

# PROG6212 POE Part 1

## Contents

1. Introduction .....	2
2. Design Choices .....	2
2.1. Assumptions and Constraints .....	2
2.1.1. Assumptions .....	2
2.1.2. Constraints .....	2
2.2. Code Structure .....	2
2.3. Database Structure .....	3
2.4. UML Diagram .....	3
2.4.1. Relationships .....	3
2.5. GUI Layout .....	4
2.6. Theme and Colour Scheme .....	4
2.7. Navigation Bar .....	4
2.8. Lecturer .....	4
2.8.1. Claims Dashboard .....	4
2.8.2. Create Claim .....	4
2.8.3. Claim Details .....	4
2.9. Academic Manager .....	5
2.9.1. Claims Dashboard .....	5
2.9.2. Claim details/review .....	5
2.10. Program Coordinator .....	5
2.10.1. Claims Dashboard .....	5
2.10.2. Claim details/review .....	5
2.10.3. Manage Lecturer Modules .....	5
2.11. Admin .....	5
2.11.1. Manage User Roles .....	5
2.12. Project Plan .....	5
3. UML Class Diagram .....	6
4. Project Plan .....	7
5. GUI/UI Showcase .....	8
5.1. Lecturer Views .....	8
5.2. Academic Manager Views .....	9
5.3. Program Coordinator Views .....	11
5.4. Admin Views .....	15
6. AI Usage and Disclaimer .....	16
References .....	22

## 1. Introduction

The Contract Monthly Claim System (CMCS) is a system for creating and reviewing contract claims. The system has three types of users: lecturers, program coordinators, and academic managers. Lecturers are able to create new claims and view the status of existing claims belonging to them. A program coordinator and an academic manager must both review a claim for it to be complete, with the claim only being approved if both reviewers accept it, otherwise it is rejected. The scope of part 1 of this project covers design choices, database structure, project plan, and a UI prototype which does not require working functionality.

## 2. Design Choices

### 2.1. Assumptions and Constraints

#### 2.1.1. Assumptions

- User authentication is required
- Users have roles which define what they are allowed to do, with roles for lecturer, program coordinator, academic manager, and admin
- New users cannot do anything, and admins must assign users to relevant roles
- Lecturers teach modules.
- Program coordinators create and manage modules, assigning modules to lecturers.
- Each claim is for one module that the lecturer teaches.
- Each claim may optionally have multiple supporting documents.
- Each claim must be reviewed by a program coordinator and an academic manager to be complete, with the claim only being accepted if both agree else rejected.
- A lecturer may leave a comment when creating a claim, and reviewers may leave comments when reviewing a claim.
- The system assumes lecturers are honest when creating claims. It does not have any way to validate whether a lecturer really worked a specific number of hours, or is really paid a specific hourly rate.

#### 2.1.2. Constraints

- Claims have one and only one module.
- One program coordinator review per claim, one academic manager review per claim.
- Uploaded documents for claims must be one of the following types: .pdf, .doc, .docx, .xls, .xlsx, .ppt, .pptx, .txt, or .md.
- Lecturers may only view their own claims, and only download their own supporting documents.

### 2.2. Code Structure

The system is implemented as an ASP.NET Core .NET 8 Model View Controller (MVC) project. The project makes use of Entity Framework Core for database integration and ASP.NET Core Identity for user authentication and role management. Following the MVC pattern, the project has multiple models, views, and controllers.

There are models for database entities and ViewModels, organised in separate folders and namespaces. Views only use ViewModels with properties relevant to the view, as opposed to

directly using models that represent database entries, in order to prevent accidents from direct access and ensure database integrity.

There are separate controllers for handling different user/role functionality, with most restricting access only to one specific role (e.g. lecturer only for the lecturer controller). Some controllers allow multiple roles, for example the controller for managing lecturer modules allows admins and program coordinators.

## 2.3. Database Structure

The database contains tables for ASP.NET Core Identity users and roles, modules, lecturer modules, contract claims, uploaded files, and contract claim documents.

The users table stores user information such as email, hashed password, etc. Lecturers, program coordinators, and academic managers are all stored as users, with their respective roles used to distinguish them.

The modules table stores module name and code. Lecturers can teach multiple modules, and modules may be taught by multiple lecturers, with the lecturer modules table to keep track of who teaches what. Contract claims reference the lecturer who created the claim, the module taught by the lecturer, hours worked, hourly rate, lecturer comment, the program coordinator and academic manager who reviews the claim along with their review decision and an optional comment, and finally the status of the claim. A contract claim may have multiple supporting documents, which are stored as uploaded files and linked to their respective claims by the contract claim documents table.

## 2.4. UML Diagram

The UML diagram depicts the database entities and their relationships with each other. For the sake of simplicity, I have depicted the “User” entity/class being extended by “LecturerUser” and “ReviewerUser”, which is further extended by “ProgramCoordinator” and “AcademicManager”. In the actual project, they are all simply users with roles assigned to distinguish them. The functionality depicted in the diagram is not implemented by the models themselves, but rather controllers with access restricted by role, for example lecturer users have access to the lecturer controller/views and the create claim functionality.

### 2.4.1. Relationships

- Lecturer users may have zero to many lecturer modules, with each lecturer module referencing one lecturer and one module, with modules referenced by zero to many lecturer modules.
- Lecturers may have zero to many contract claims, with each contract claim referencing one lecturer.
- Modules may be referenced by zero to many contract claims, with each contract claim referencing one module.
- Program coordinators may review zero to many contract claims, with each contract claim referencing zero or one program coordinator based on whether one has reviewed it yet.
- Academic managers may review zero to many contract claims, with each contract claim referencing zero or one academic managers based on whether one has reviewed it yet.

- Each contract claim document refers to one contract claim and one uploaded file, with contract claims having zero to many contract claim documents, and uploaded files referenced by zero to many contract claim documents.

## 2.5. GUI Layout

## 2.6. Theme and Colour Scheme

I use the Flatly bootstrap theme, which features a flat and modern aesthetic with a neutral yet attractive colour-scheme (Park, 2013). I chose this theme because it looks good, it's easy on the eyes but isn't over the top or obnoxious. It helps to distinguish my web application from the default theme featured in every ASP.NET Core MVC web application.

## 2.7. Navigation Bar

The GUI features a navigation bar at the top, allowing users to easily navigate to and access different parts of the web application. The items in the navigation bar are dynamic, changing based on the role of the current user. If the user is not logged in or does not have a role yet, it only displays links to the home and privacy pages. Otherwise, it will display links to the user's role specific claim dashboard view, and additional links to other views featuring role specific functionality (such as role management for admins, lecturer module management for admins/program coordinators).

## 2.8. Lecturer

### 2.8.1. Claims Dashboard

The lecturer claims dashboard features a prominent create claim button, and two tables displaying pending and completed claims. The claims shown include useful information such as payment amount, reviewer decisions, and claim status, with a details button for each claim that goes to the claim details view to show more information. This design is practical, allowing lecturers to easily view claims status and information, as well as create a new claim with a click of a button.

### 2.8.2. Create Claim

The create claim page lets a lecturer select which module the claim is for, input the hours worked, hourly rate, optional comment, upload multiple supporting documents, and submit the claim or return to the dashboard with a click of a button. There is also a helpful preview of the total payment amount calculated based on hours worked and hourly rate. The design is intuitive and user-friendly.

### 2.8.3. Claim Details

The claim details page shows all the information that a lecturer is allowed to see, including: module name, hours worked, hourly rate, payment amount, lecturer comment, program coordinator decision and comment, academic manager decision and comment, claim status, and any uploaded supporting documents with options to download them. The design is simple and straightforward.

## 2.9. Academic Manager

### 2.9.1. Claims Dashboard

The academic manager claims dashboard features tables displaying pending claims that may be reviewed, claims pending further confirmation, and completed claims. Each claim has a button that directs to details about the claim and enables the claim to be reviewed. The design provides a clear overview of existing claims.

### 2.9.2. Claim details/review

The claim details page for academic managers shows more information than the lecturer's version, with additional sections for: lecturer name, program coordinator name, and academic manager name. Additionally, if the claim has not yet been reviewed by an academic manager, there is a section to review the claim with an optional comment and simple accept/reject buttons. The design is clear and functional, displaying relevant information and allowing review.

## 2.10. Program Coordinator

### 2.10.1. Claims Dashboard

The claims dashboard for program coordinators is very similar to that of academic managers, with the difference being that pending claims are claims previously reviewed by a program coordinator not an academic manager.

### 2.10.2. Claim details/review

The claim details/review for program coordinators is also very similar to that of academic managers, with the difference being that the review is done by a program coordinator.

### 2.10.3. Manage Lecturer Modules

This view is only accessible by admins or program coordinators. It displays lecturers and the modules taught by them, as well as a list of all modules, with options to add/remove modules and edit modules taught by specific lecturers.

## 2.11. Admin

### 2.11.1. Manage User Roles

Displays all application users and their respective roles with options to change specific user roles, as well as a list of all roles with options to add new roles or delete existing roles. The purpose of this page is to allow admins/developers to modify user roles via the front-end instead of the back-end.

## 2.12. Project Plan

The project plan consists of epics for each part of this assignment, with tasks outlining things to do for each part, with subtasks describing things to do more specifically. The plan includes dependencies as well as start and end times for epic and tasks.

### 3. UML Class Diagram

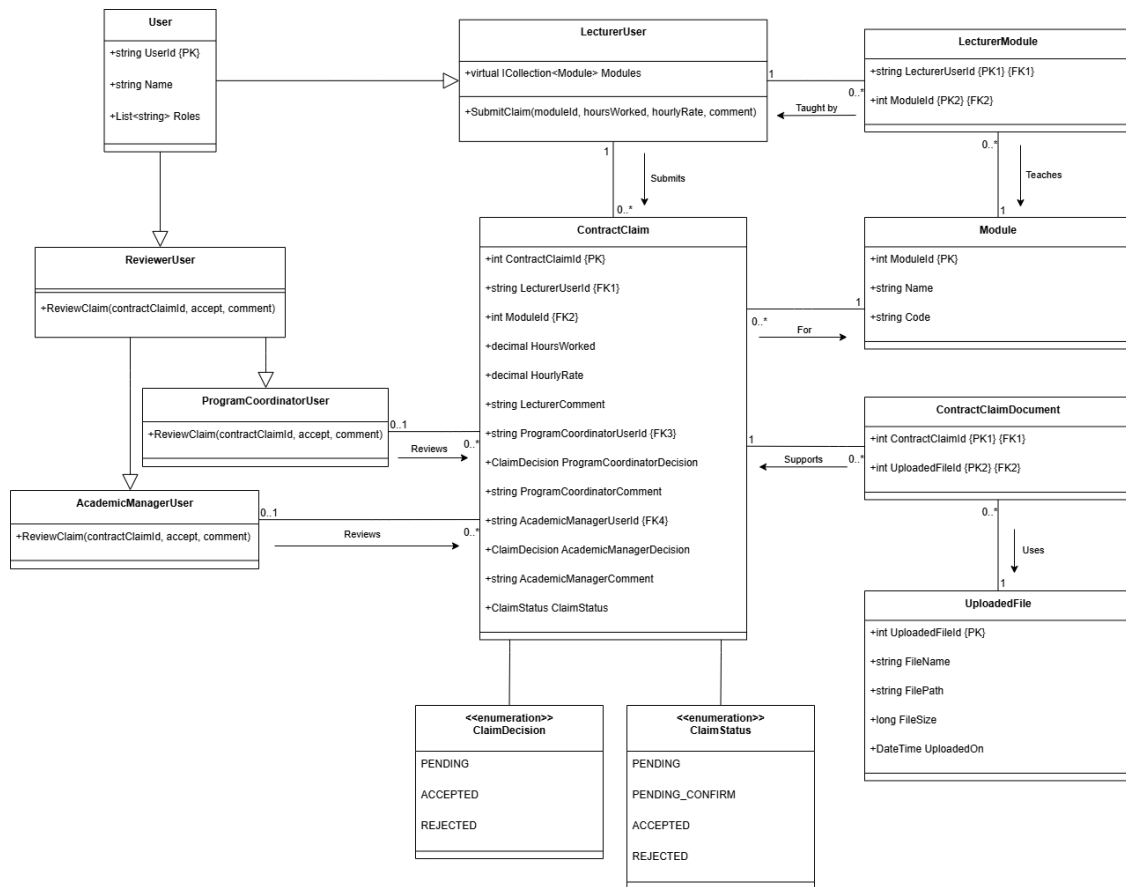


Figure 1: The UML Class Diagram.

#### 4. Project Plan

The project plan was created using Jira. The plan includes epics for parts 1-3 of the POE, as well as tasks with subtasks. To view all the tasks with their subtasks you will have to view the project on Jira. You may view the project plan via [this Jira link](#).

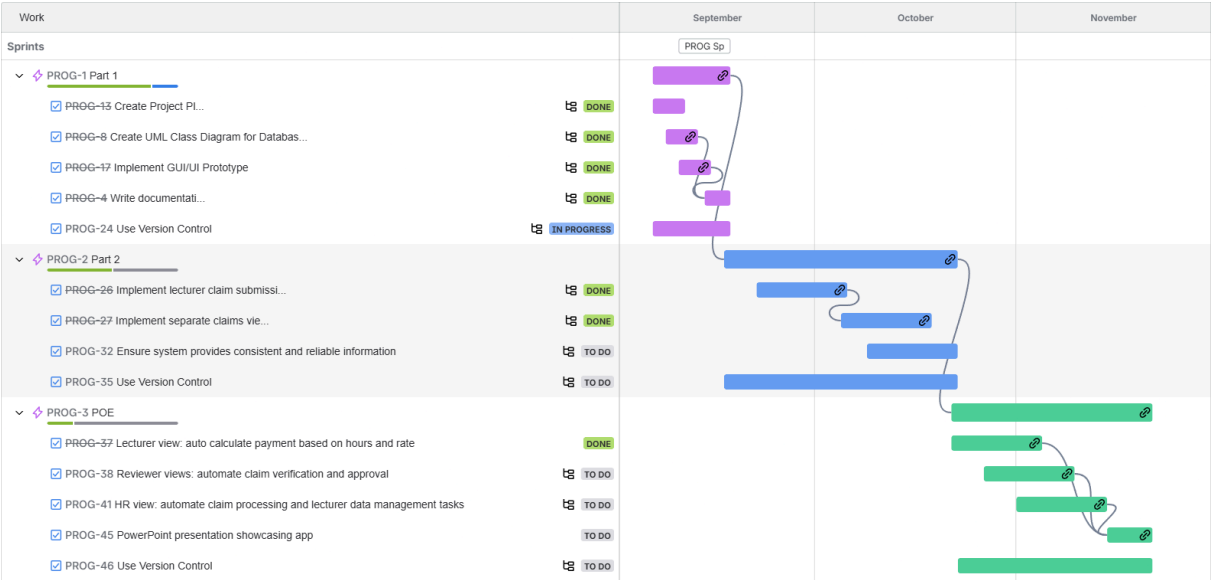


Figure 2: The Project Plan Timeline.

## 5. GUI/UI Showcase

### 5.1. Lecturer Views

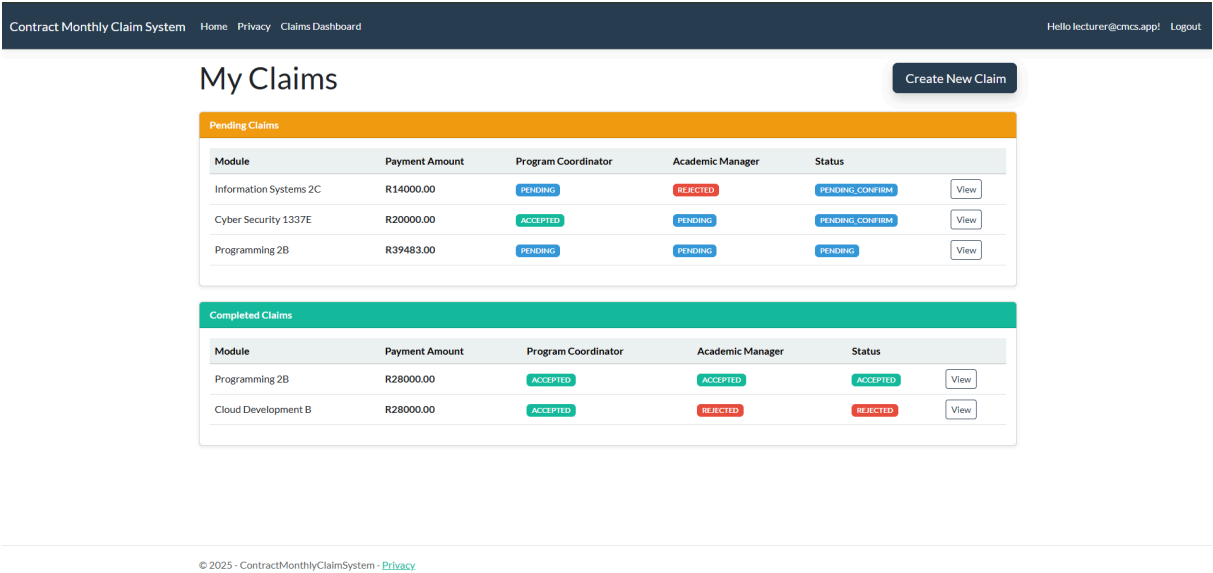


Figure 3: Lecturer Claims Dashboard

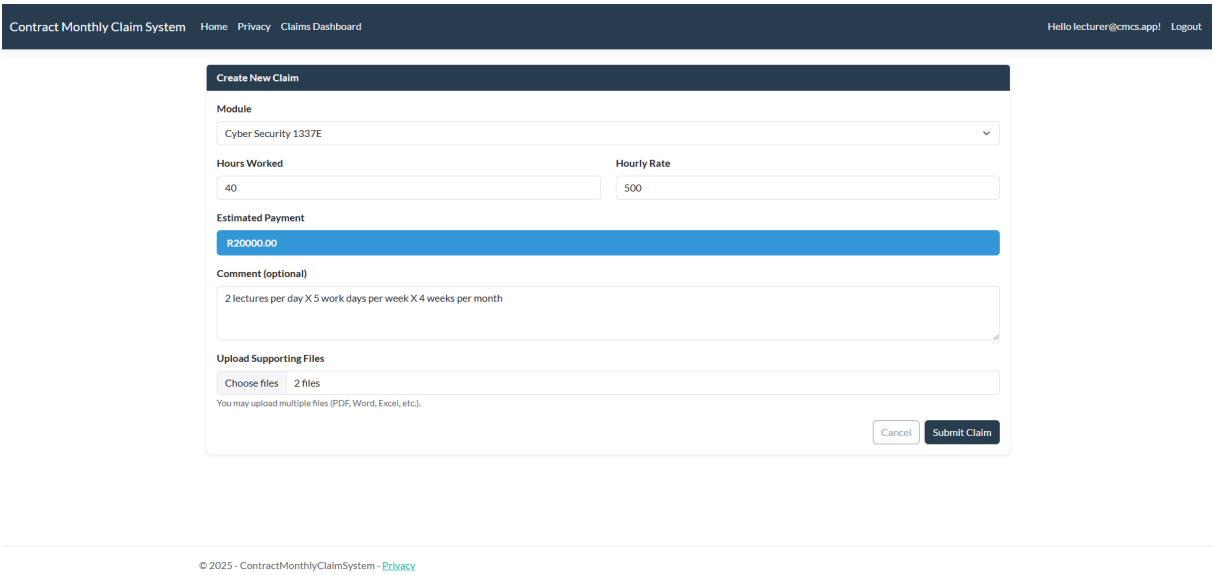


Figure 4: Lecturer Create Claim



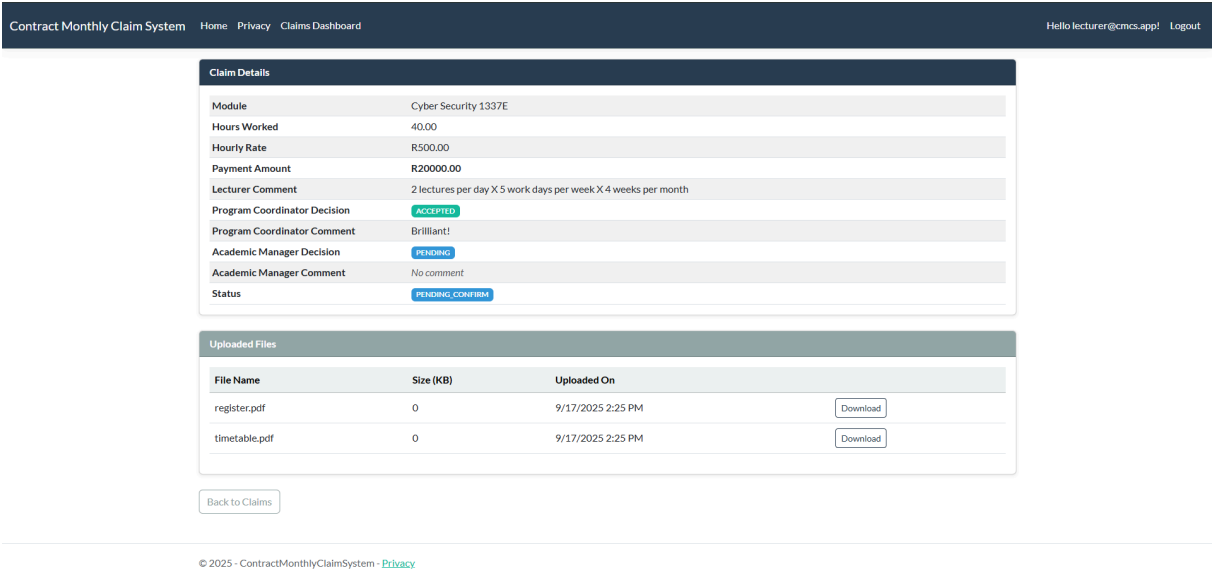


Figure 5: Lecturer Claim Details

5.2. Academic Manager Views

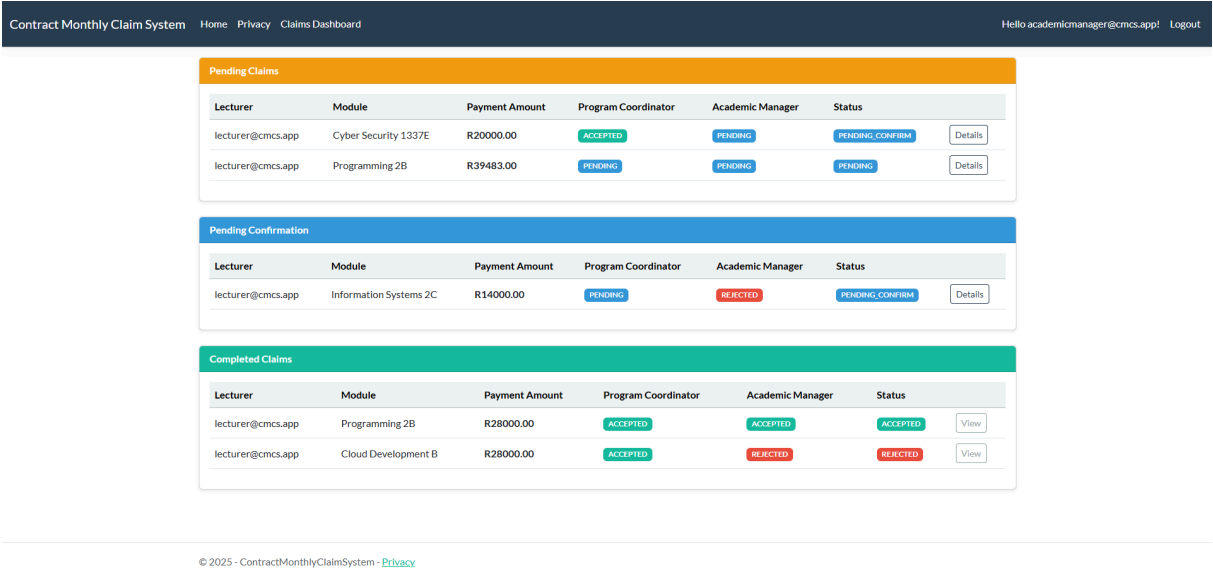


Figure 6: Academic Manager Claims Dashboard

Contract Monthly Claim System

Home

Privacy

Claims Dashboard

Hello academicmanager@cmcs.app!

Logout

Claim Details

Lecturer	lecturer@cmcs.app
Module	Programming 2B
Hours Worked	123.00
Hourly Rate	R321.00
Payment Amount	R39483.00
Lecturer Comment	123 X 321
Program Coordinator	
Program Coordinator Decision	PENDING
Program Coordinator Comment	No comment
Academic Manager	
Academic Manager Decision	PENDING
Academic Manager Comment	No comment
Status	PENDING

Uploaded Files

No files uploaded for this claim.

Review Claim

Comment

\*123X321\* - what?

Figure 7: Academic Manager Review Claim 1

Hourly Rate

R321.00

Payment Amount

R39483.00

Lecturer Comment

123 X 321

Program Coordinator

Program Coordinator Decision

PENDING

Program Coordinator Comment

No comment

Academic Manager

Academic Manager Decision

PENDING

Academic Manager Comment

No comment

Status

PENDING

Uploaded Files

No files uploaded for this claim.

Review Claim

Comment

\*123X321\* - what?

Accept

Reject

Back to Claims

© 2025 - ContractMonthlyClaimSystem - Privacy

Figure 8: Academic Manager Review Claim 2

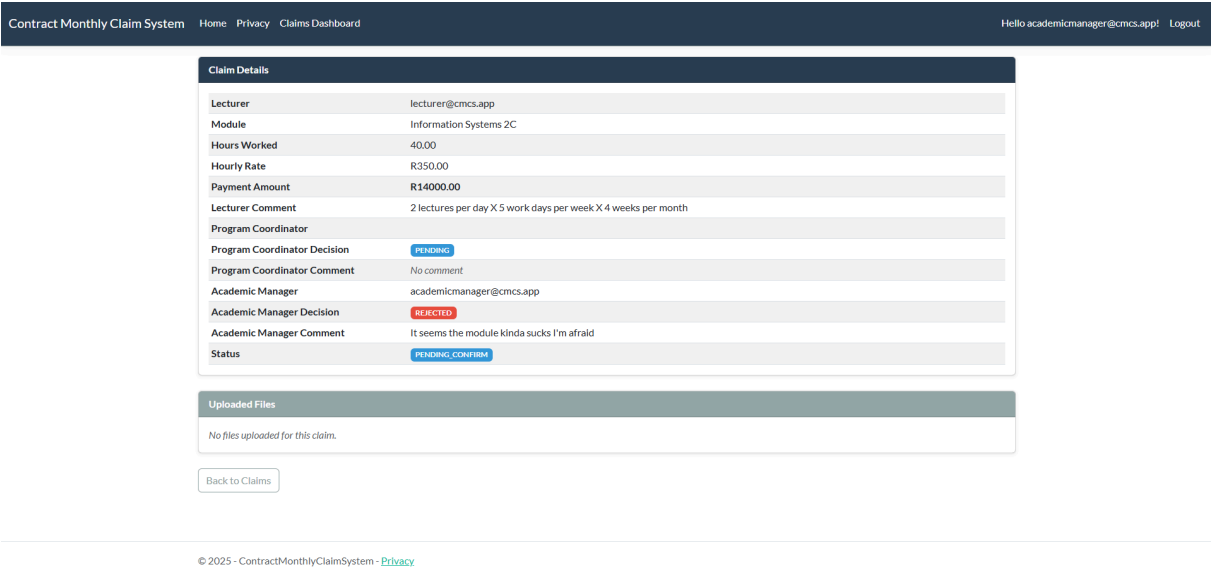


Figure 9: Academic Manager Claim Details

5.3. Program Coordinator Views

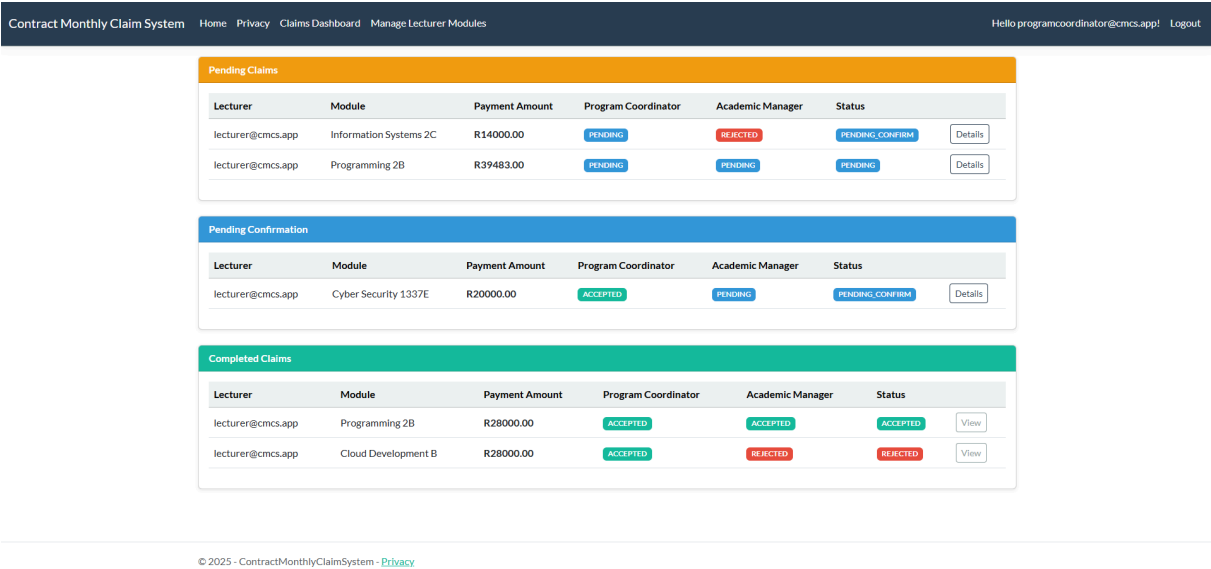


Figure 10: Program Coordinator Claims Dashboard

Contract Monthly Claim System

Home

Privacy

Claims Dashboard

Manage Lecturer Modules

Hello program.coordinator@cmcs.app!

Logout

Claim Details

Lecturer

lecturer@cmcs.app

Module

Cyber Security 133/E

Hours Worked

40.00

Hourly Rate

R500.00

Payment Amount

R20000.00

Lecturer Comment

2 lectures per day X 5 work days per week X 4 weeks per month

Program Coordinator

Program Coordinator Decision

PENDING

Program Coordinator Comment

No comment

Academic Manager

Academic Manager Decision

PENDING

Academic Manager Comment

No comment

Status

PENDING

Uploaded Files

File Name	Size (KB)	Uploaded On	
register.pdf	0	9/17/2025 2:25 PM	Download
timetable.pdf	0	9/17/2025 2:25 PM	Download

Review Claim

Figure 11: Program Coordinator Review Claim 1

Program Coordinator

Program Coordinator Decision

PENDING

Program Coordinator Comment

No comment

Academic Manager

Academic Manager Decision

PENDING

Academic Manager Comment

No comment

Status

PENDING

Uploaded Files

File Name	Size (KB)	Uploaded On	
register.pdf	0	9/17/2025 2:25 PM	Download
timetable.pdf	0	9/17/2025 2:25 PM	Download

Review Claim

Comment

Brilliant!

Accept

Reject

Back to Claims

© 2025 - ContractMonthlyClaimSystem - Privacy

Figure 12: Program Coordinator Review Claim 2

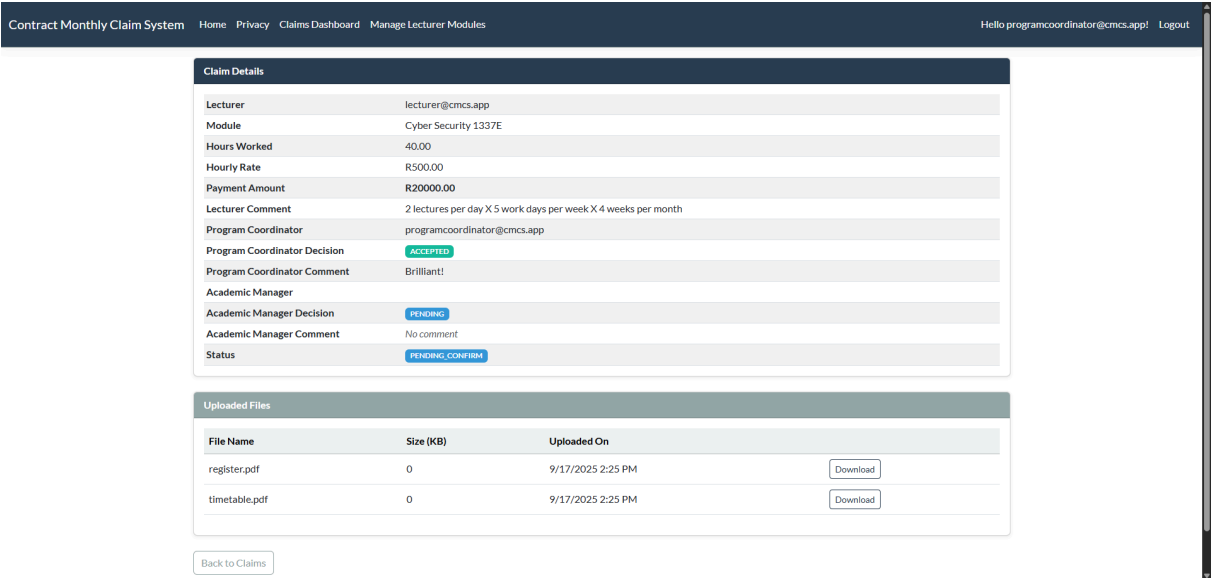


Figure 13: Program Coordinator Claim Details 1

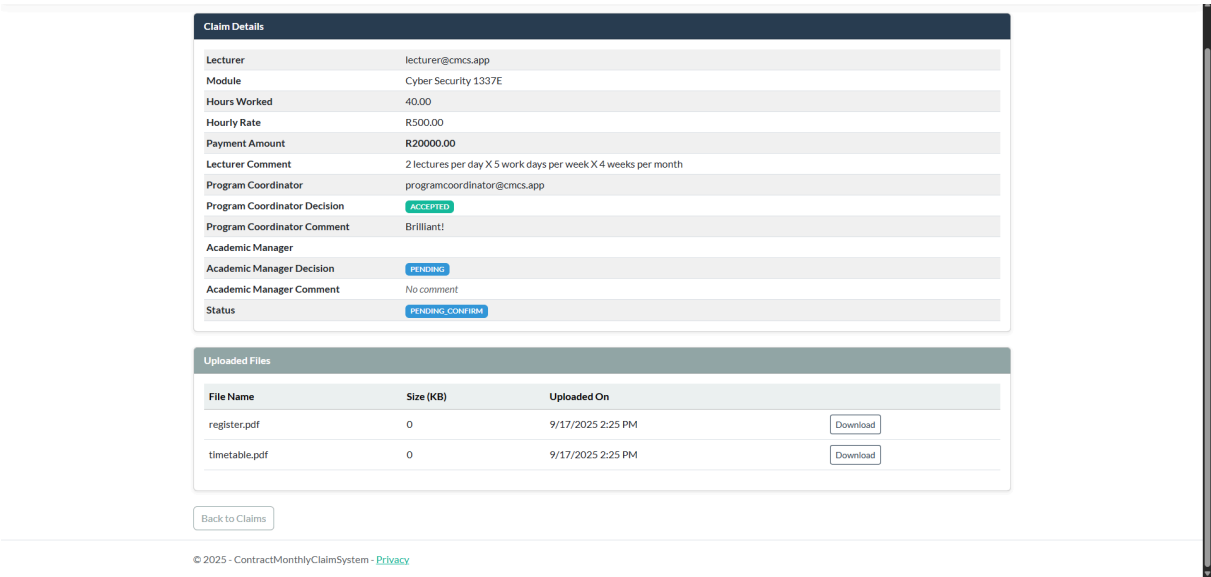


Figure 14: Program Coordinator Claim Details 2

Contract Monthly Claim System

HomePrivacyClaims DashboardManage Lecturer Modules

Hello programcoordinator@cmcs.app!Logout

Lecturers

Name	Modules
lecturer@cmcs.app	PROG6212, CLDV6212, INSY7213, CSEC1337

Manage Modules

All Modules

Module Name	Module Code
Programming 2B	PROG6212
Cloud Development B	CLDV6212
Information Systems 2C	INSY7213
Cyber Security 1337E	CSEC1337

Add New Module

Module Name

Enter module name

Module Code

Enter module code

Add Module

© 2025 - ContractMonthlyClaimSystem - [Privacy](#)

Figure 15: Program Coordinator Manage Lecturer Modules

Contract Monthly Claim System

HomePrivacyClaims DashboardManage Lecturer Modules

Hello programcoordinator@cmcs.app!Logout

Manage Modules for lecturer@cmcs.app

Assigned Modules

Module	Action
Programming 2B	Remove
Cloud Development B	Remove
Cyber Security 1337E	Remove

Available Modules

Module	Action
Information Systems 2C	Assign

Back to List

© 2025 - ContractMonthlyClaimSystem - [Privacy](#)

Figure 16: Program Coordinator Assign Lecturer Modules

## 5.4. Admin Views

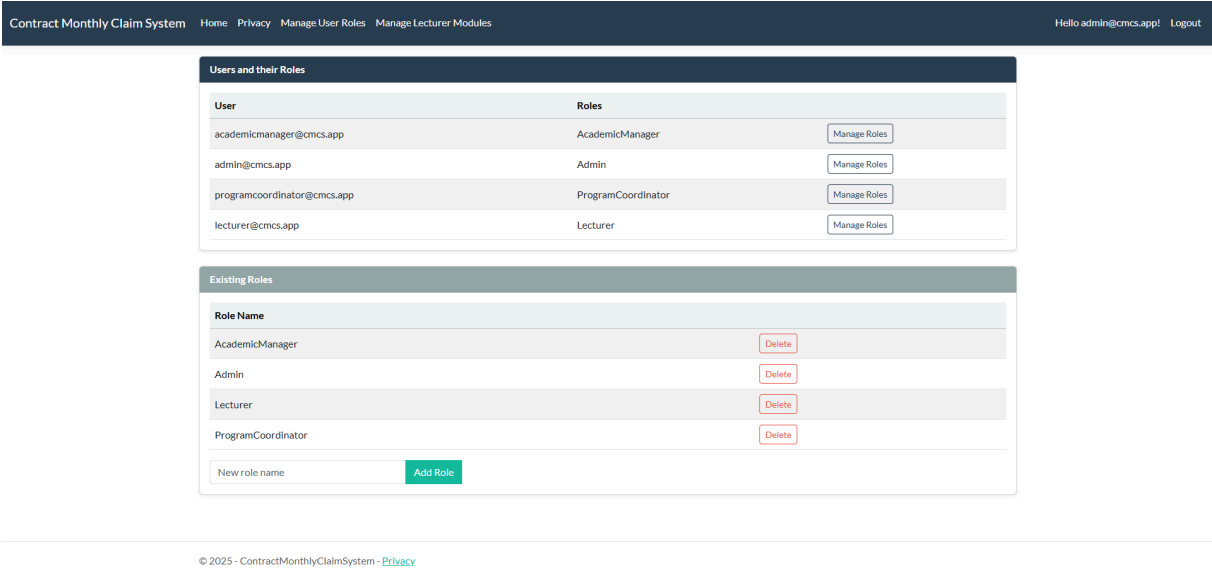


Figure 17: Admin Manage Roles

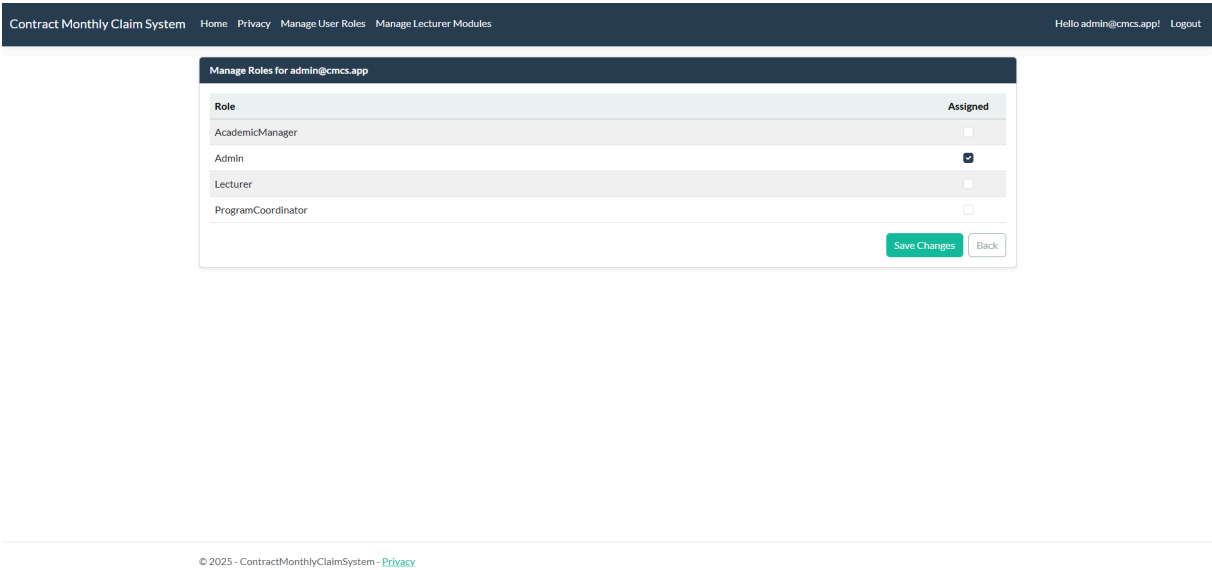
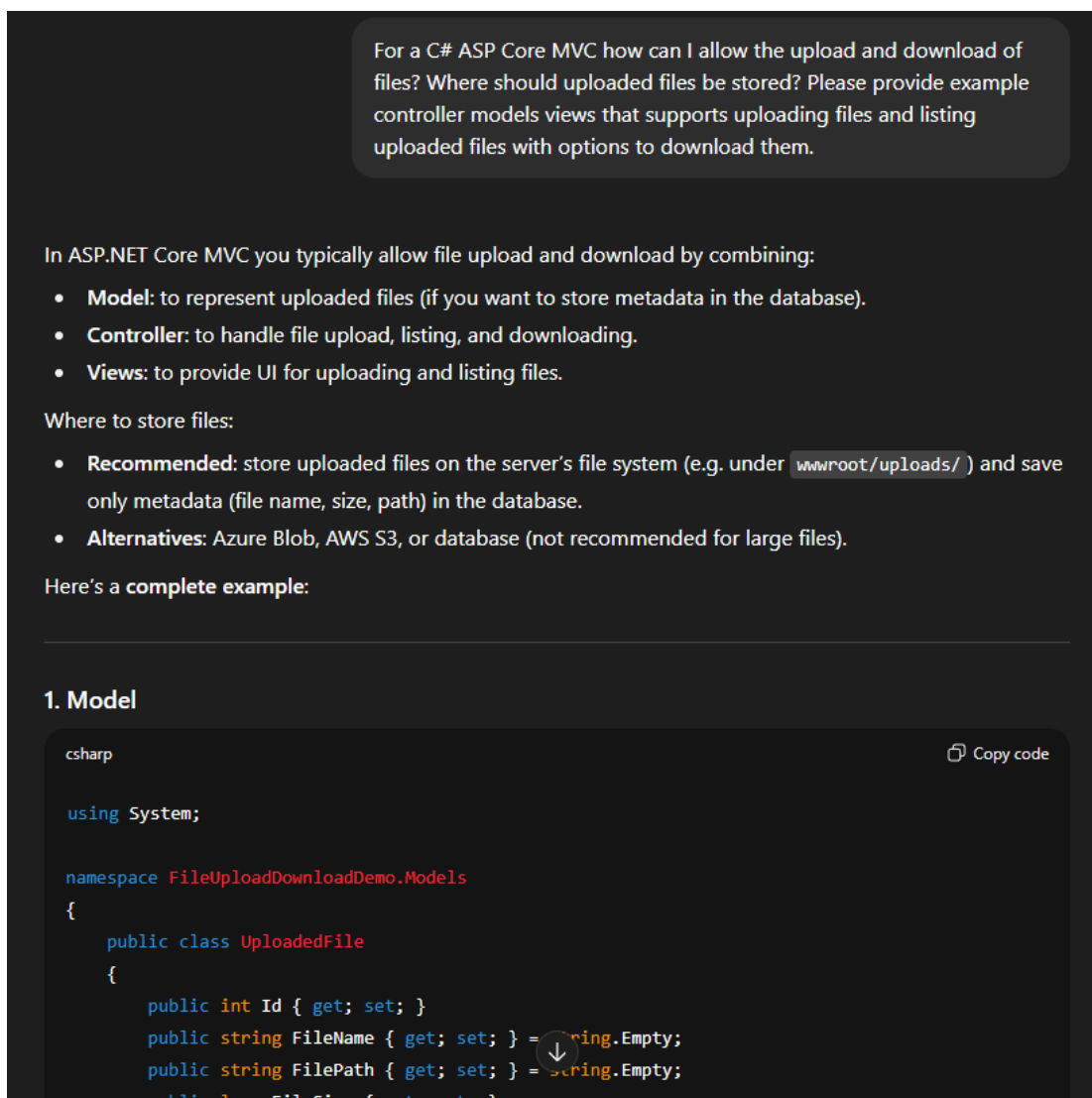


Figure 18: Admin Assign User Roles

## 6. 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.

- 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:





- 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;
        }
    }
}
```

Copy code

- 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

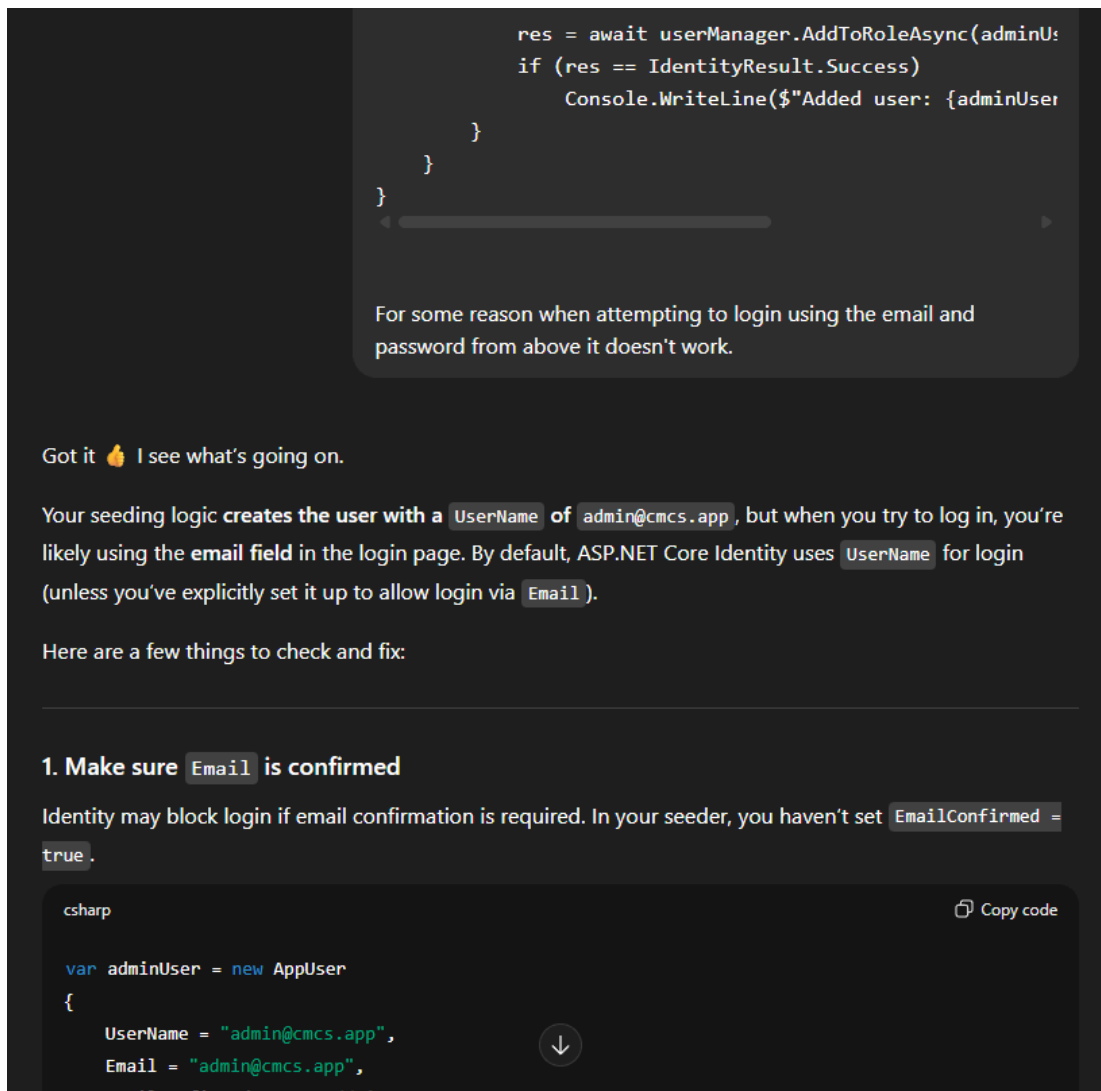
csharp

 Copy code

```
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:



- 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"
        };

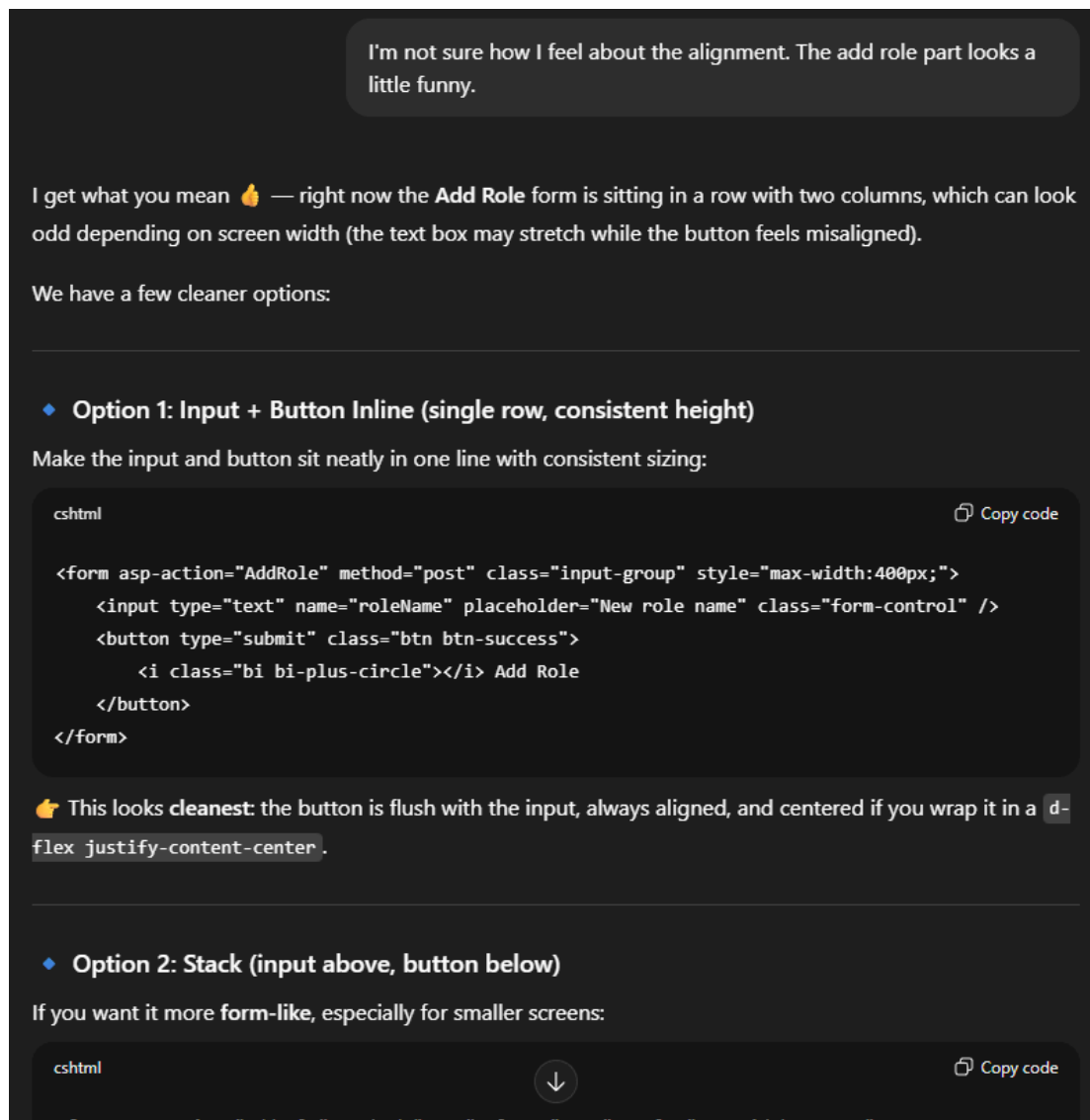
        public static string ToBgClass(this ClaimStatus status) => status switch
        {
            ClaimStatus.PENDING => "bg-info",
            ClaimStatus.PENDING_CONFIRM => "bg-warning",
            ClaimStatus.ACCEPTED => "bg-success",
            ClaimStatus.REJECTED => "bg-danger",
            _ =>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

Copy code

↓

- 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:



## References

OpenAI, 2025. *ChatGPT*. [online] Available at: <<https://chatgpt.com/>> [Accessed 17 September 2025].

Park, T., 2013. *Bootswatch: Flatly*. [online] Available at: <<https://bootswatch.com/flatly/>> [Accessed 17 September 2025].