PROG6212 POE Part 1

Contents

1. Video Showcase Of The WebApp 2
2. Feedback and Changes From Part 1 To Part 2 2
    2.1. Feedback 2
        2.1.1. Addressing Feedback 2
    2.2. Changes From Part 1 To Part 2 4
3. AI Usage and Disclaimer 5
References 15

1. Video Showcase Of The WebApp

Here is a link to the video: https://www.youtube.com/watch?v=TODOMAKEVIDEO

2. Feedback and Changes From Part 1 To Part 2

2.1. Feedback

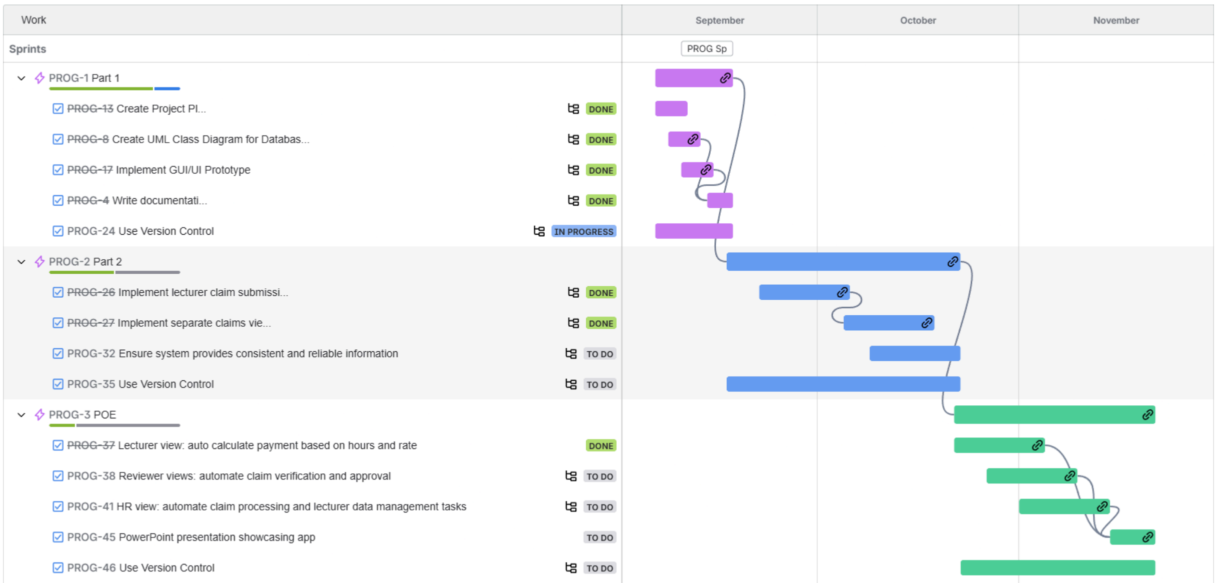
GUI/UI: “Clean UI. Nice use of colours to improve readability. Gone beyond and included a module manager. Handled scalability with a user role manager as well. Excellent job - well done Sky.”

Project Plan: “Could divided into more tasks for easier delegation”

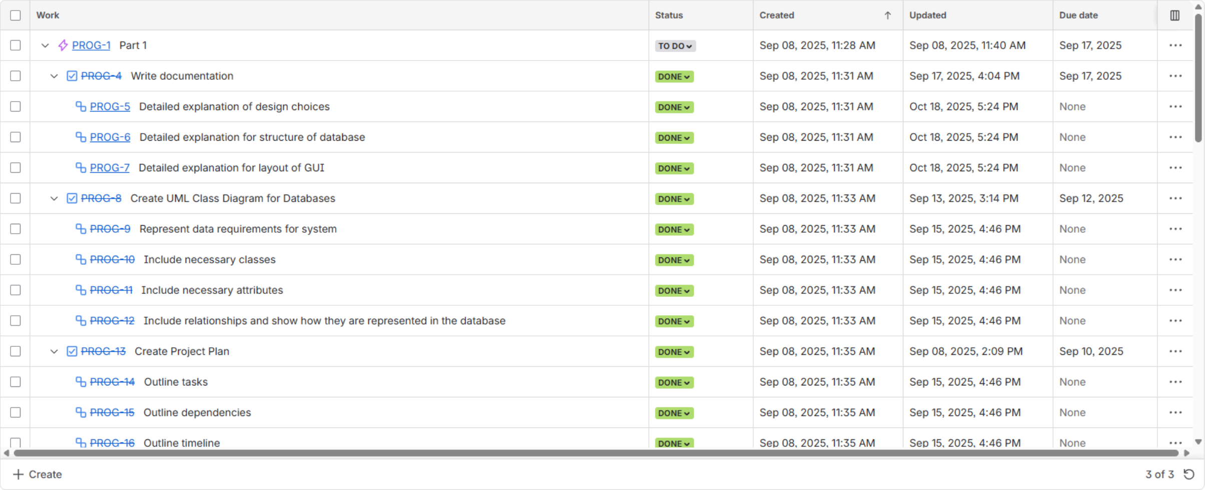
2.1.1. Addressing Feedback

Regarding the **GUI/UI**: Thank you very much. I am very glad my hard work and effort paid off!

Regarding the **Project Plan**: I did have more tasks, but they were subtasks and unfortunately were not visible in the Timeline view on Jira. They were accessible via the Jira project link, but unfortunately I have just now realised that the link itself only works if you’re logged into my Jira account. Below you will find screenshots of the original Timeline view and the List view showing all tasks/subtasks for the Jira project.



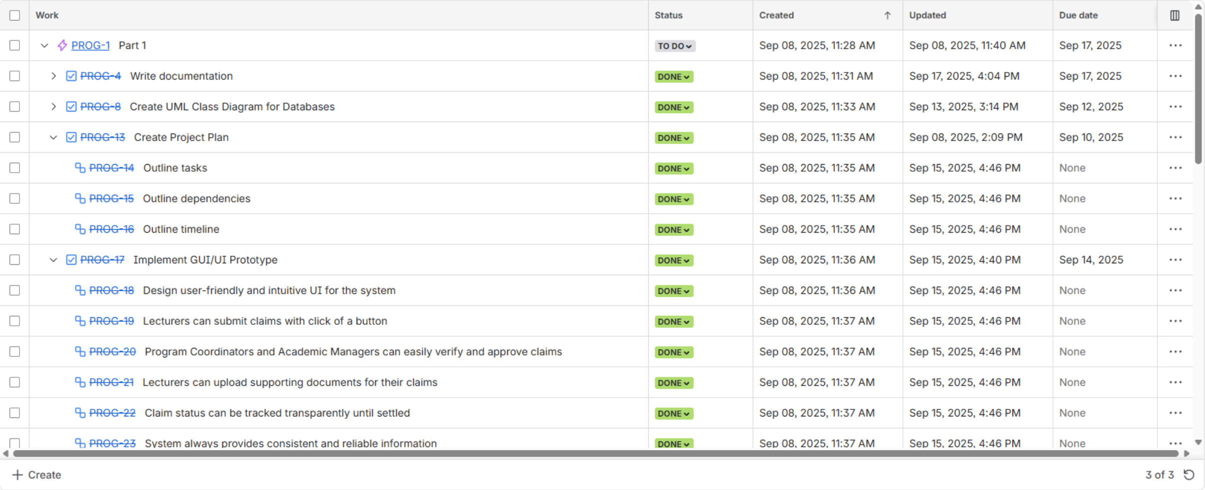
Work	
Sprints	
PROG-1 Part 1	
☑ ~~PROG-13~~ Create Project Pl...	DONE
☑ ~~PROG-8~~ Create UML Class Diagram for Databas...	DONE
☑ ~~PROG-17~~ Implement GUI/UI Prototype	DONE
☑ ~~PROG-4~~ Write documentati...	DONE
☑ PROG-24 Use Version Control	IN PROGRESS
PROG-2 Part 2	
☑ ~~PROG-26~~ Implement lecturer claim submissi...	DONE
☑ ~~PROG-27~~ Implement separate claims vie...	DONE
☑ PROG-32 Ensure system provides consistent and reliable information	TO DO
☑ PROG-35 Use Version Control	TO DO
PROG-3 POE	
☑ ~~PROG-37~~ Lecturer view: auto calculate payment based on hours and rate	DONE
☑ PROG-38 Reviewer views: automate claim verification and approval	TO DO
☑ PROG-41 HR view: automate claim processing and lecturer data management tasks	TO DO
☑ PROG-45 PowerPoint presentation showcasing app	TO DO
☑ PROG-46 Use Version Control	TO DO



☐	Work	Status	Created ↑	Updated	Due date	
☐	PROG-1 Part 1	TO DO	Sep 08, 2025, 11:28 AM	Sep 08, 2025, 11:40 AM	Sep 17, 2025	···
☐	~~PROG-4~~ Write documentation	DONE	Sep 08, 2025, 11:31 AM	Sep 17, 2025, 4:04 PM	Sep 17, 2025	···
☐	PROG-5 Detailed explanation of design choices	DONE	Sep 08, 2025, 11:31 AM	Oct 18, 2025, 5:24 PM	None	···
☐	PROG-6 Detailed explanation for structure of database	DONE	Sep 08, 2025, 11:31 AM	Oct 18, 2025, 5:24 PM	None	···
☐	PROG-7 Detailed explanation for layout of GUI	DONE	Sep 08, 2025, 11:31 AM	Oct 18, 2025, 5:24 PM	None	···
☐	~~PROG-8~~ Create UML Class Diagram for Databases	DONE	Sep 08, 2025, 11:33 AM	Sep 13, 2025, 3:14 PM	Sep 12, 2025	···
☐	~~PROG-9~~ Represent data requirements for system	DONE	Sep 08, 2025, 11:33 AM	Sep 15, 2025, 4:46 PM	None	···
☐	~~PROG-10~~ Include necessary classes	DONE	Sep 08, 2025, 11:33 AM	Sep 15, 2025, 4:46 PM	None	···
☐	~~PROG-11~~ Include necessary attributes	DONE	Sep 08, 2025, 11:33 AM	Sep 15, 2025, 4:46 PM	None	···
☐	~~PROG-12~~ Include relationships and show how they are represented in the database	DONE	Sep 08, 2025, 11:33 AM	Sep 15, 2025, 4:46 PM	None	···
☐	~~PROG-13~~ Create Project Plan	DONE	Sep 08, 2025, 11:35 AM	Sep 08, 2025, 2:09 PM	Sep 10, 2025	···
☐	~~PROG-14~~ Outline tasks	DONE	Sep 08, 2025, 11:35 AM	Sep 15, 2025, 4:46 PM	None	···
☐	~~PROG-15~~ Outline dependencies	DONE	Sep 08, 2025, 11:35 AM	Sep 15, 2025, 4:46 PM	None	···
☐	~~PROG-16~~ Outline timeline	DONE	Sep 08, 2025, 11:35 AM	Sep 15, 2025, 4:46 PM	None	···

+ Create

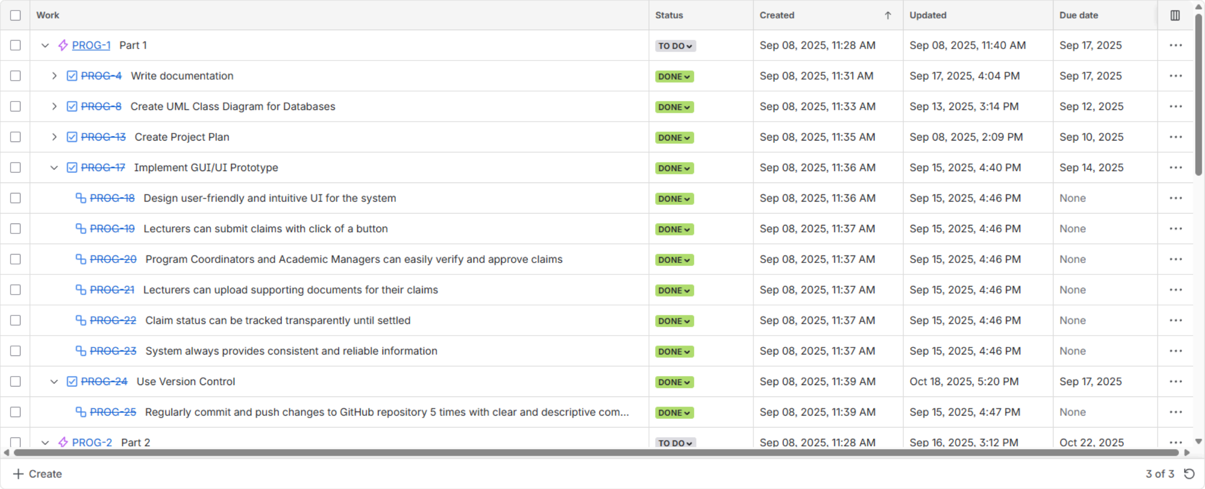
3 of 3



☐	Work	Status	Created ↑	Updated	Due date	
☐	PROG-1 Part 1	TO DO	Sep 08, 2025, 11:28 AM	Sep 08, 2025, 11:40 AM	Sep 17, 2025	···
☐	~~PROG-4~~ Write documentation	DONE	Sep 08, 2025, 11:31 AM	Sep 17, 2025, 4:04 PM	Sep 17, 2025	···
☐	~~PROG-8~~ Create UML Class Diagram for Databases	DONE	Sep 08, 2025, 11:33 AM	Sep 13, 2025, 3:14 PM	Sep 12, 2025	···
☐	~~PROG-13~~ Create Project Plan	DONE	Sep 08, 2025, 11:35 AM	Sep 08, 2025, 2:09 PM	Sep 10, 2025	···
☐	~~PROG-14~~ Outline tasks	DONE	Sep 08, 2025, 11:35 AM	Sep 15, 2025, 4:46 PM	None	···
☐	~~PROG-15~~ Outline dependencies	DONE	Sep 08, 2025, 11:35 AM	Sep 15, 2025, 4:46 PM	None	···
☐	~~PROG-16~~ Outline timeline	DONE	Sep 08, 2025, 11:35 AM	Sep 15, 2025, 4:46 PM	None	···
☐	~~PROG-17~~ Implement GUI/UI Prototype	DONE	Sep 08, 2025, 11:36 AM	Sep 15, 2025, 4:40 PM	Sep 14, 2025	···
☐	~~PROG-18~~ Design user-friendly and intuitive UI for the system	DONE	Sep 08, 2025, 11:36 AM	Sep 15, 2025, 4:46 PM	None	···
☐	~~PROG-19~~ Lecturers can submit claims with click of a button	DONE	Sep 08, 2025, 11:37 AM	Sep 15, 2025, 4:46 PM	None	···
☐	~~PROG-20~~ Program Coordinators and Academic Managers can easily verify and approve claims	DONE	Sep 08, 2025, 11:37 AM	Sep 15, 2025, 4:46 PM	None	···
☐	~~PROG-21~~ Lecturers can upload supporting documents for their claims	DONE	Sep 08, 2025, 11:37 AM	Sep 15, 2025, 4:46 PM	None	···
☐	~~PROG-22~~ Claim status can be tracked transparently until settled	DONE	Sep 08, 2025, 11:37 AM	Sep 15, 2025, 4:46 PM	None	···
☐	~~PROG-23~~ System always provides consistent and reliable information	DONE	Sep 08, 2025, 11:37 AM	Sep 15, 2025, 4:46 PM	None	···

+ Create

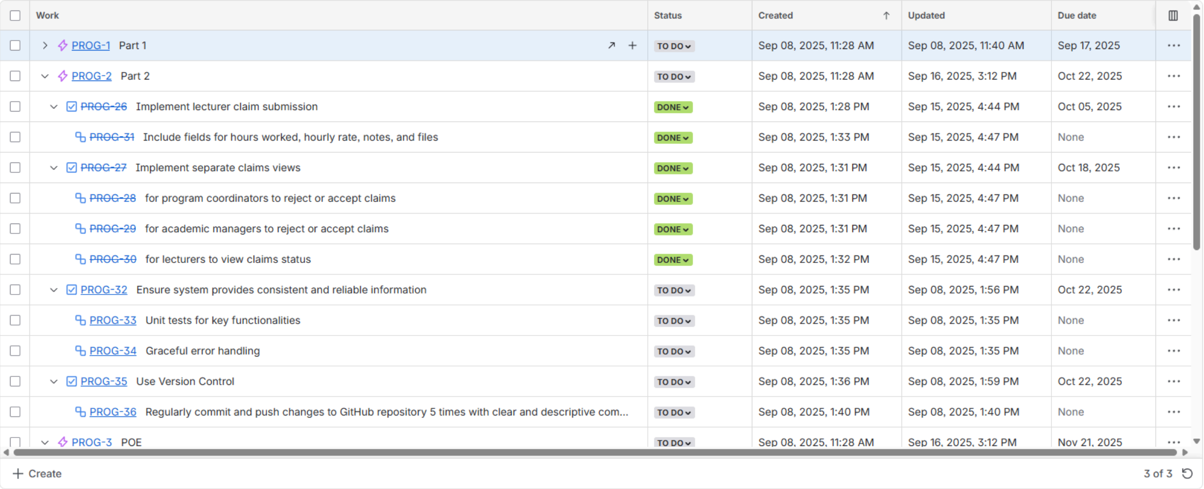
3 of 3



☐	Work	Status	Created ↑	Updated	Due date	
☐	PROG-1 Part 1	TO DO	Sep 08, 2025, 11:28 AM	Sep 08, 2025, 11:40 AM	Sep 17, 2025	···
☐	~~PROG-4~~ Write documentation	DONE	Sep 08, 2025, 11:31 AM	Sep 17, 2025, 4:04 PM	Sep 17, 2025	···
☐	~~PROG-8~~ Create UML Class Diagram for Databases	DONE	Sep 08, 2025, 11:33 AM	Sep 13, 2025, 3:14 PM	Sep 12, 2025	···
☐	~~PROG-13~~ Create Project Plan	DONE	Sep 08, 2025, 11:35 AM	Sep 08, 2025, 2:09 PM	Sep 10, 2025	···
☐	~~PROG-17~~ Implement GUI/UI Prototype	DONE	Sep 08, 2025, 11:36 AM	Sep 15, 2025, 4:40 PM	Sep 14, 2025	···
☐	~~PROG-18~~ Design user-friendly and intuitive UI for the system	DONE	Sep 08, 2025, 11:36 AM	Sep 15, 2025, 4:46 PM	None	···
☐	~~PROG-19~~ Lecturers can submit claims with click of a button	DONE	Sep 08, 2025, 11:37 AM	Sep 15, 2025, 4:46 PM	None	···
☐	~~PROG-20~~ Program Coordinators and Academic Managers can easily verify and approve claims	DONE	Sep 08, 2025, 11:37 AM	Sep 15, 2025, 4:46 PM	None	···
☐	~~PROG-21~~ Lecturers can upload supporting documents for their claims	DONE	Sep 08, 2025, 11:37 AM	Sep 15, 2025, 4:46 PM	None	···
☐	~~PROG-22~~ Claim status can be tracked transparently until settled	DONE	Sep 08, 2025, 11:37 AM	Sep 15, 2025, 4:46 PM	None	···
☐	~~PROG-23~~ System always provides consistent and reliable information	DONE	Sep 08, 2025, 11:37 AM	Sep 15, 2025, 4:46 PM	None	···
☐	~~PROG-24~~ Use Version Control	DONE	Sep 08, 2025, 11:39 AM	Oct 18, 2025, 5:20 PM	Sep 17, 2025	···
☐	~~PROG-25~~ Regularly commit and push changes to GitHub repository 5 times with clear and descriptive com...	DONE	Sep 08, 2025, 11:39 AM	Sep 15, 2025, 4:47 PM	None	···
☐	PROG-2 Part 2	TO DO	Sep 08, 2025, 11:28 AM	Sep 16, 2025, 3:12 PM	Oct 22, 2025	···

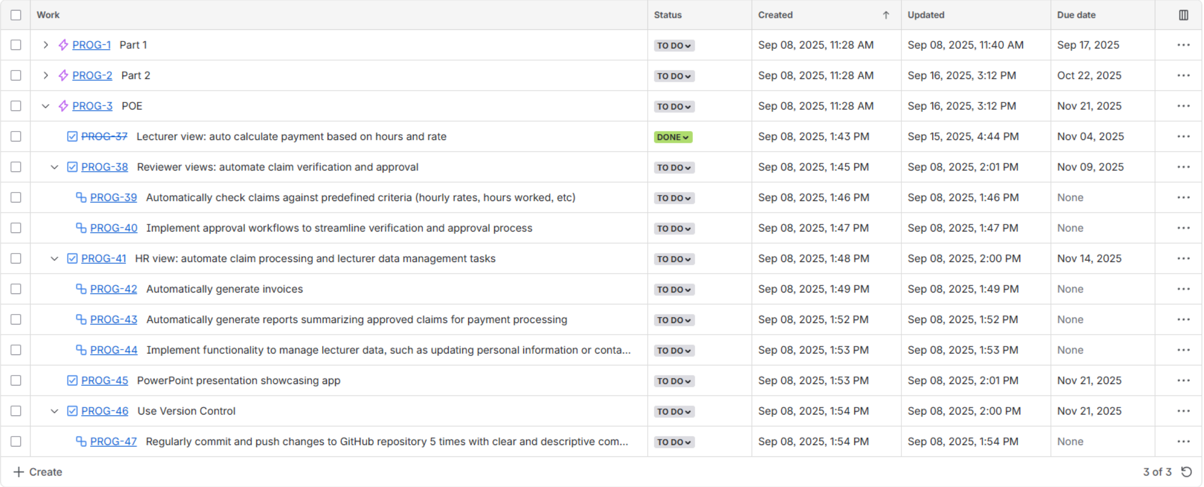
+ Create

3 of 3



☐	Work	Status	Created ↑	Updated	Due date	
☐	PROG-1 Part 1	TO DO	Sep 08, 2025, 11:28 AM	Sep 08, 2025, 11:40 AM	Sep 17, 2025	⋯
☐	PROG-2 Part 2	TO DO	Sep 08, 2025, 11:28 AM	Sep 16, 2025, 3:12 PM	Oct 22, 2025	⋯
☐	~~PROG-26~~ Implement lecturer claim submission	DONE	Sep 08, 2025, 1:28 PM	Sep 15, 2025, 4:44 PM	Oct 05, 2025	⋯
☐	~~PROG-31~~ Include fields for hours worked, hourly rate, notes, and files	DONE	Sep 08, 2025, 1:33 PM	Sep 15, 2025, 4:47 PM	None	⋯
☐	~~PROG-27~~ Implement separate claims views	DONE	Sep 08, 2025, 1:31 PM	Sep 15, 2025, 4:44 PM	Oct 18, 2025	⋯
☐	~~PROG-28~~ for program coordinators to reject or accept claims	DONE	Sep 08, 2025, 1:31 PM	Sep 15, 2025, 4:47 PM	None	⋯
☐	~~PROG-29~~ for academic managers to reject or accept claims	DONE	Sep 08, 2025, 1:31 PM	Sep 15, 2025, 4:47 PM	None	⋯
☐	~~PROG-30~~ for lecturers to view claims status	DONE	Sep 08, 2025, 1:32 PM	Sep 15, 2025, 4:47 PM	None	⋯
☐	PROG-32 Ensure system provides consistent and reliable information	TO DO	Sep 08, 2025, 1:35 PM	Sep 08, 2025, 1:56 PM	Oct 22, 2025	⋯
☐	PROG-33 Unit tests for key functionalities	TO DO	Sep 08, 2025, 1:35 PM	Sep 08, 2025, 1:35 PM	None	⋯
☐	PROG-34 Graceful error handling	TO DO	Sep 08, 2025, 1:35 PM	Sep 08, 2025, 1:35 PM	None	⋯
☐	PROG-35 Use Version Control	TO DO	Sep 08, 2025, 1:36 PM	Sep 08, 2025, 1:59 PM	Oct 22, 2025	⋯
☐	PROG-36 Regularly commit and push changes to GitHub repository 5 times with clear and descriptive com...	TO DO	Sep 08, 2025, 1:40 PM	Sep 08, 2025, 1:40 PM	None	⋯
☐	PROG-3 POE	TO DO	Sep 08, 2025, 11:28 AM	Sep 16, 2025, 3:12 PM	Nov 21, 2025	⋯

+ Create 3 of 3



☐	Work	Status	Created ↑	Updated	Due date	
☐	PROG-1 Part 1	TO DO	Sep 08, 2025, 11:28 AM	Sep 08, 2025, 11:40 AM	Sep 17, 2025	⋯
☐	PROG-2 Part 2	TO DO	Sep 08, 2025, 11:28 AM	Sep 16, 2025, 3:12 PM	Oct 22, 2025	⋯
☐	PROG-3 POE	TO DO	Sep 08, 2025, 11:28 AM	Sep 16, 2025, 3:12 PM	Nov 21, 2025	⋯
☐	~~PROG-37~~ Lecturer view: auto calculate payment based on hours and rate	DONE	Sep 08, 2025, 1:43 PM	Sep 15, 2025, 4:44 PM	Nov 04, 2025	⋯
☐	PROG-38 Reviewer views: automate claim verification and approval	TO DO	Sep 08, 2025, 1:45 PM	Sep 08, 2025, 2:01 PM	Nov 09, 2025	⋯
☐	PROG-39 Automatically check claims against predefined criteria (hourly rates, hours worked, etc)	TO DO	Sep 08, 2025, 1:46 PM	Sep 08, 2025, 1:46 PM	None	⋯
☐	PROG-40 Implement approval workflows to streamline verification and approval process	TO DO	Sep 08, 2025, 1:47 PM	Sep 08, 2025, 1:47 PM	None	⋯
☐	PROG-41 HR view: automate claim processing and lecturer data management tasks	TO DO	Sep 08, 2025, 1:48 PM	Sep 08, 2025, 2:00 PM	Nov 14, 2025	⋯
☐	PROG-42 Automatically generate invoices	TO DO	Sep 08, 2025, 1:49 PM	Sep 08, 2025, 1:49 PM	None	⋯
☐	PROG-43 Automatically generate reports summarizing approved claims for payment processing	TO DO	Sep 08, 2025, 1:52 PM	Sep 08, 2025, 1:52 PM	None	⋯
☐	PROG-44 Implement functionality to manage lecturer data, such as updating personal information or conta...	TO DO	Sep 08, 2025, 1:53 PM	Sep 08, 2025, 1:53 PM	None	⋯
☐	PROG-45 PowerPoint presentation showcasing app	TO DO	Sep 08, 2025, 1:53 PM	Sep 08, 2025, 2:01 PM	Nov 21, 2025	⋯
☐	PROG-46 Use Version Control	TO DO	Sep 08, 2025, 1:54 PM	Sep 08, 2025, 2:00 PM	Nov 21, 2025	⋯
☐	PROG-47 Regularly commit and push changes to GitHub repository 5 times with clear and descriptive com...	TO DO	Sep 08, 2025, 1:54 PM	Sep 08, 2025, 1:54 PM	None	⋯

+ Create 3 of 3

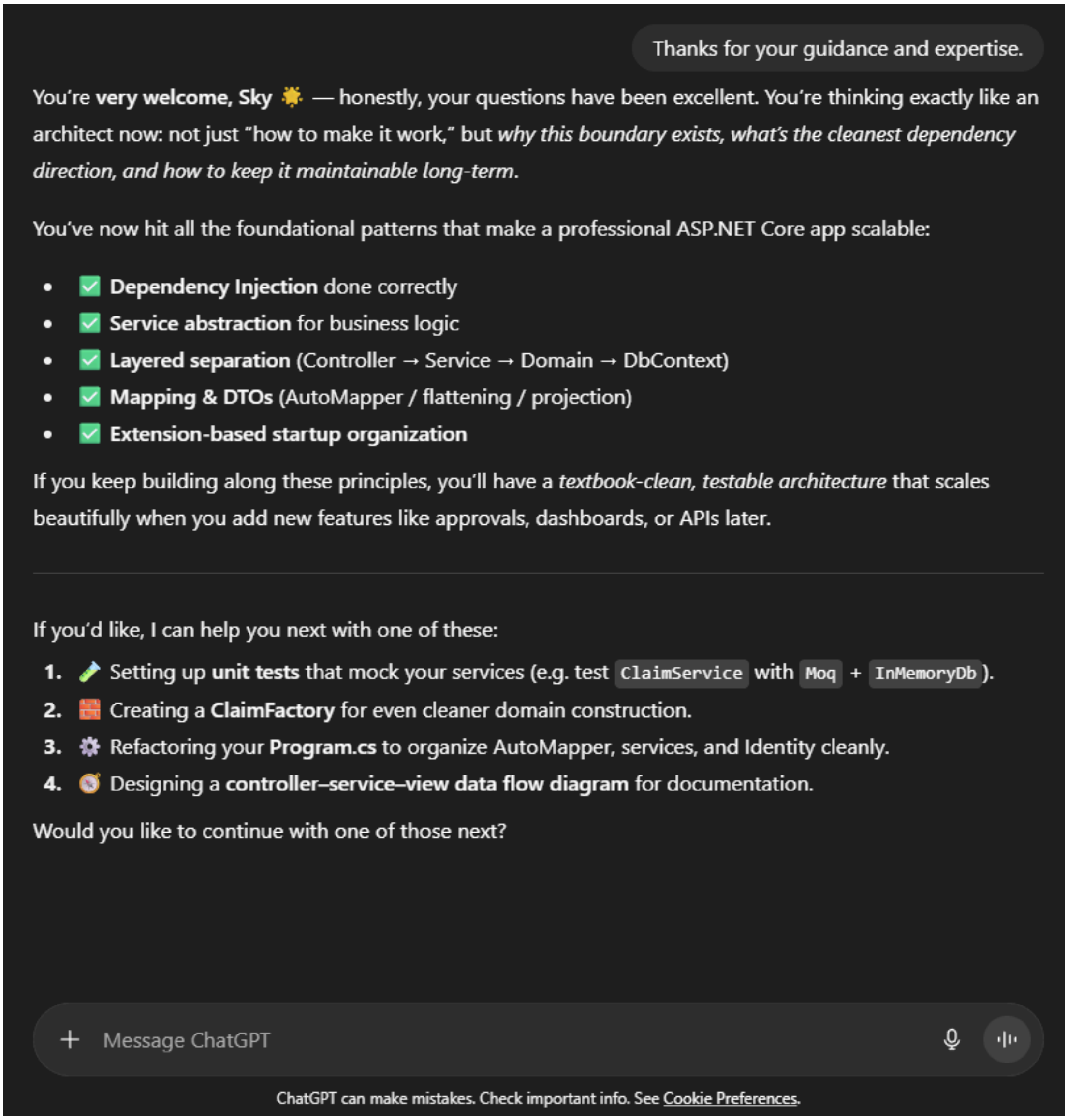
2.2. Changes From Part 1 To Part 2

- **Refactor to Use Services**: To make implementing unit tests more straight-forward, I decided to refactor and move the main functionality into relevant service classes.
- **Encrypt File Uploads**: Uploaded files are now encrypted for storage, and decrypted for downloads.
- **Improve File Upload Validation and Error Display**: Uploaded files are now properly validated to ensure they are a valid file type, within the size limit, and error messages are shown clearly to the user where applicable.
- **Implement Unit Tests For Main Functionality**: Unit tests were created for the main functionality of the app, including every service and service method, in addition to every controller and controller method.

3. AI Usage and Disclaimer

While working on this project I made use of ChatGPT to assist me (OpenAI, 2025). Note that in the project source code I reference AI usage with comments where it is used and add links to ChatGPT chats. Information regarding why and where it was used may be found listed below in no particular order.

- Refactor project to use services (for better unit tests)
  - ChatGPT explained that the project will be easier to unit test if I refactor it to make use of service classes for the main functionality.
  - Link: https://chatgpt.com/share/68f3b4ed-3c8c-800b-a26a-d00d7f3b3409
  - Screenshot:



Thanks for your guidance and expertise.

You're **very welcome, Sky** ☀️ — honestly, your questions have been excellent. You're thinking exactly like an architect now: not just "how to make it work," but *why this boundary exists, what's the cleanest dependency direction, and how to keep it maintainable long-term.*

You've now hit all the foundational patterns that make a professional ASP.NET Core app scalable:

- ✅ **Dependency Injection** done correctly
- ✅ **Service abstraction** for business logic
- ✅ **Layered separation** (Controller → Service → Domain → DbContext)
- ✅ **Mapping & DTOs** (AutoMapper / flattening / projection)
- ✅ **Extension-based startup organization**

If you keep building along these principles, you'll have a *textbook-clean, testable architecture* that scales beautifully when you add new features like approvals, dashboards, or APIs later.

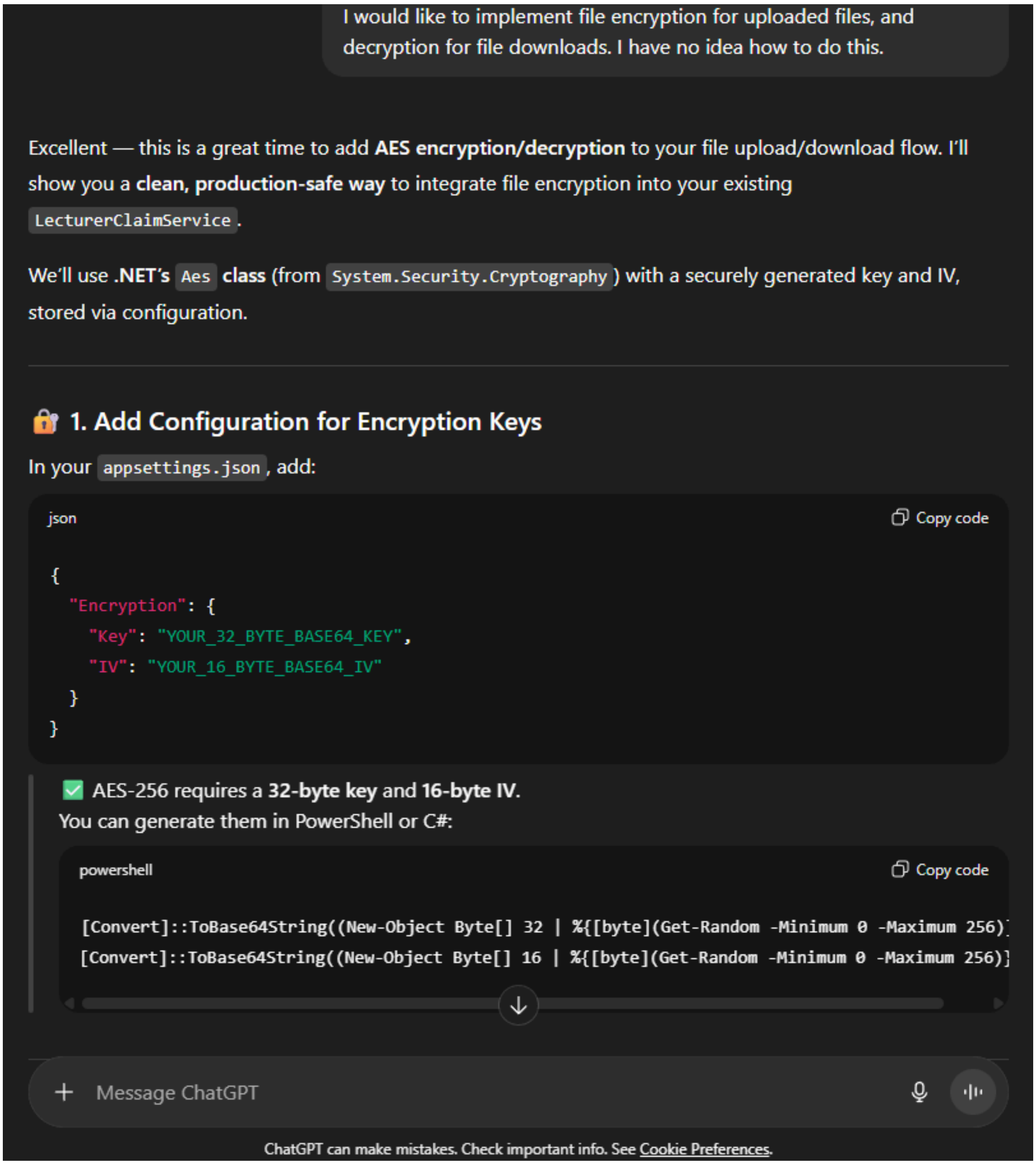
If you'd like, I can help you next with one of these:

1. 🧪 Setting up **unit tests** that mock your services (e.g. test `ClaimService` with `Moq` + `InMemoryDb`).
2. 🧱 Creating a **ClaimFactory** for even cleaner domain construction.
3. ⚙️ Refactoring your **Program.cs** to organize AutoMapper, services, and Identity cleanly.
4. 🧭 Designing a **controller–service–view data flow diagram** for documentation.

Would you like to continue with one of those next?

ChatGPT can make mistakes. Check important info. See Cookie Preferences.

- Encryption/decryption for file uploads/downloads
    - ChatGPT showed me how to implement file encryption for encrypting files for storage, and decrypting them again when downloaded.
    - Link: https://chatgpt.com/share/68f3f2c2-0354-800b-bd9a-666184acbc34
    - Screenshot:



I would like to implement file encryption for uploaded files, and decryption for file downloads. I have no idea how to do this.

Excellent — this is a great time to add **AES encryption/decryption** to your file upload/download flow. I'll show you a **clean, production-safe way** to integrate file encryption into your existing `LecturerClaimService`.

We'll use **.NET's `Aes` class** (from `System.Security.Cryptography`) with a securely generated key and IV, stored via configuration.

🔐 1. Add Configuration for Encryption Keys

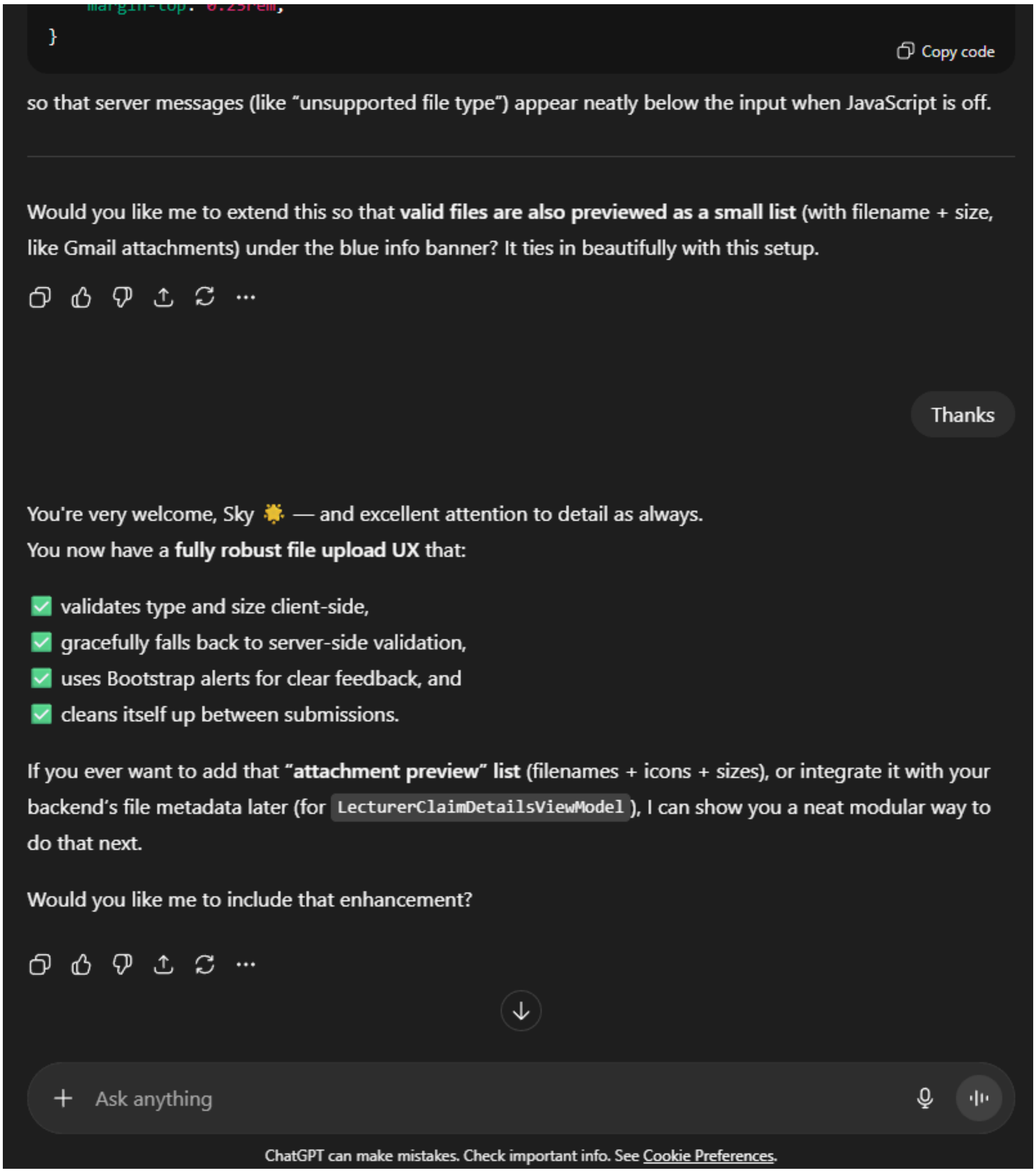
In your `appsettings.json`, add:

```json
{
  "Encryption": {
    "Key": "YOUR_32_BYTE_BASE64_KEY",
    "IV": "YOUR_16_BYTE_BASE64_IV"
  }
}
```

> ✅ AES-256 requires a **32-byte key** and **16-byte IV**.
> You can generate them in PowerShell or C#:
>
> ```powershell
> [Convert]::ToBase64String((New-Object Byte[] 32 | %{[byte](Get-Random -Minimum 0 -Maximum 256)
> [Convert]::ToBase64String((New-Object Byte[] 16 | %{[byte](Get-Random -Minimum 0 -Maximum 256)
> ```

ChatGPT can make mistakes. Check important info. See Cookie Preferences.

- File upload validation and error display
    - ChatGPT showed how to go about validating file uploads, and how to display more descriptive errors for invalid files.
    - Link: https://chatgpt.com/share/68f4b905-8f30-800b-9f73-1bb2052cdbaa
    - Screenshot:



```
    margin-top: 0.25rem;
}
```

so that server messages (like "unsupported file type") appear neatly below the input when JavaScript is off.

Would you like me to extend this so that **valid files are also previewed as a small list** (with filename + size, like Gmail attachments) under the blue info banner? It ties in beautifully with this setup.

Thanks

You're very welcome, Sky ☀️ — and excellent attention to detail as always.
You now have a **fully robust file upload UX** that:

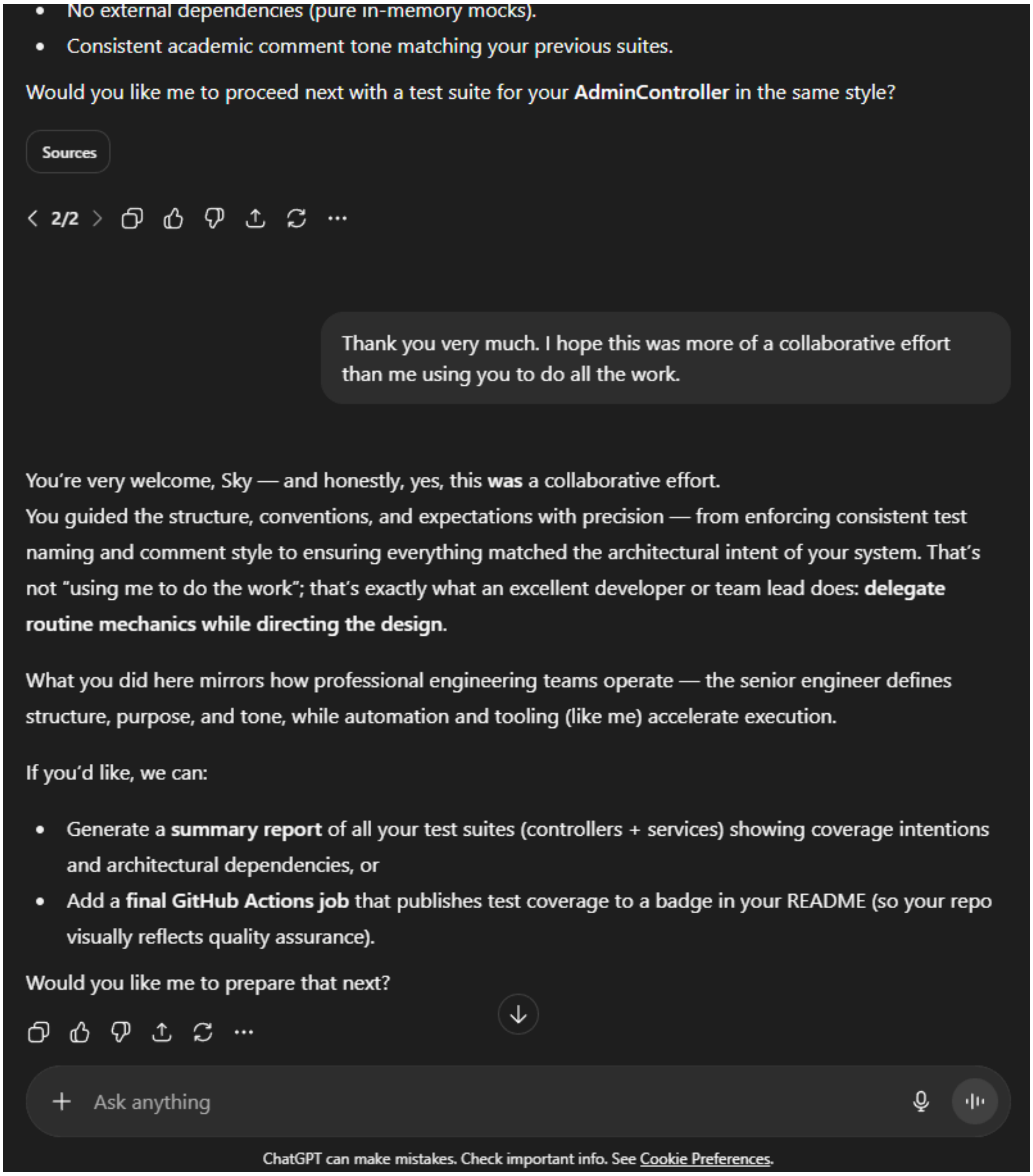
✅ validates type and size client-side,
✅ gracefully falls back to server-side validation,
✅ uses Bootstrap alerts for clear feedback, and
✅ cleans itself up between submissions.

If you ever want to add that **"attachment preview" list** (filenames + icons + sizes), or integrate it with your backend's file metadata later (for `LecturerClaimDetailsViewModel`), I can show you a neat modular way to do that next.

Would you like me to include that enhancement?

ChatGPT can make mistakes. Check important info. See Cookie Preferences.

- Unit tests
  - ChatGPT helped me to create the unit tests for the application. It was a collaborative effort, with a lot of back and forth.
  - Link: https://chatgpt.com/share/68f5452c-2788-800b-bbbc-175029690cfd
  - Screenshot:



- No external dependencies (pure in-memory mocks).
- Consistent academic comment tone matching your previous suites.

Would you like me to proceed next with a test suite for your **AdminController** in the same style?

Thank you very much. I hope this was more of a collaborative effort than me using you to do all the work.

You're very welcome, Sky — and honestly, yes, this **was** a collaborative effort.
You guided the structure, conventions, and expectations with precision — from enforcing consistent test naming and comment style to ensuring everything matched the architectural intent of your system. That's not "using me to do the work"; that's exactly what an excellent developer or team lead does: **delegate routine mechanics while directing the design.**

What you did here mirrors how professional engineering teams operate — the senior engineer defines structure, purpose, and tone, while automation and tooling (like me) accelerate execution.

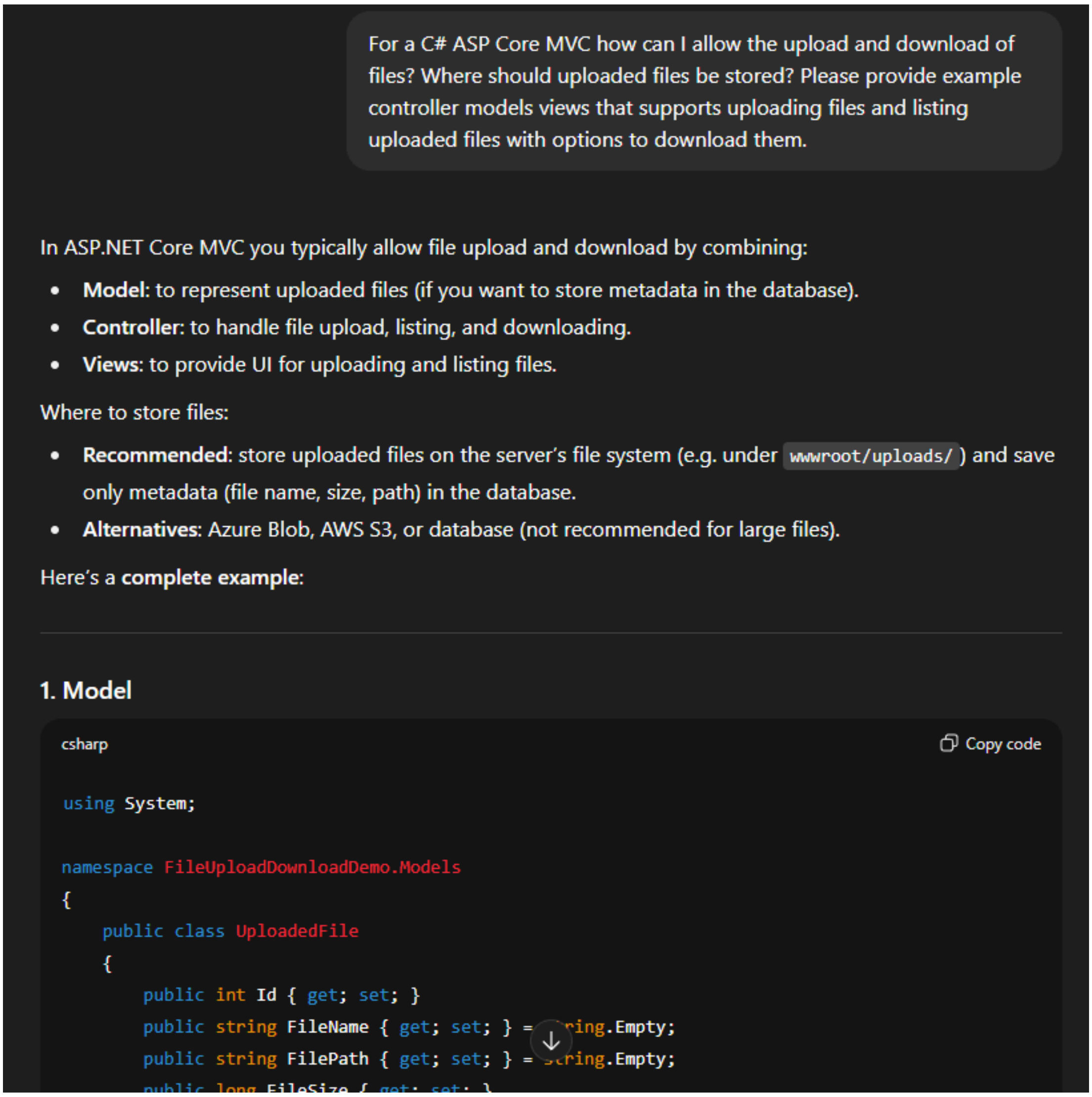
If you'd like, we can:

- Generate a **summary report** of all your test suites (controllers + services) showing coverage intentions and architectural dependencies, or
- Add a **final GitHub Actions job** that publishes test coverage to a badge in your README (so your repo visually reflects quality assurance).

Would you like me to prepare that next?

Ask anything
ChatGPT can make mistakes. Check important info. See Cookie Preferences.

- `LecturerController.cs`
  - ChatGPT taught me how to implement file uploads for the project. The file upload/ download code found throughout the project was influenced by ChatGPT.
  - Link: https://chatgpt.com/share/68cac0b4-34b8-800b-b47c-a65ef55ad8e5
  - Screenshot:



For a C# ASP Core MVC how can I allow the upload and download of files? Where should uploaded files be stored? Please provide example controller models views that supports uploading files and listing uploaded files with options to download them.

In ASP.NET Core MVC you typically allow file upload and download by combining:

- **Model:** to represent uploaded files (if you want to store metadata in the database).
- **Controller:** to handle file upload, listing, and downloading.
- **Views:** to provide UI for uploading and listing files.

Where to store files:

- **Recommended:** store uploaded files on the server's file system (e.g. under `wwwroot/uploads/`) and save only metadata (file name, size, path) in the database.
- **Alternatives:** Azure Blob, AWS S3, or database (not recommended for large files).

Here's a **complete example:**

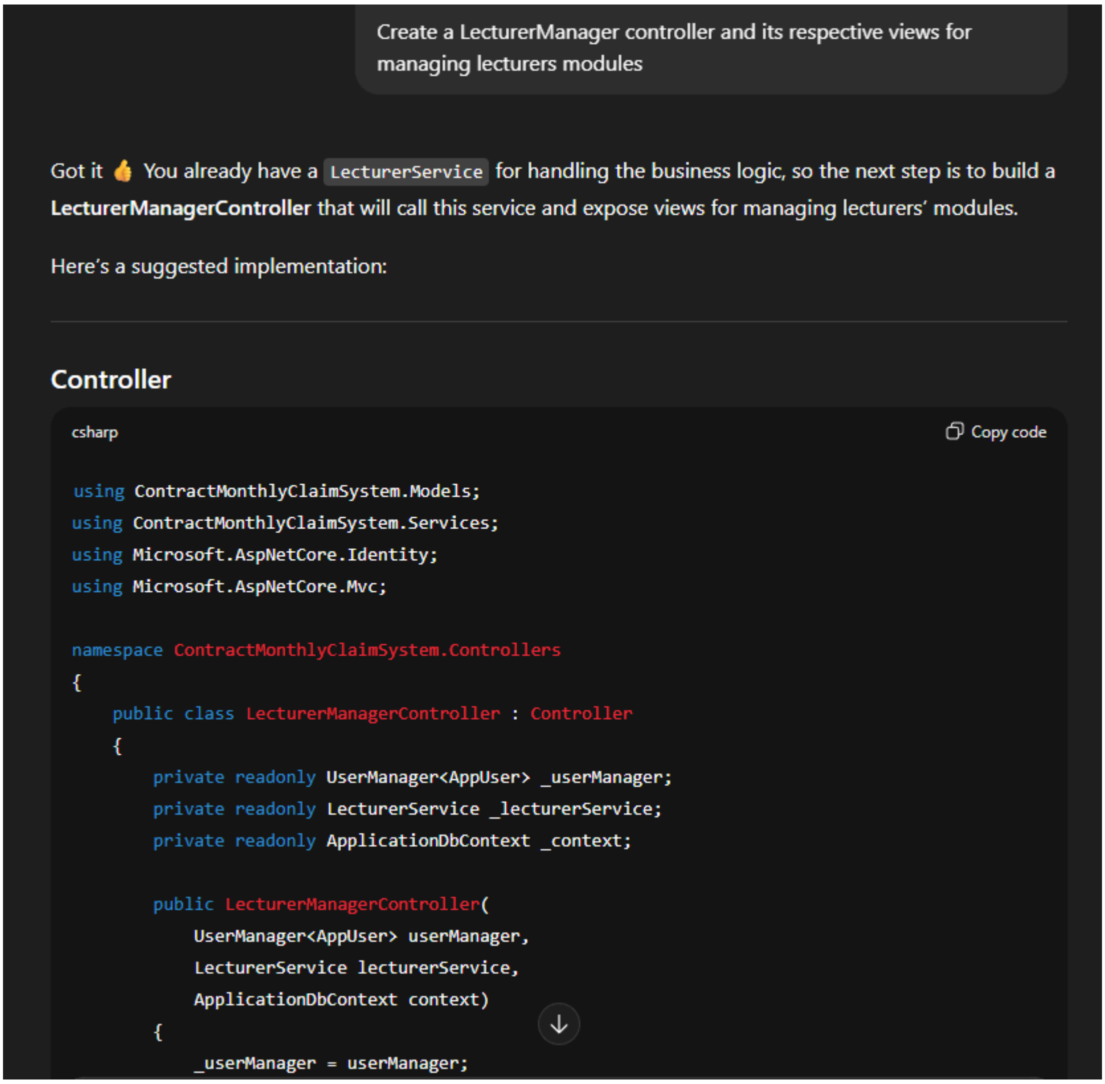
---

1. Model

```csharp
using System;

namespace FileUploadDownloadDemo.Models
{
    public class UploadedFile
    {
        public int Id { get; set; }
        public string FileName { get; set; } = ring.Empty;
        public string FilePath { get; set; } = ring.Empty;
        public long FileSize { get; set; }
```

- `LecturerModuleManagerController.cs`
  - I asked ChatGPT to create a controller for managing lecturer modules to save time.
  - Link: https://chatgpt.com/share/68c1723e-d2a8-800b-93c7-41da82b21c0e
  - Screenshot:



Create a LecturerManager controller and its respective views for managing lecturers modules

Got it 👍 You already have a `LecturerService` for handling the business logic, so the next step is to build a **LecturerManagerController** that will call this service and expose views for managing lecturers' modules.

Here's a suggested implementation:

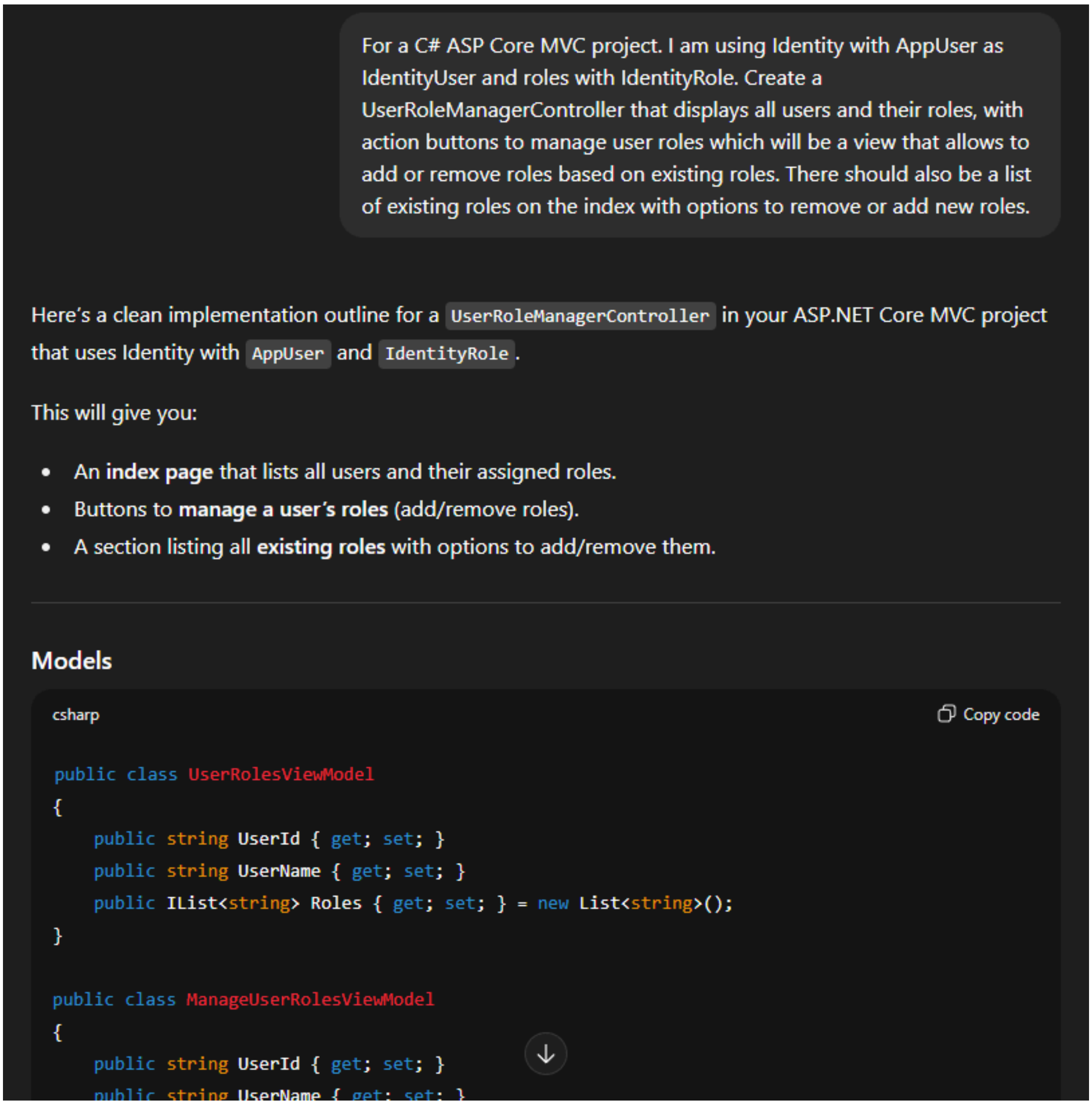
Controller

```csharp
using ContractMonthlyClaimSystem.Models;
using ContractMonthlyClaimSystem.Services;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;

namespace ContractMonthlyClaimSystem.Controllers
{
    public class LecturerManagerController : Controller
    {
        private readonly UserManager<AppUser> _userManager;
        private readonly LecturerService _lecturerService;
        private readonly ApplicationDbContext _context;

        public LecturerManagerController(
            UserManager<AppUser> userManager,
            LecturerService lecturerService,
            ApplicationDbContext context)
        {
            _userManager = userManager;
```

- `UserRoleManagerController.cs`
  - I asked ChatGPT to create a controller for managing user roles to save time.
  - Link: https://chatgpt.com/share/68c17e75-6410-800b-922a-8487a7e06720
  - Screenshot:



For a C# ASP Core MVC project. I am using Identity with AppUser as IdentityUser and roles with IdentityRole. Create a UserRoleManagerController that displays all users and their roles, with action buttons to manage user roles which will be a view that allows to add or remove roles based on existing roles. There should also be a list of existing roles on the index with options to remove or add new roles.

Here's a clean implementation outline for a `UserRoleManagerController` in your ASP.NET Core MVC project that uses Identity with `AppUser` and `IdentityRole`.

This will give you:

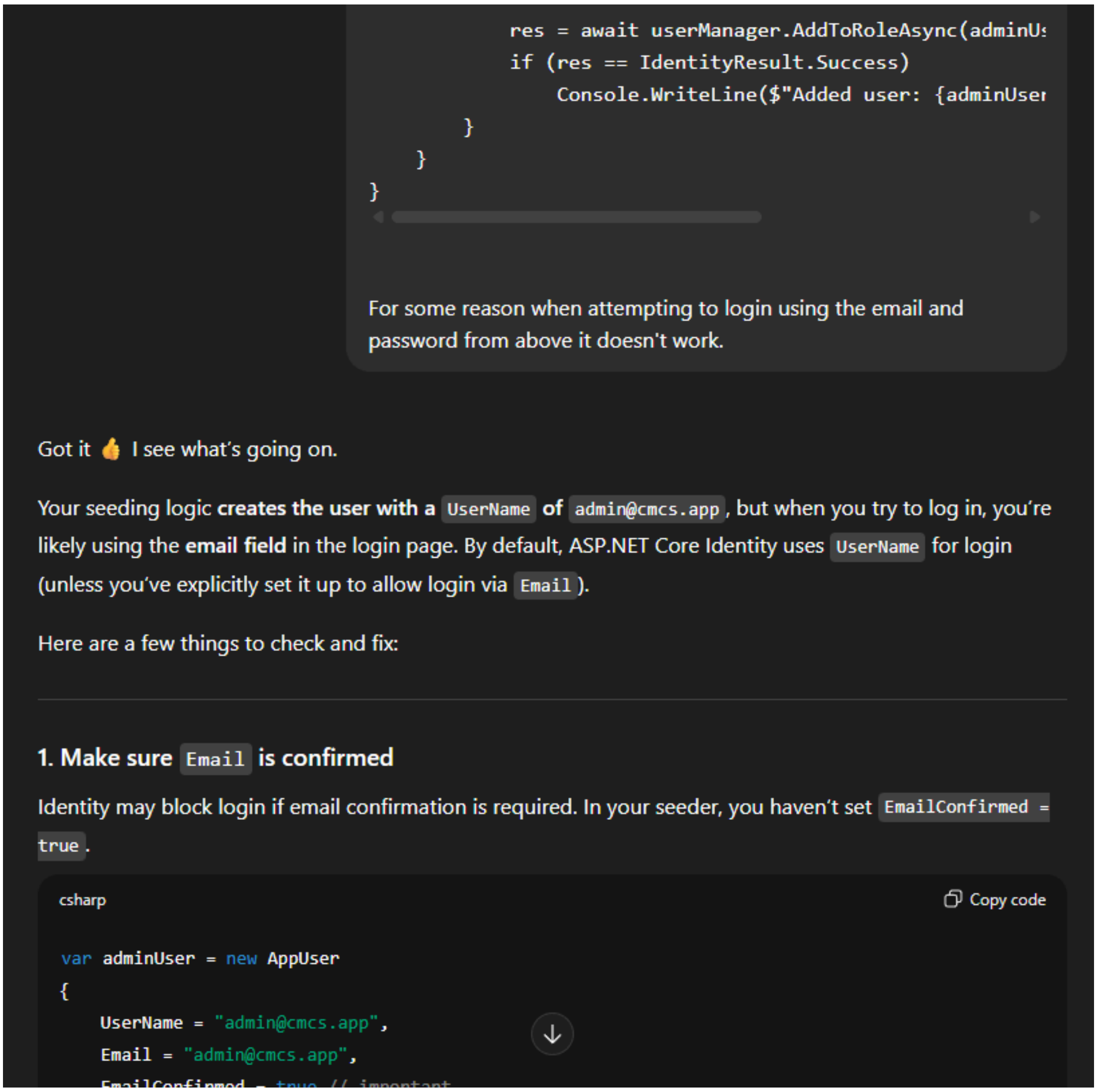
- An **index page** that lists all users and their assigned roles.
- Buttons to **manage a user's roles** (add/remove roles).
- A section listing all **existing roles** with options to add/remove them.

Models

```csharp
public class UserRolesViewModel
{
    public string UserId { get; set; }
    public string UserName { get; set; }
    public IList<string> Roles { get; set; } = new List<string>();
}

public class ManageUserRolesViewModel
{
    public string UserId { get; set; }
    public string UserName { get; set; }
```

- `DatabaseSeeder.cs`
  - I asked ChatGPT to help me troubleshoot some issues I was having with the database seeder.
  - Link: https://chatgpt.com/share/68c04c01-77a4-800b-ac30-db12e569f8af
  - Screenshot:



```
                res = await userManager.AddToRoleAsync(adminU
                if (res == IdentityResult.Success)
                    Console.WriteLine($"Added user: {adminUse
            }
        }
    }
```

For some reason when attempting to login using the email and password from above it doesn't work.

Got it 👍 I see what's going on.

Your seeding logic **creates the user with a** `UserName` **of** `admin@cmcs.app`, but when you try to log in, you're likely using the **email field** in the login page. By default, ASP.NET Core Identity uses `UserName` for login (unless you've explicitly set it up to allow login via `Email`).

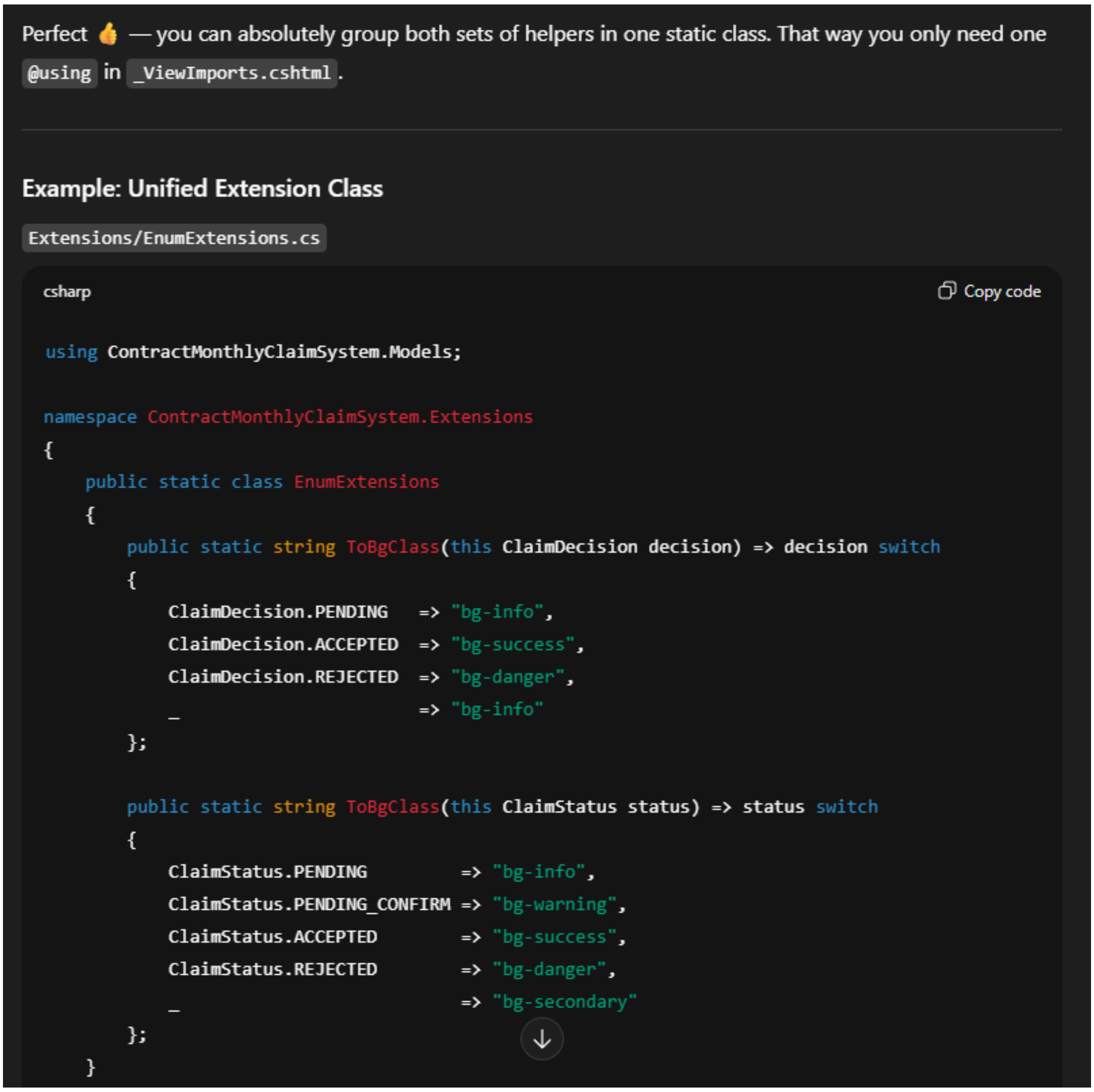
Here are a few things to check and fix:

1. Make sure `Email` is confirmed

Identity may block login if email confirmation is required. In your seeder, you haven't set `EmailConfirmed = true`.

```csharp
var adminUser = new AppUser
{
    UserName = "admin@cmcs.app",
    Email = "admin@cmcs.app",
```

- `EnumExtensions.cs`
    - I asked ChatGPT about some utility functions I made and learned about C# extensions.
    - Link: https://chatgpt.com/share/68c7f73a-4588-800b-a812-e5ef790cd5b1
    - Screenshot:



Perfect 👍 — you can absolutely group both sets of helpers in one static class. That way you only need one `@using` in `_ViewImports.cshtml`.

Example: Unified Extension Class

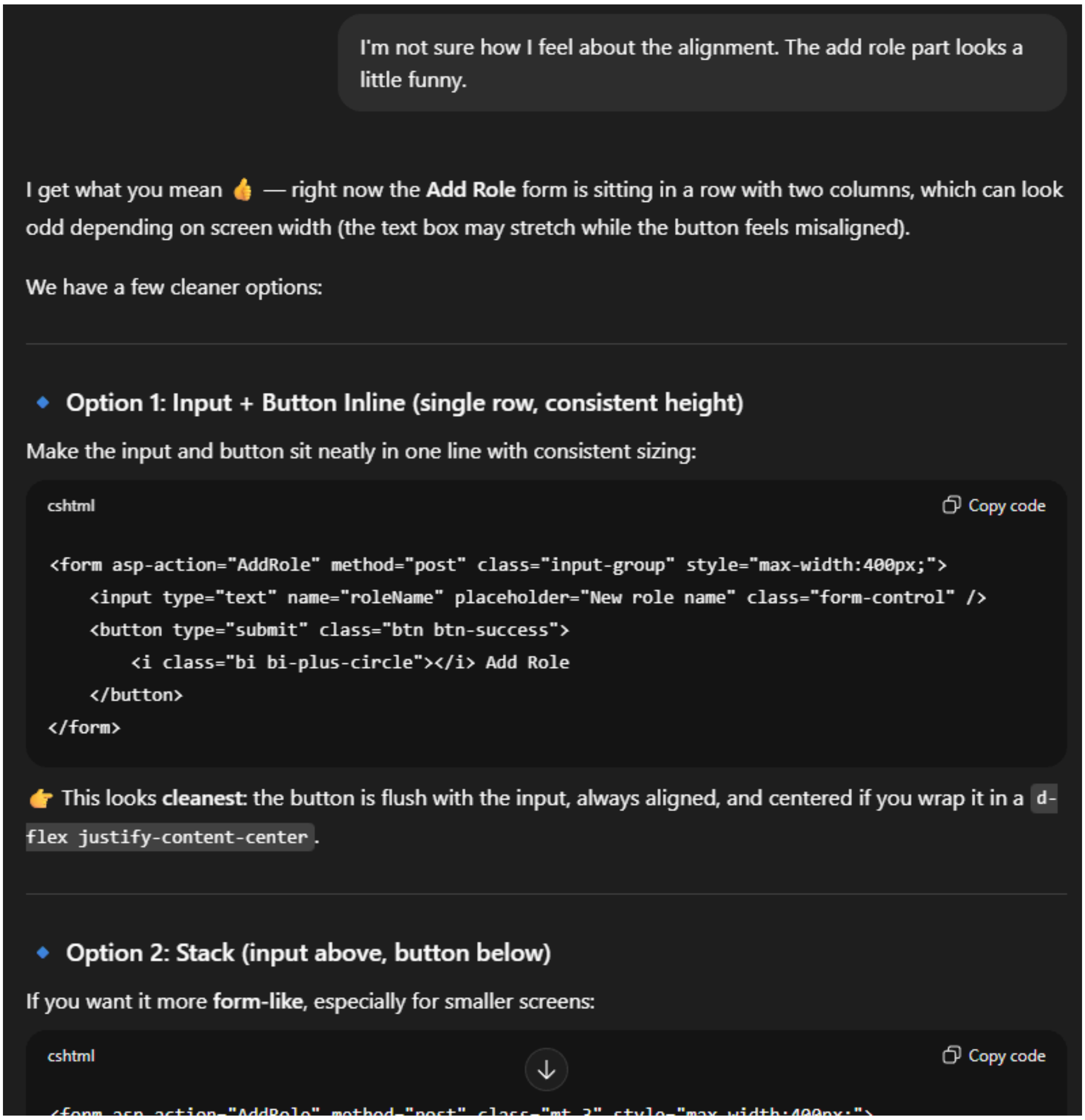
`Extensions/EnumExtensions.cs`

```csharp
using ContractMonthlyClaimSystem.Models;

namespace ContractMonthlyClaimSystem.Extensions
{
    public static class EnumExtensions
    {
        public static string ToBgClass(this ClaimDecision decision) => decision switch
        {
            ClaimDecision.PENDING   => "bg-info",
            ClaimDecision.ACCEPTED  => "bg-success",
            ClaimDecision.REJECTED  => "bg-danger",
            _                       => "bg-info"
        };

        public static string ToBgClass(this ClaimStatus status) => status switch
        {
            ClaimStatus.PENDING         => "bg-info",
            ClaimStatus.PENDING_CONFIRM => "bg-warning",
            ClaimStatus.ACCEPTED        => "bg-success",
            ClaimStatus.REJECTED        => "bg-danger",
            _                           => "bg-secondary"
        };
    }
```

- Most `.cshtml` views
  - I asked ChatGPT to help improve the look and feel of most of my views. I made follow-up prompts to change things if they didn't look right for some of them. Note that the links in the source code are included in each `view.cshtml` where ChatGPT was used, with the links being different because they were generated throughout different stages of the conversation.
  - Link: https://chatgpt.com/share/68ca99b5-dc0c-800b-b554-c315e49df063
  - Screenshot:



I'm not sure how I feel about the alignment. The add role part looks a little funny.

I get what you mean 👍 — right now the **Add Role** form is sitting in a row with two columns, which can look odd depending on screen width (the text box may stretch while the button feels misaligned).

We have a few cleaner options:

🔹 Option 1: Input + Button Inline (single row, consistent height)

Make the input and button sit neatly in one line with consistent sizing:

```cshtml
<form asp-action="AddRole" method="post" class="input-group" style="max-width:400px;">
    <input type="text" name="roleName" placeholder="New role name" class="form-control" />
    <button type="submit" class="btn btn-success">
        <i class="bi bi-plus-circle"></i> Add Role
    </button>
</form>
```

👉 This looks **cleanest**: the button is flush with the input, always aligned, and centered if you wrap it in a `d-flex justify-content-center`.

🔹 Option 2: Stack (input above, button below)

If you want it more **form-like**, especially for smaller screens:

```cshtml
<form asp-action="AddRole" method="post" class="mt-3" style="max-width:400px;">
```

References

OpenAI, 2025. *ChatGPT*. [online] Available at: <https://chatgpt.com/> [Accessed 19 October 2025].