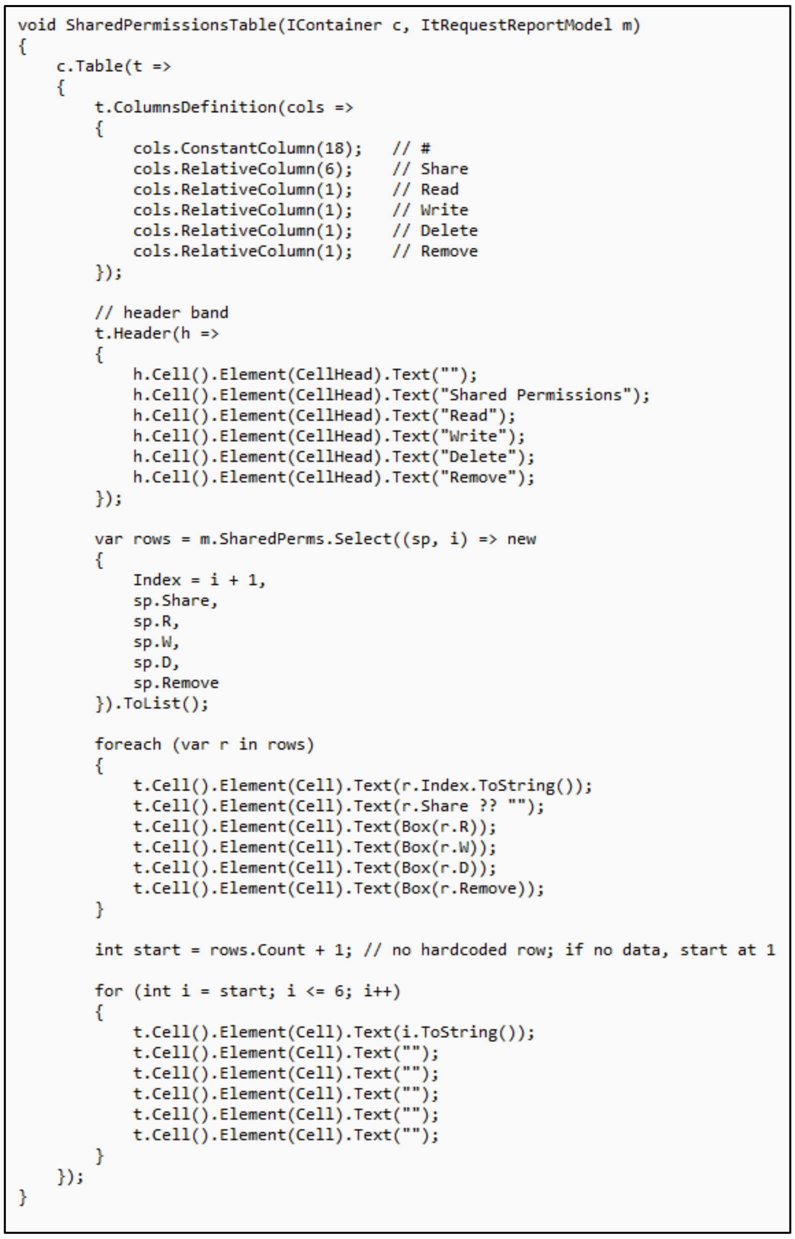
```
        // --- HEADER: OS Requirements ---
        t.Cell().Element(x => x.Background(Colors.Black).Padding(3))
            .Text("OS Requirements (Leave blank if no specific requirement)")
            .FontColor(Colors.White).Bold();

        // --- OS Body ---
        t.Cell().Border(1).Padding(5).Column(col =>
        {
            col.Item().Text("Requirements:");
            col.Item().Border(1).Height(30).Padding(4)
                .Text(m.OsRequirements.Any() ? string.Join(Environment.NewLine, m.OsRequirements) : "-");
        });

        // --- HEADER: Software Requirements ---
        t.Cell().Element(x => x.Background(Colors.Black).Padding(3))
            .Text("Software Requirements").FontColor(Colors.White).Bold();

        // --- Software Body (3 columns) ---
        t.Cell().Border(1).Padding(5).Table(st =>
        {
            st.ColumnsDefinition(cols =>
            {
                cols.RelativeColumn(1); // General
                cols.RelativeColumn(1); // Utility
                cols.RelativeColumn(1); // Custom
            });

            // Headings
            st.Header(h =>
            {
                h.Cell().Element(CellHead).Text("General Software");
                h.Cell().Element(CellHead).Text("Utility Software");
                h.Cell().Element(CellHead).Text("Custom Software");
            });

            int maxRows = Math.Max(general.Length, utility.Length);
            for (int i = 0; i < maxRows; i++)
            {
                st.Cell().Element(Cell)
                    .Text(i < general.Length ? $"{Box(HasSoft(general[i]))} {general[i]}" : "");
                st.Cell().Element(Cell)
                    .Text(i < utility.Length ? $"{Box(HasSoft(utility[i]))} {utility[i]}" : "");

                if (i == 0)
                {
                    st.Cell().Element(Cell).Text("Others (Specify) :");
                }
                else if (i == 1)
                {
                    st.Cell().Element(Cell).Border(1).Height(15).Padding(3).Text("-");
                }
                else st.Cell().Element(Cell).Text("");
            }
        });
    });
}
```

- **Section A (right): Email, Internet permissions, Shared permissions, Copier**
    - This column covers the user's email address, outlines internet access options with a short justification box, renders the shared-folder ACL grid, and ends with copier preferences. The shared permissions table auto-numbers rows from the model and pads up to six rows to keep height predictable.



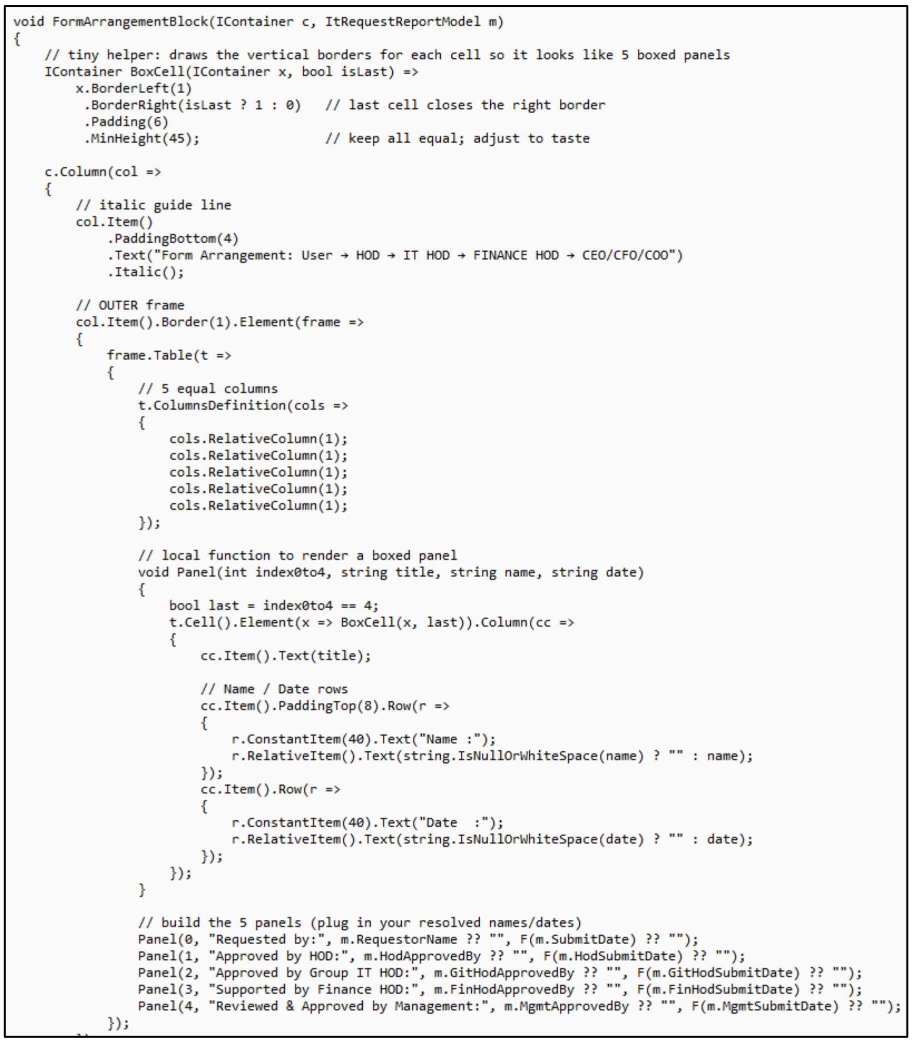
```
void SectionA_RightPane_EmailInternetShared(IContainer c, ItRequestReportModel m)
{
    c.Column(col =>
    {
        // ===== Email =====
        col.Item().Element(x => x.Background(Colors.Black).Padding(3))
            .Text("Email").FontColor(Colors.White).Bold();

        col.Item().Border(1).Padding(6).Column(cc =>
        {
            // single-line "Email Address:" with inline box
            cc.Item().Row(r =>
            {
                r.ConstantItem(100).Text("Email Address:");   // label column width
                r.RelativeItem().Border(1).Height(16).PaddingHorizontal(4).AlignMiddle()
                    .Text(m.Emails.Any() ? string.Join("; ", m.Emails) : "");
            });

            cc.Item().PaddingTop(4).Text("Arrangement Guide: FirstName&LastName or Name&Surname or Surname&Name.");
            cc.Item().Text("Full name and short form/ initial is allowed to be used if the name is too long.");
            cc.Item().Text("e.g. siti.nurhaliza, jackie.chan, ks.chan");
        });

        // ===== Internet Permissions =====
        col.Item().Element(x => x.Background(Colors.Black).Padding(3))
            .Text("Internet Permissions").FontColor(Colors.White).Bold();

        col.Item().Border(1).Padding(0).Column(cc =>
        {
            cc.Item().BorderBottom(1).Padding(6).Text($"{Box(false)} Unlimited Internet Access");
            cc.Item().BorderBottom(1).Padding(6).Text($"{Box(false)} Limited Internet Access");
            cc.Item().BorderBottom(1).Padding(6).Text($"{Box(false)} PSI Instant Messenger");
            cc.Item().BorderBottom(1).Padding(6).Text($"{Box(false)} VPN");

            cc.Item().PaddingLeft(14).PaddingTop(4).Text("Justification for the Internet Permissions:");
            cc.Item().Padding(6).Border(1).Height(20).Padding(4).Text("-");
        });

        // ===== Shared Permissions =====
        col.Item().PaddingTop(0).Element(x => x.Background(Colors.Black).Padding(3))
            .Text("Shared Permissions").FontColor(Colors.White).Bold();

        col.Item().Border(1).Padding(0).Element(x => SharedPermissionsTable(x, m));

        // ===== Copier =====
        col.Item().PaddingTop(0).Element(x => x.Border(1).Padding(8)).Column(cc =>
        {
            cc.Item().Row(rr =>
            {
                rr.RelativeItem().Row(r1 =>
                {
                    r1.RelativeItem().Text("Copier Scanner");
                    r1.RelativeItem().Text($"{Box(true)} Black");
                    r1.RelativeItem().Text($"{Box(false)} Color");
                });

                rr.RelativeItem().Row(r2 =>
                {
                    r2.RelativeItem().Text("Copier Printing");
                    r2.RelativeItem().Text($"{Box(true)} Black");
                    r2.RelativeItem().Text($"{Box(false)} Color");
                });
            });
        });
    });
}
```

- **SharedPermissionsTable: ACL grid with fixed height**
    - This table displays folder access in a way that is easy to scan. It shows row numbers, the share path, and boolean flags for read, write, delete, and remove. When the model has fewer than six entries, the table fills the remaining lines with blanks so the right pane’s height stays stable.



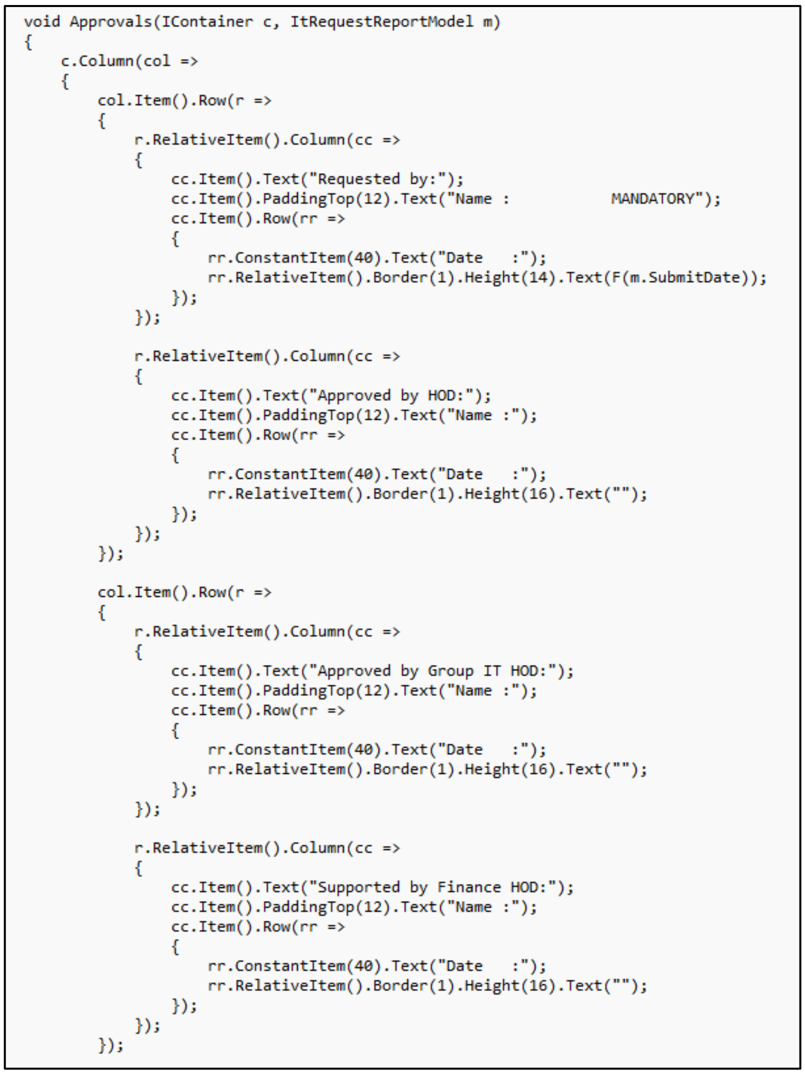
```
void SharedPermissionsTable(IContainer c, ItRequestReportModel m)
{
    c.Table(t =>
    {
        t.ColumnsDefinition(cols =>
        {
            cols.ConstantColumn(18);   // #
            cols.RelativeColumn(6);    // Share
            cols.RelativeColumn(1);    // Read
            cols.RelativeColumn(1);    // Write
            cols.RelativeColumn(1);    // Delete
            cols.RelativeColumn(1);    // Remove
        });

        // header band
        t.Header(h =>
        {
            h.Cell().Element(CellHead).Text("");
            h.Cell().Element(CellHead).Text("Shared Permissions");
            h.Cell().Element(CellHead).Text("Read");
            h.Cell().Element(CellHead).Text("Write");
            h.Cell().Element(CellHead).Text("Delete");
            h.Cell().Element(CellHead).Text("Remove");
        });

        var rows = m.SharedPerms.Select((sp, i) => new
        {
            Index = i + 1,
            sp.Share,
            sp.R,
            sp.W,
            sp.D,
            sp.Remove
        }).ToList();

        foreach (var r in rows)
        {
            t.Cell().Element(Cell).Text(r.Index.ToString());
            t.Cell().Element(Cell).Text(r.Share ?? "");
            t.Cell().Element(Cell).Text(Box(r.R));
            t.Cell().Element(Cell).Text(Box(r.W));
            t.Cell().Element(Cell).Text(Box(r.D));
            t.Cell().Element(Cell).Text(Box(r.Remove));
        }

        int start = rows.Count + 1; // no hardcoded row; if no data, start at 1

        for (int i = start; i <= 6; i++)
        {
            t.Cell().Element(Cell).Text(i.ToString());
            t.Cell().Element(Cell).Text("");
            t.Cell().Element(Cell).Text("");
            t.Cell().Element(Cell).Text("");
            t.Cell().Element(Cell).Text("");
            t.Cell().Element(Cell).Text("");
        }
    });
}
```

- **FormArrangementBlock: five approval panels in one frame**
    - This block shows the approval flow as five equal panels inside a single outer border. It starts with a one-line guide so requestors know the route. Each panel prints a title, then the captured name and date. The layout uses equal columns so it reads like a baton passing from left to right.

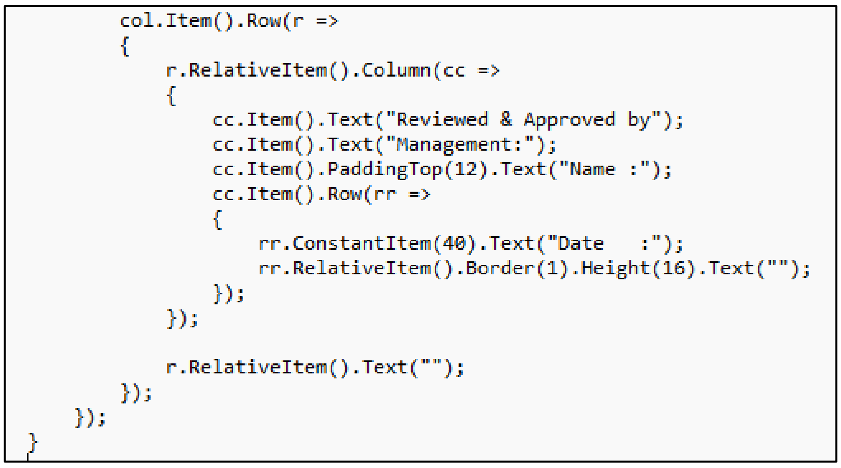


```
void FormArrangementBlock(IContainer c, ItRequestReportModel m)
{
    // tiny helper: draws the vertical borders for each cell so it looks like 5 boxed panels
    IContainer BoxCell(IContainer x, bool isLast) =>
        x.BorderLeft(1)
         .BorderRight(isLast ? 1 : 0)   // last cell closes the right border
         .Padding(6)
         .MinHeight(45);                // keep all equal; adjust to taste

    c.Column(col =>
    {
        // italic guide line
        col.Item()
            .PaddingBottom(4)
            .Text("Form Arrangement: User → HOD → IT HOD → FINANCE HOD → CEO/CFO/COO")
            .Italic();

        // OUTER frame
        col.Item().Border(1).Element(frame =>
        {
            frame.Table(t =>
            {
                // 5 equal columns
                t.ColumnsDefinition(cols =>
                {
                    cols.RelativeColumn(1);
                    cols.RelativeColumn(1);
                    cols.RelativeColumn(1);
                    cols.RelativeColumn(1);
                    cols.RelativeColumn(1);
                });

                // local function to render a boxed panel
                void Panel(int index0to4, string title, string name, string date)
                {
                    bool last = index0to4 == 4;
                    t.Cell().Element(x => BoxCell(x, last)).Column(cc =>
                    {
                        cc.Item().Text(title);

                        // Name / Date rows
                        cc.Item().PaddingTop(8).Row(r =>
                        {
                            r.ConstantItem(40).Text("Name :");
                            r.RelativeItem().Text(string.IsNullOrWhiteSpace(name) ? "" : name);
                        });
                        cc.Item().Row(r =>
                        {
                            r.ConstantItem(40).Text("Date  :");
                            r.RelativeItem().Text(string.IsNullOrWhiteSpace(date) ? "" : date);
                        });
                    });
                }

                // build the 5 panels (plug in your resolved names/dates)
                Panel(0, "Requested by:", m.RequestorName ?? "", F(m.SubmitDate) ?? "");
                Panel(1, "Approved by HOD:", m.HodApprovedBy ?? "", F(m.HodSubmitDate) ?? "");
                Panel(2, "Approved by Group IT HOD:", m.GitHodApprovedBy ?? "", F(m.GitHodSubmitDate) ?? "");
                Panel(3, "Supported by Finance HOD:", m.FinHodApprovedBy ?? "", F(m.FinHodSubmitDate) ?? "");
                Panel(4, "Reviewed & Approved by Management:", m.MgmtApprovedBy ?? "", F(m.MgmtSubmitDate) ?? "");
            });
```

- **Approvals (legacy layout, kept for reference)**
  - This is an earlier approval layout that shows three rows of two panels. You kept it in case you ever want a stacked version instead of the single-row baton. It is not called in `Generate`, but retaining it gives you flexibility without rewriting.



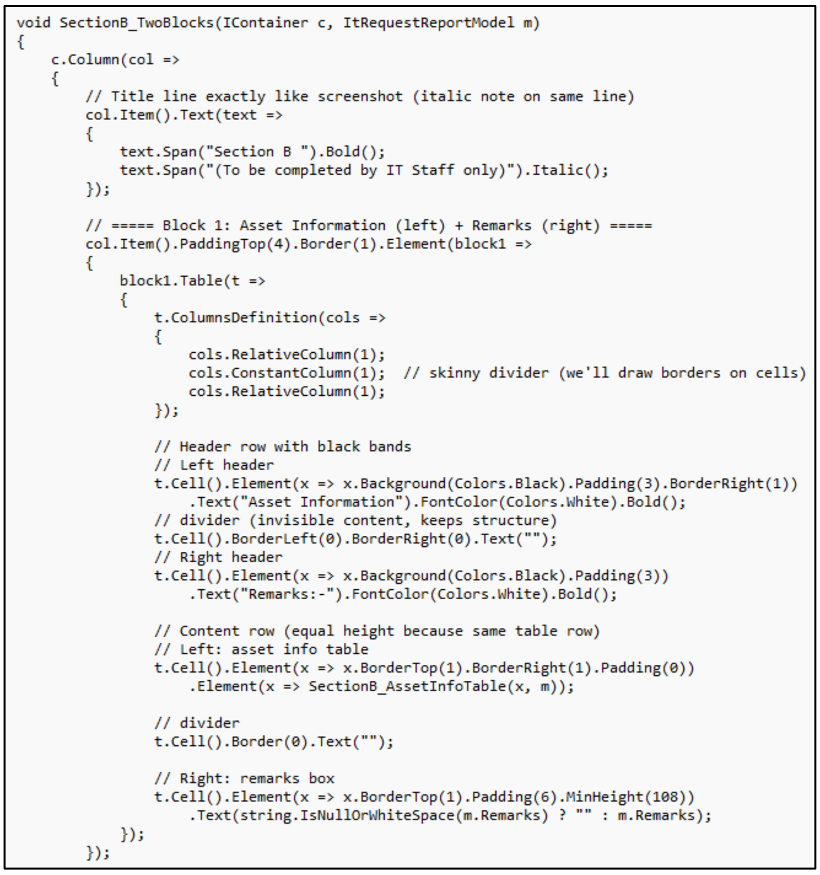
```
void Approvals(IContainer c, ItRequestReportModel m)
{
    c.Column(col =>
    {
        col.Item().Row(r =>
        {
            r.RelativeItem().Column(cc =>
            {
                cc.Item().Text("Requested by:");
                cc.Item().PaddingTop(12).Text("Name :           MANDATORY");
                cc.Item().Row(rr =>
                {
                    rr.ConstantItem(40).Text("Date   :");
                    rr.RelativeItem().Border(1).Height(14).Text(F(m.SubmitDate));
                });
            });

            r.RelativeItem().Column(cc =>
            {
                cc.Item().Text("Approved by HOD:");
                cc.Item().PaddingTop(12).Text("Name :");
                cc.Item().Row(rr =>
                {
                    rr.ConstantItem(40).Text("Date   :");
                    rr.RelativeItem().Border(1).Height(16).Text("");
                });
            });
        });

        col.Item().Row(r =>
        {
            r.RelativeItem().Column(cc =>
            {
                cc.Item().Text("Approved by Group IT HOD:");
                cc.Item().PaddingTop(12).Text("Name :");
                cc.Item().Row(rr =>
                {
                    rr.ConstantItem(40).Text("Date   :");
                    rr.RelativeItem().Border(1).Height(16).Text("");
                });
            });

            r.RelativeItem().Column(cc =>
            {
                cc.Item().Text("Supported by Finance HOD:");
                cc.Item().PaddingTop(12).Text("Name :");
                cc.Item().Row(rr =>
                {
                    rr.ConstantItem(40).Text("Date   :");
                    rr.RelativeItem().Border(1).Height(16).Text("");
                });
            });
        });
```



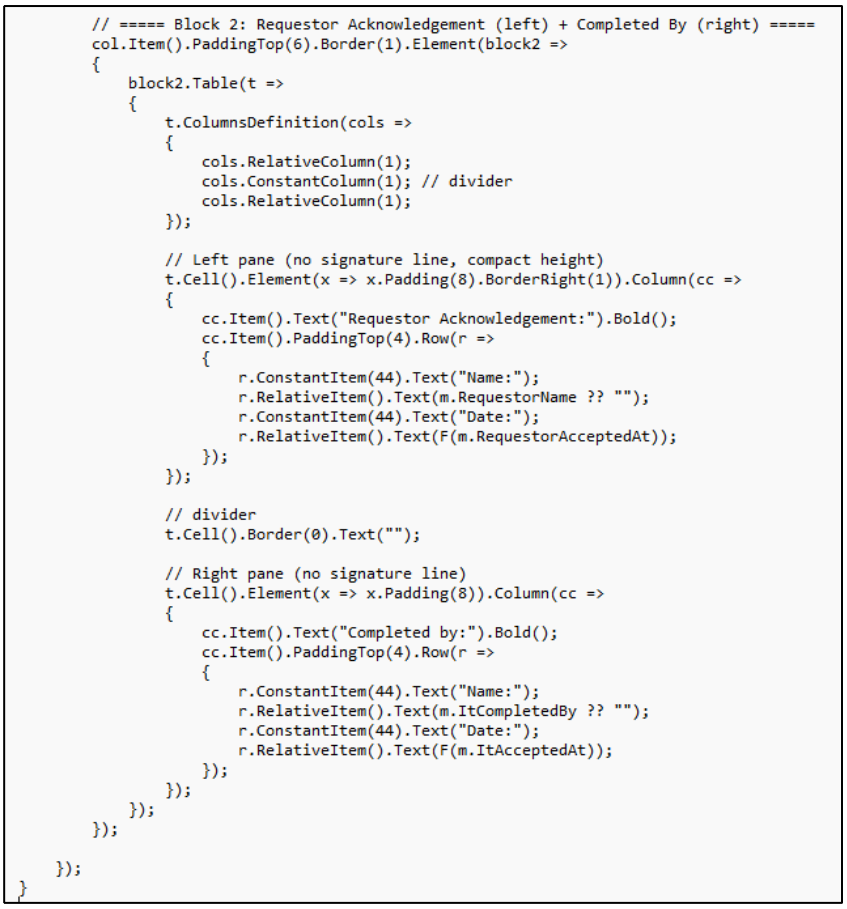
```
        col.Item().Row(r =>
        {
            r.RelativeItem().Column(cc =>
            {
                cc.Item().Text("Reviewed & Approved by");
                cc.Item().Text("Management:");
                cc.Item().PaddingTop(12).Text("Name :");
                cc.Item().Row(rr =>
                {
                    rr.ConstantItem(40).Text("Date   :");
                    rr.RelativeItem().Border(1).Height(16).Text("");
                });
            });

            r.RelativeItem().Text("");
        });
    });
}
```

- **Section B: TwoBlocks with Asset Information, Remarks, Acknowledgements, and Completion**
    - Section B opens with a title that includes a small note indicating it is for IT staff. Block 1 pairs the asset information table on the left with a tall remarks area on the right inside one framed unit. Block 2 shows acknowledgement by

the requestor on the left and the IT completion on the right, both in a compact height that fits the page nicely.



```
void SectionB_TwoBlocks(IContainer c, ItRequestReportModel m)
{
    c.Column(col =>
    {
        // Title line exactly like screenshot (italic note on same line)
        col.Item().Text(text =>
        {
            text.Span("Section B ").Bold();
            text.Span("(To be completed by IT Staff only)").Italic();
        });

        // ===== Block 1: Asset Information (left) + Remarks (right) =====
        col.Item().PaddingTop(4).Border(1).Element(block1 =>
        {
            block1.Table(t =>
            {
                t.ColumnsDefinition(cols =>
                {
                    cols.RelativeColumn(1);
                    cols.ConstantColumn(1);  // skinny divider (we'll draw borders on cells)
                    cols.RelativeColumn(1);
                });

                // Header row with black bands
                // Left header
                t.Cell().Element(x => x.Background(Colors.Black).Padding(3).BorderRight(1))
                    .Text("Asset Information").FontColor(Colors.White).Bold();
                // divider (invisible content, keeps structure)
                t.Cell().BorderLeft(0).BorderRight(0).Text("");
                // Right header
                t.Cell().Element(x => x.Background(Colors.Black).Padding(3))
                    .Text("Remarks:-").FontColor(Colors.White).Bold();

                // Content row (equal height because same table row)
                // Left: asset info table
                t.Cell().Element(x => x.BorderTop(1).BorderRight(1).Padding(0))
                    .Element(x => SectionB_AssetInfoTable(x, m));

                // divider
                t.Cell().Border(0).Text("");

                // Right: remarks box
                t.Cell().Element(x => x.BorderTop(1).Padding(6).MinHeight(108))
                    .Text(string.IsNullOrWhiteSpace(m.Remarks) ? "" : m.Remarks);
            });
        });
```



```
        // ===== Block 2: Requestor Acknowledgement (left) + Completed By (right) =====
        col.Item().PaddingTop(6).Border(1).Element(block2 =>
        {
            block2.Table(t =>
            {
                t.ColumnsDefinition(cols =>
                {
                    cols.RelativeColumn(1);
                    cols.ConstantColumn(1); // divider
                    cols.RelativeColumn(1);
                });

                // Left pane (no signature line, compact height)
                t.Cell().Element(x => x.Padding(8).BorderRight(1)).Column(cc =>
                {
                    cc.Item().Text("Requestor Acknowledgement:").Bold();
                    cc.Item().PaddingTop(4).Row(r =>
                    {
                        r.ConstantItem(44).Text("Name:");
                        r.RelativeItem().Text(m.RequestorName ?? "");
                        r.ConstantItem(44).Text("Date:");
                        r.RelativeItem().Text(F(m.RequestorAcceptedAt));
                    });
                });

                // divider
                t.Cell().Border(0).Text("");

                // Right pane (no signature line)
                t.Cell().Element(x => x.Padding(8)).Column(cc =>
                {
                    cc.Item().Text("Completed by:").Bold();
                    cc.Item().PaddingTop(4).Row(r =>
                    {
                        r.ConstantItem(44).Text("Name:");
                        r.RelativeItem().Text(m.ItCompletedBy ?? "");
                        r.ConstantItem(44).Text("Date:");
                        r.RelativeItem().Text(F(m.ItAcceptedAt));
                    });
                });
            });
        });

    });
}
```

- **SectionB_AssetInfoTable: simple two-column key/value table**
    - This is the left side of Block 1. It prints asset identifiers in a two-column key/value table with a clean inner border. Keeping the borders consistent makes the whole block feel like a single component.

```
void SectionB_AssetInfoTable(IContainer c, ItRequestReportModel m)
{
    c.Table(t =>
    {
        t.ColumnsDefinition(cols =>
        {
            cols.RelativeColumn(2); // label
            cols.RelativeColumn(2); // value box
        });

        void Row(string label, string value)
        {
            t.Cell().BorderBottom(1).Padding(6).Text(label);
            t.Cell().BorderLeft(1).BorderBottom(1).Padding(3).Text(value ?? "");
        }

        // top row gets its own bottom borders; left cell also has right divider
        Row("Asset No:", m.AssetNo);
        Row("Machine ID:", m.MachineId);
        Row("IP Add:", m.IpAddress);
        Row("Wired Mac Add:", m.WiredMac);
        Row("Wi-Fi Mac Add:", m.WifiMac);
        Row("Dial-up Acc:", m.DialupAcc);
    });
}
```