PROG6212 POE Part3

Student Number: ST10443516

Presentation Topic

Contract Monthly Claim System

Digital workflow for lecturer Claim Management

What this Presentation covers

- Key system updates from part 2 to part 3
- Full walk through of the CMCS user journey
- Visual demonstration of all major features
- UX/UI Improvements
- The system value and impact

Part 2

Core Features of Part2

- **Lecturer Functionality**
  - Submit monthly claims
  - Upload supporting documents
  - View (Track) list of submitted claims
- **Coordinator Functionality**
  - View pending claims
  - Approve or reject lecturer claims
- **Manager Functionality**
  - View coordinator-approved claims
  - Approve or reject them
- **Database & Models Built**
  - Claims, Documents, Users, User Roles
- **Basic UI & Bootstrap Layout**
  - Tables, forms, and buttons functioning but minimal design

Part 3

Core Features of Part3

- Added HR who is a super user
- Login & Sessions
- Auto-calculation for lecturers
- Secure role based access
- PDF GENERATOR
- Removal of manual rate entry

System Architecture Overview

User Roles & Workflow:

Lecturer → Coordinator → Manager → HR → PDF Reports

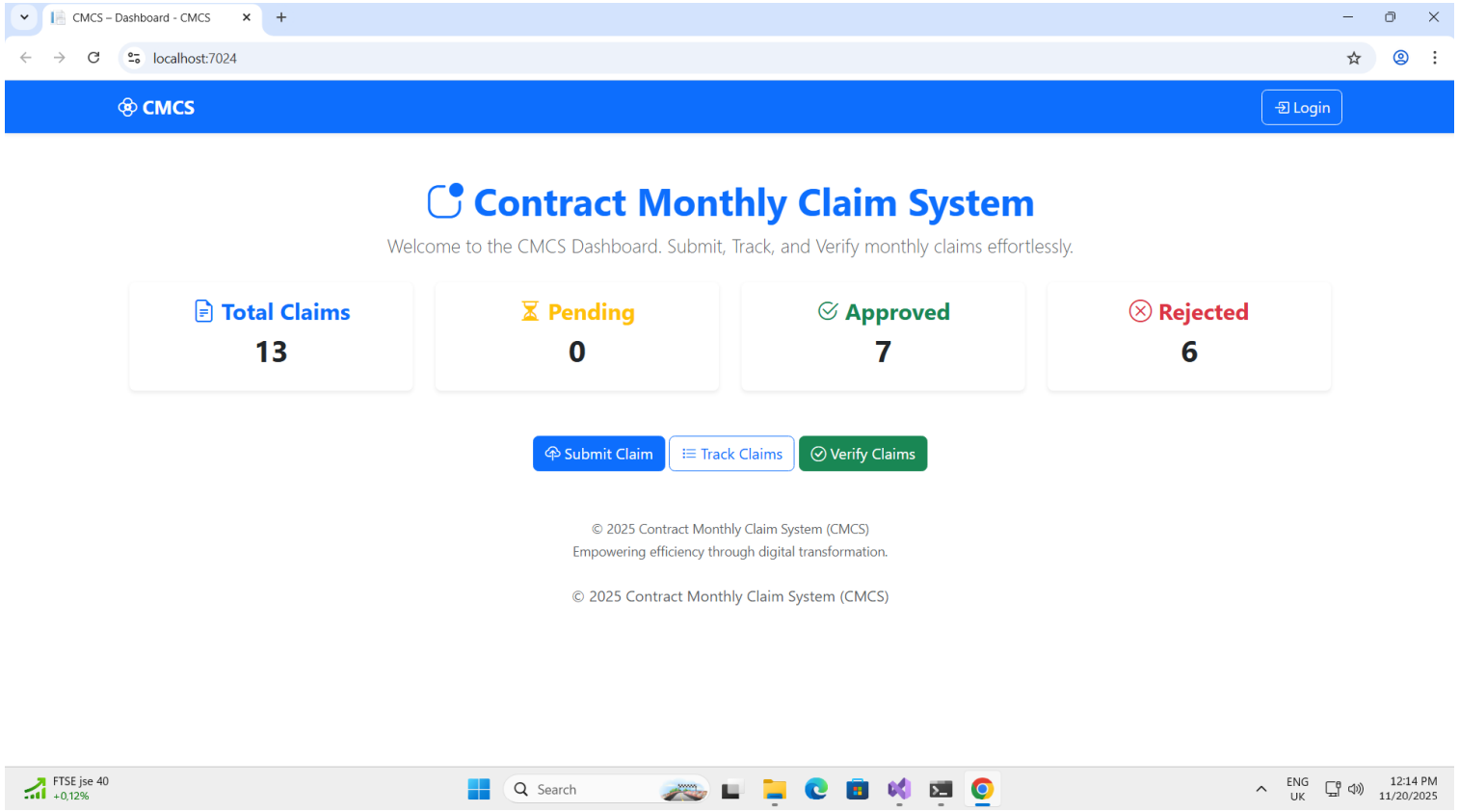
Functional Flow:

- Lecturer submits claim + uploads evidence
- Coordinator validates & approves/rejects
- Manager reviews coordinator-approved claims
- HR completes administrative processing + manages user accounts
- System generates PDF reports for auditing

Technologies Used:

- ASP.NET Core MVC
- SQL / EF Core / Azure Storage
- Bootstrap 5
- QuestPDF for reporting

Home Dashboard.

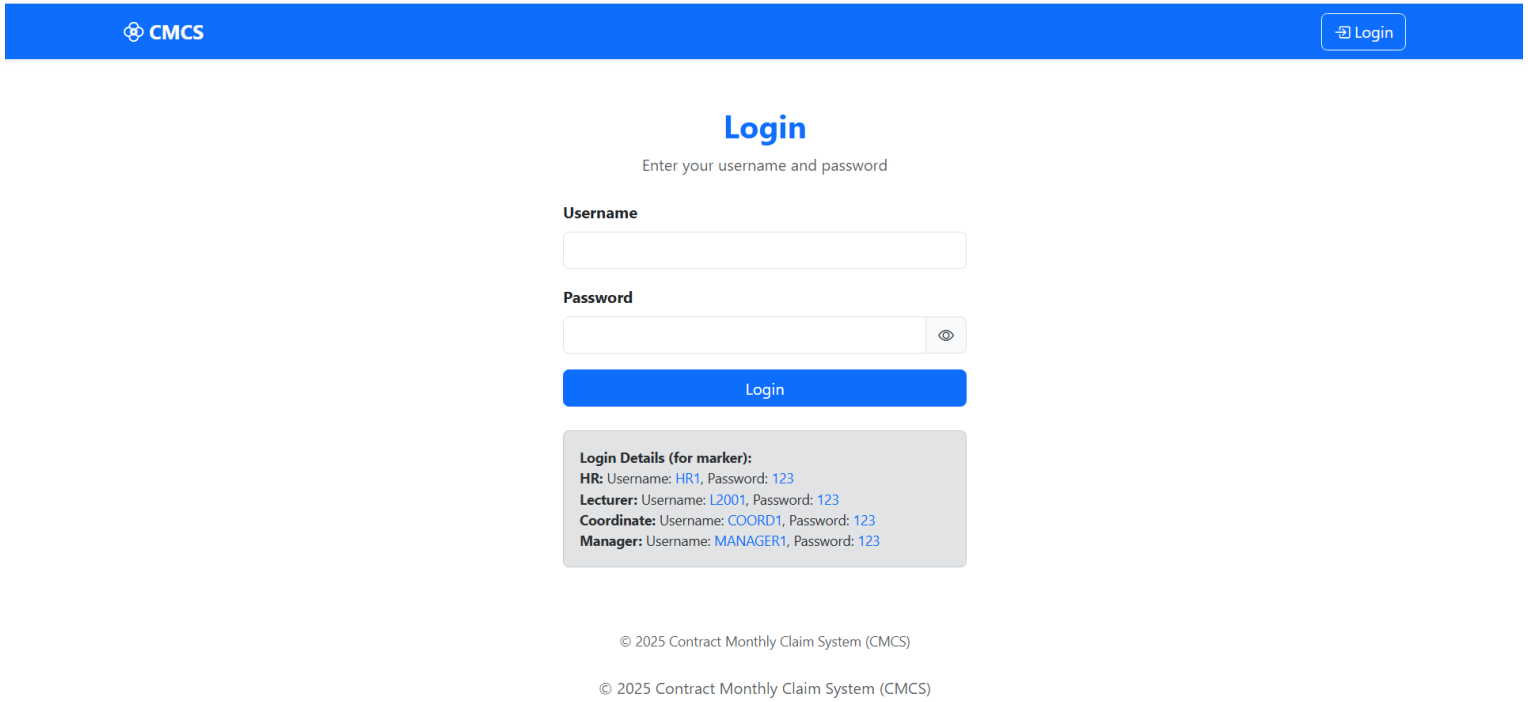


Home Dashboard

Features of the Home dashboard

- Instant overview of user activity
- Login button
- Summary cards:
  - Total Claims
  - Pending Claims
  - Approved Claims
  - Rejected Claims
- Personalized greeting for logged-in user
- Clean Bootstrap card styling

Login screen



© 2025 Contract Monthly Claim System (CMCS)

Login Screen

Features of the login page

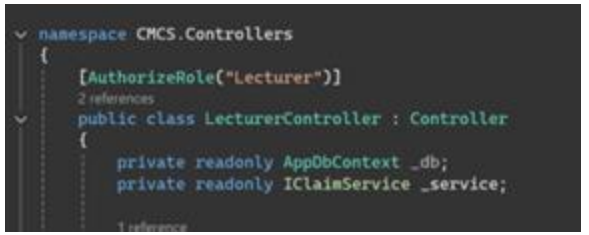
- Clean, professional login layout
- **“Show/Hide Password”** eye icon for accessibility
- User role-based redirection after login
- Validation messages for incorrect credentials
- Secure handling of sessions

Sessions + Role-Based Access Control

- Users can only access screens meant for their role.
- HR → User management

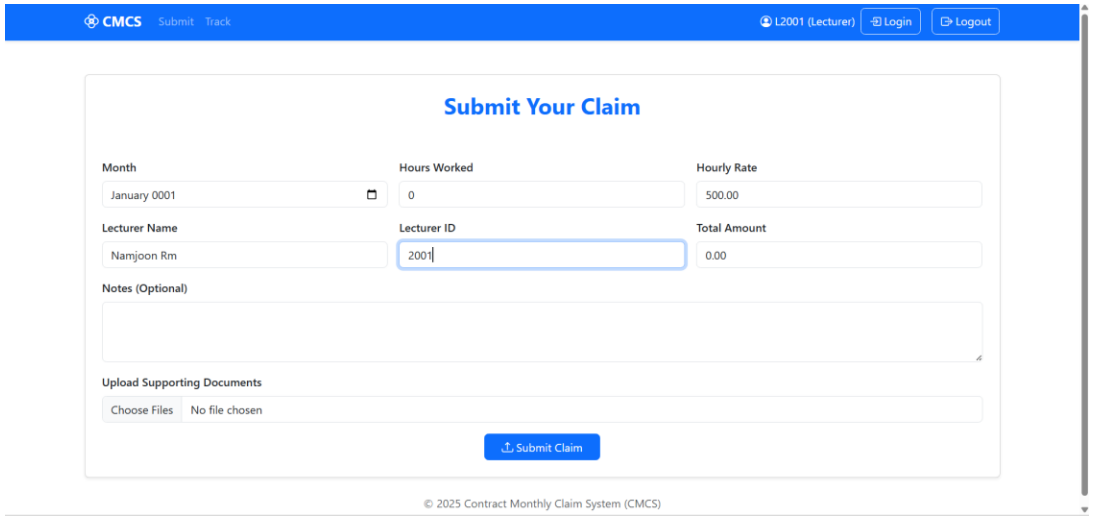
Lecturer → Submit + Track
Coordinator → Verify
Manager → Approve & Pay

- Explain Under Screenshot
- Each controller action is restricted
- If a user attempts to access another role's page → access denied

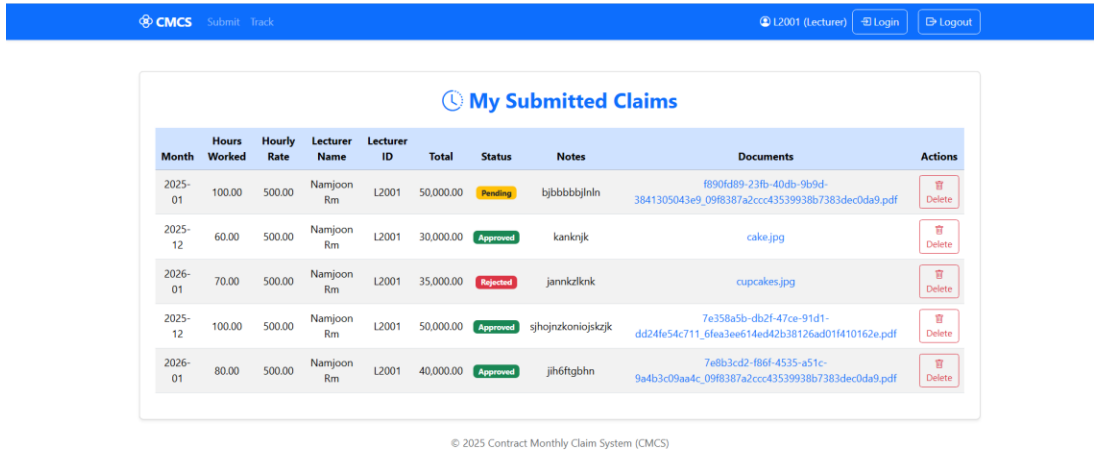


Lecturer Page

Submit Claim Page



Track Claim page



Lecturer Page

Submit Claim

What the Page Shows:

- Input for Month, Hours Worked, Notes
- File upload for evidence (JPEG/PDF/PNG)
- Auto-total calculation
- Rate is added on the HR page
- Name is auto added from HR page
- Limits only to 180 hours per month

Track Claim Page

What the Page Shows:

- Interactive table with claims
- Status badges (Pending, Approved, Rejected)
- Downloadable evidence links
- Delete option for pending claims
- Accurate totals per month

Lecturer Data Pull

LecturerController → SubmitClaim()

Explaintation of Screenshot

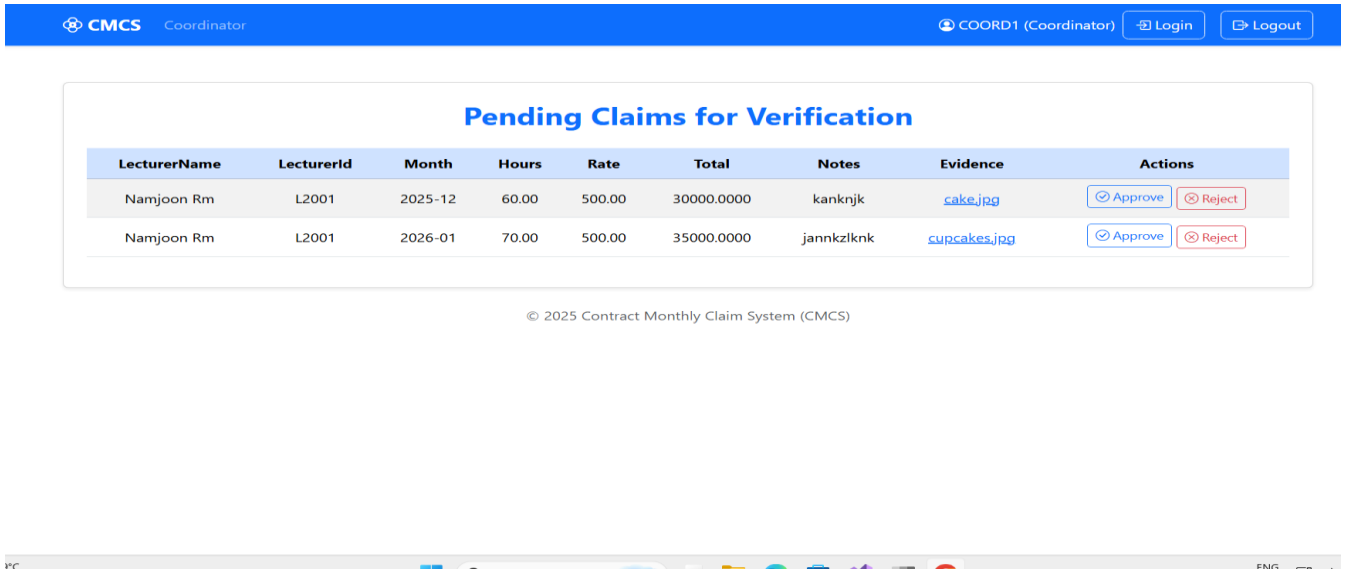
- System fetches Lecturer HourlyRate, Name, ID from DB
- Only HoursWorked + Notes are entered manually

```
public async Task<IActionResult> Submit(SubmitClaimVm vm)
{
    if (!ModelState.IsValid)
    {
        return View(vm);
    }

    try
    {
        var claim = new Claim
        {
            LecturerName = vm.LecturerName,
            LecturerId = HttpContext.Session.GetString("Username"),
            Month = vm.Month,
            HoursWorked = vm.HoursWorked,
            HourlyRate = vm.HourlyRate,
            Notes = vm.Notes,
            Status = ClaimStatus.Pending
        };

        await _service.SubmitClaimAsync(claim, vm.FileUploads);

        TempData["Success"] = "Claim submitted successfully!";
        return RedirectToAction("Track");
    }
    catch (Exception ex)
    {
        ModelState.AddModelError(string.Empty, ex.Message);
        return View(vm);
    }
}

0 references
public IActionResult Submit()
{
    var fullName = HttpContext.Session.GetString("FullName");
    var lecturerId = HttpContext.Session.GetInt32("UserId");
    var rate = HttpContext.Session.GetString("HourlyRate");

    if (fullName == null || lecturerId == null)
        return RedirectToAction("Login", "Account");

    var vm = new SubmitClaimVm
    {
        LecturerName = fullName,
        LecturerId = lecturerId.Value.ToString(),
        HourlyRate = decimal.Parse(rate ?? "0")
    };

    ModelState.Clear();
    return View(vm);
}
```

Coordinator Page

Features :

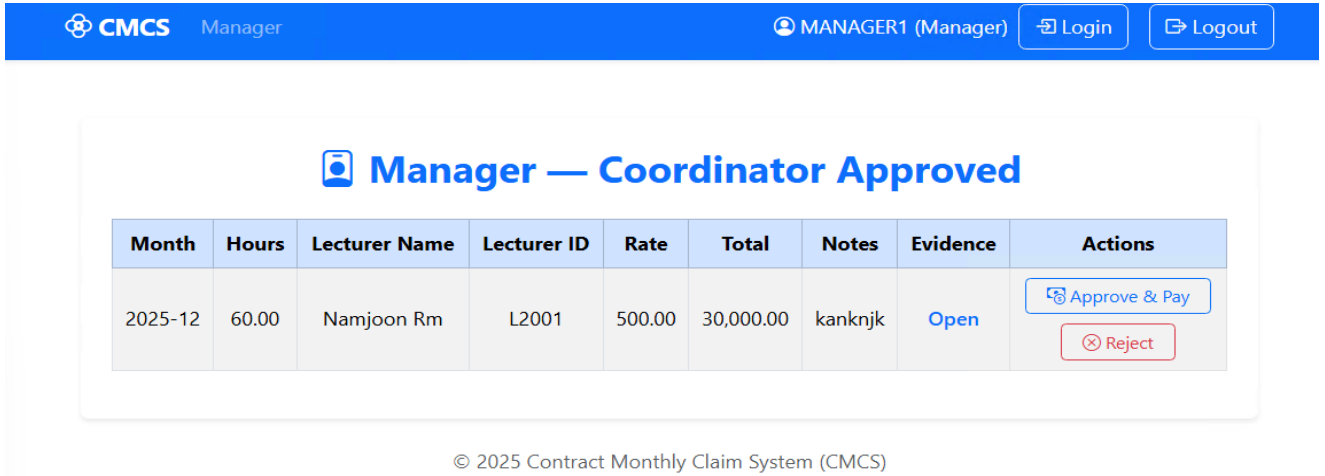
- Holds the pending claims from the Lecturer
- Approve or Reject
- Notes & document preview
- Clearly labelled tables



Manager: **Coordinator-Approved Claims**

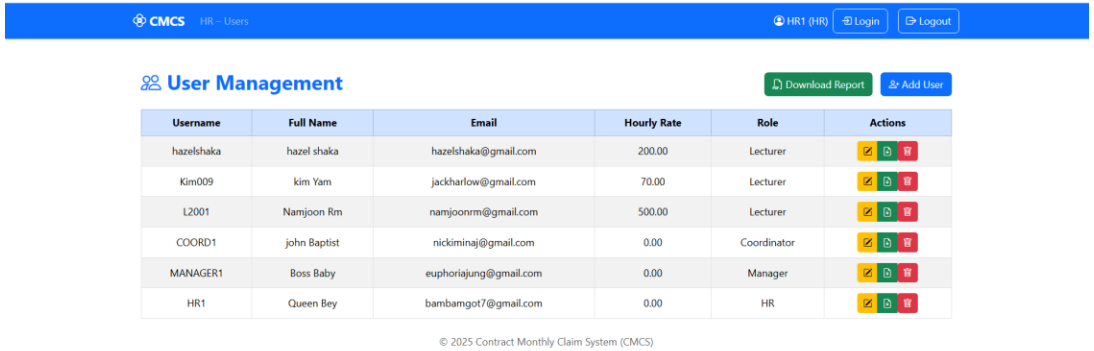
Features

- Hold the approved claims from the coordinator
- Approve & Pay
- Reject with a click
- Access to documents and notes

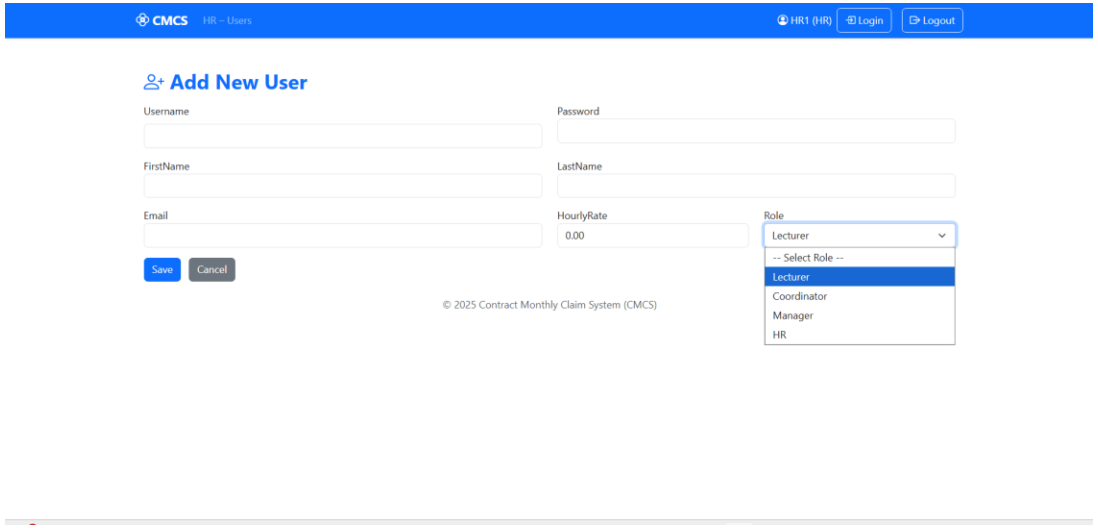


HR Page

HR Index Page



HR Create New User Page



HR Page

HR index page

- Add users
- Edit users
- Delete accounts
- Access PDF reports

HR Create New User

- Create username
- Add password
- Add the users rate
- The users email
- Adding the role

HR Creates & Manages All Users

Screenshot your HRController.cs → Create() method

```
[HttpGet]
0 references
public IActionResult CreateUser()
{
    return View(new AppUser());
}


[HttpPost]
[ValidateAntiForgeryToken]
0 references
public async Task<IActionResult> CreateUser(AppUser model)
{
    if (!ModelState.IsValid)
        return View(model);

    if (string.IsNullOrWhiteSpace(model.PasswordHash))
    {
        ModelState.AddModelError("PasswordHash", "Password is required.");
        return View(model);
    }

    _db.Users.Add(model);
    await _db.SaveChangesAsync();

    TempData["Success"] = "User created successfully!";
    return RedirectToAction(nameof(Index));
}
```

Screenshot the EF Model AppUser.cs

```
CMCS | CMCS.Models.AppUser | FirstName
1   using System.ComponentModel.DataAnnotations;
2
3   namespace CMCS.Models
4   {
        16 references
5       public class AppUser
6       {
7
8           [Key]
            11 references
9           public int Id { get; set; }
10
11          [Required, MaxLength(50)]
            15 references
12          public string Username { get; set; } = string.Empty;
13
14          [Required, MaxLength(200)]
            8 references
15          public string PasswordHash { get; set; } = string.Empty;
16
17          [Required, MaxLength(50)]
            12 references
18          public string FirstName { get; set; } = string.Empty;
19
20          [Required, MaxLength(50)]
            12 references
21          public string LastName { get; set; } = string.Empty;
22
23          [Required, EmailAddress]
            12 references
24          public string Email { get; set; } = string.Empty;
25
26          [Range(0, 2000)]
            12 references
27          public decimal HourlyRate { get; set; }
28
29          [Required]
            15 references
30          public UserRole Role { get; set; }
31      }
32  }
33
```

Explaintation of Screenshot

- Shows HR creating users with name, username, role, hourly rate
- Password hashing or default password assignment
- Role assignment logic

Pdf generated by HR

User Report 1 / 1 100%

1

CMCS - User Report

ID	Username	First Name	Last Name	Email	Rate
1	L2001	Namjoon	Rm	namjoonrm@gmail.com	500.00
9	Kim009	kim	Yam	jackharlow@gmail.com	70.00
10	hazelshaka	hazel	shaka	hazelshaka@gmail.com	200.00

HR Generates Reports (PDF)

The GeneratePdf() method

```
0 references
public async Task<IActionResult> GeneratePdf()
{
    var lecturers = await _db.Users
.Where(u => u.Role == UserRole.Lecturer)

.Select(u => new AppUser
{
    Id = u.Id,
    Username = u.Username,
    Email = u.Email,
    Role = u.Role,

    FirstName = u.FirstName,
    LastName = u.LastName,
    HourlyRate = u.HourlyRate
})
.ToListAsync();


    var vm = new ReportVm
    {
        Users = lecturers
    };

    var pdfBytes = new ReportDocument(vm).GeneratePdf();

    return File(pdfBytes, "application/pdf", "LecturerReport.pdf");
}

}
}
```

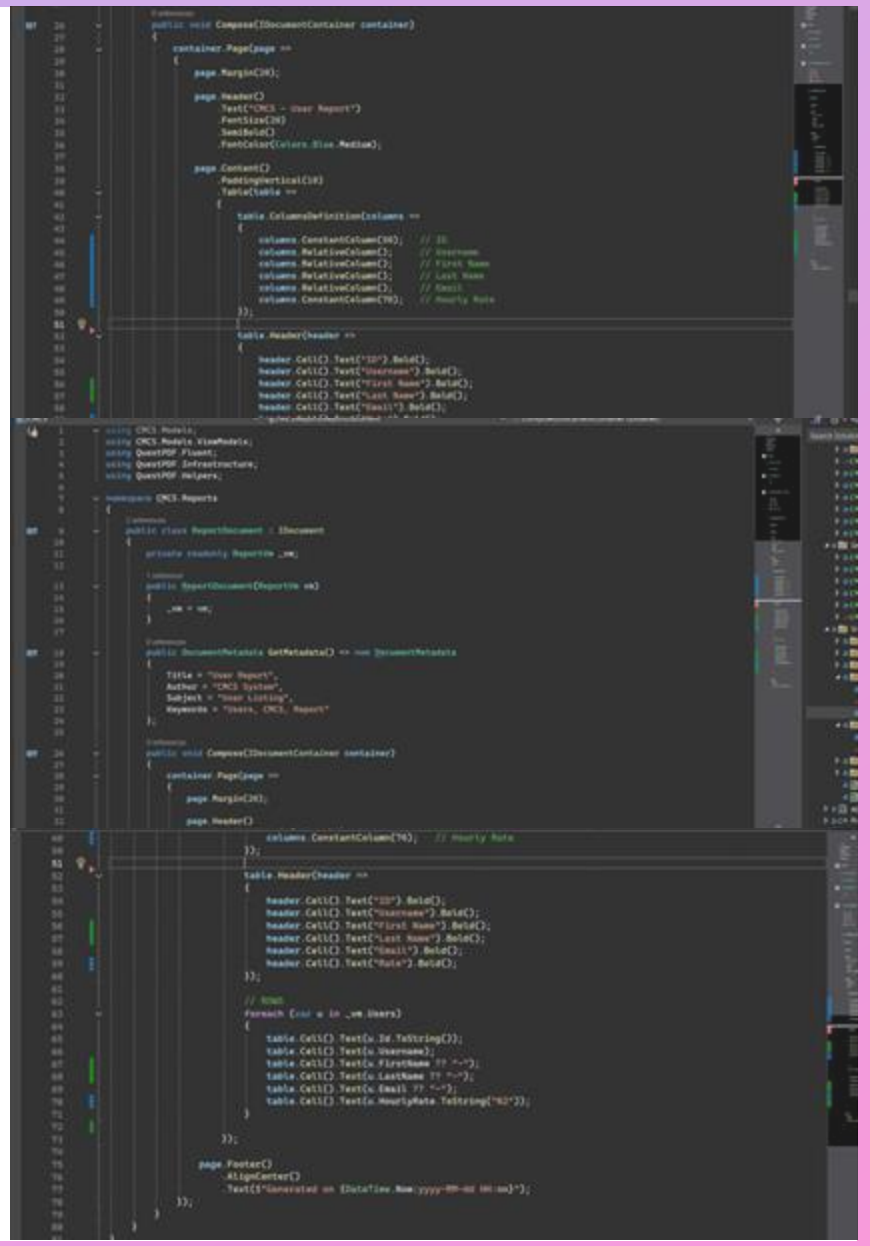
Explaintation of Screenshot

- System loads all users or claims
- Renders them into a structured PDF
- HR can download with one click

The QuestPDF document layout class

Explaintation of Screenshot

- System loads all users or claims
- Renders them into a structured PDF
- HR can download with one click



Full System Demonstration Summary

The CMCS System has :

- Full claim workflow from submission to payment
- Multi-role approval chain
- Secure authentication
- Evidence management
- PDF reporting
- Robust search
- Clean, user-friendly interface