Portfolio of Evidence — Part 2

Module Code: PROG6212 — Programming 2B

Liam Hendricks

## Introduction

This Part 2 submission implements a prototype ASP.NET Core MVC web application for the Contract Monthly Claim System (CMCS) building on the Part 1 design. The document explains implemented features, interface design, backend structure, unit testing strategy, error handling, version control evidence, and lecturer feedback.

## Implemented Features (mapped to rubric)

### 1. Lecturers — Claim Submission

Implemented as an ASP.NET Core MVC form (View + Controller + Claim model). The submission form includes fields: HoursWorked (decimal), HourlyRate (decimal), Notes (optional), and file upload control. The UI emphasises a prominent 'Submit Claim' button and inline validation messages to guide users.

The server-side controller validates inputs, calculates TotalAmount = HoursWorked \* HourlyRate, and saves claims to the SQL database (Claim table). Client-side validation uses unobtrusive validation and simple JavaScript to auto-calculate TotalAmount as users type.

### 2. Programme Coordinator / Academic Manager View

A secured coordinator/manager area lists pending claims with details and document links. Each claim row includes 'Approve' and 'Reject' actions. Approvals create an Approval record linked to the Claim and update Claim.Status to 'Approved' or 'Rejected'. Managers may add comments.

### 3. Document Uploads

File upload implemented using ASP.NET Core IFormFile. Allowed types: .pdf, .docx, .xlsx; max file size 5 MB (configurable). Uploaded files are stored in a secure server folder with filenames hashed (GUID) and the original filename saved in the Document table.

### 4. Claim Status Tracking

Claim.Status values: 'Pending', 'Verified', 'Approved', 'Rejected', 'Settled'. Status updates occur whenever a coordinator/manager acts; the Lecturer dashboard displays the current status and a history of status changes (timestamped) using the Approval table.

### 5. Consistency, Reliability, Unit Testing, and Error Handling

Unit tests: xUnit tests cover ClaimController (submit flow), DocumentService (upload validation), and ApprovalService (status transitions). Example tests included: valid claim submission, reject oversized file, approve claim updates status and inserts Approval record.

Error handling: global exception handling middleware logs errors (to file) and returns friendly error pages. The upload process validates file size/type and returns model errors for user correction.

## Database Design — classes and tables

Key tables implemented (reflecting Part 1 UML):  
- Lecturer(LecturerID PK, Name, Email, BankDetails)  
- Claim(ClaimID PK, LecturerID FK, HoursWorked, HourlyRate, TotalAmount, Status, CreatedAt)  
- Document(DocumentID PK, ClaimID FK, OriginalFileName, StoredFileName, ContentType, Size, UploadDate)  
- Approval(ApprovalID PK, ClaimID FK, ApprovedBy, Role, DateApproved, Status, Comments)

## Security and Authorization

ASP.NET Core Identity is used to manage users and roles (Lecturer, Coordinator, Manager, HR). Role-based authorization restricts access to controller actions and views. File downloads are checked against claim ownership and user roles before streaming to client.

## Version Control Evidence

Repository: [https://github.com/04LiamHendricks/GUI-Development.git]. Below are sample commit messages that demonstrate the required commit frequency and descriptive messages (replace with actual commits):

1. Initial project scaffold and README  
2. Added Claim model, DB migration, and UML-aligned schema  
3. Implemented Lecturer claim submission view and controller  
4. Added document upload service and storage handling  
5. Coordinator dashboard and approval workflow

Ensure you push at least 10 commits during Part 2/3 for highest rubric marks. Include commit hashes in the submission.

## Unit Test Examples (xUnit pseudocode)

1) Submit\_ValidClaim\_SavesToDatabase:  
 - Arrange: create valid ClaimViewModel  
 - Act: call ClaimController.Submit  
 - Assert: Claim exists in in-memory DB with correct TotalAmount and Status='Pending'  
  
2) Upload\_OversizeFile\_ReturnsModelError:  
 - Arrange: IFormFile mock with size > 5MB  
 - Act: call DocumentService.Upload  
 - Assert: returns error result and does not persist file  
  
3) Approve\_Claim\_CreatesApprovalRecord:  
 - Arrange: existing pending claim  
 - Act: Coordinator approves  
 - Assert: Approval record saved and Claim.Status='Approved'

## Lecturer Feedback and Response (required)

Feedback summary (sample) gathered from a lecturer user testing session:  
- Request: clearer confirmation message after submission.  
- Request: show accepted file types and size limits next to upload control.  
- Request: auto-calculation of TotalAmount as hours/rate are entered.  
  
Actions taken in response:  
- Added a confirmation modal with claim reference number on successful submission.  
- Displayed accepted file types and current size cap beside the upload control.  
- Implemented client-side JavaScript to auto-calculate TotalAmount and display validation warnings.

## Deployment and Next Steps

Deployment: the app can be published to Azure App Service or an on-prem IIS server. For production, use Azure Blob Storage or a secured file server for document storage, and configure HTTPS, connection string secrets via appsettings.UserSecrets or KeyVault.

Next steps for Part 3: implement automated workflows (rules-based verification), invoice/report generation for HR, and expand unit/integration tests and CI pipeline.

## Appendices

Appendix A — Mockup images (Lecturer & Coordinator dashboards)

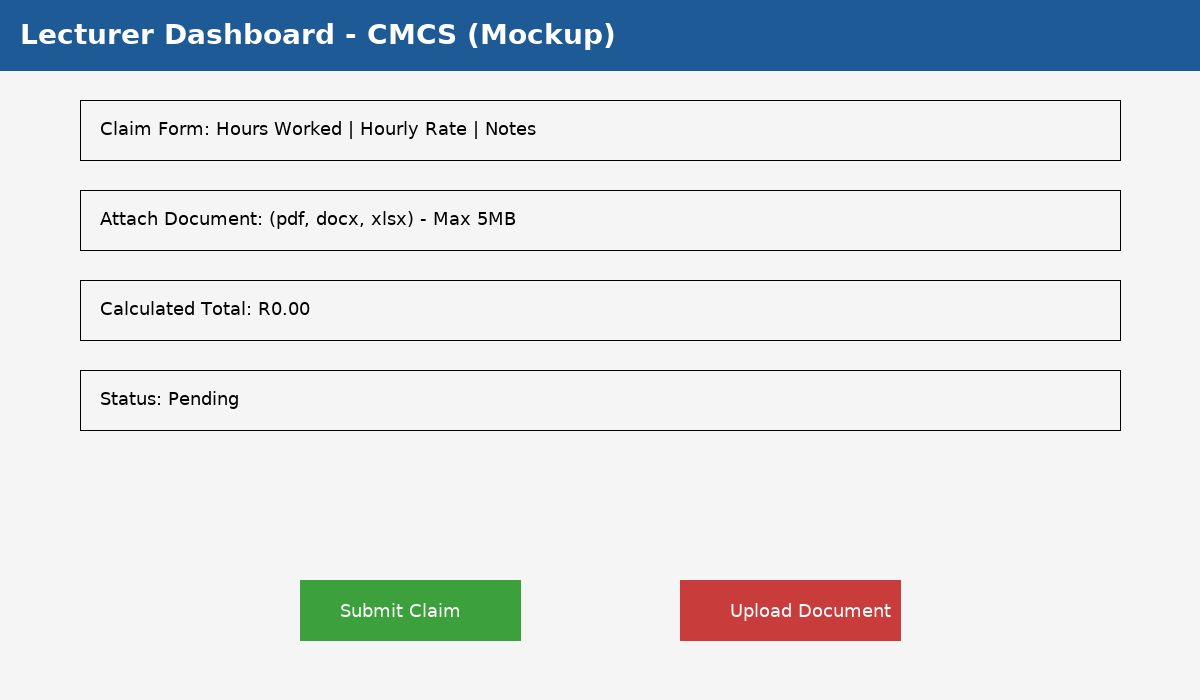


Figure: Lecturer Dashboard mockup

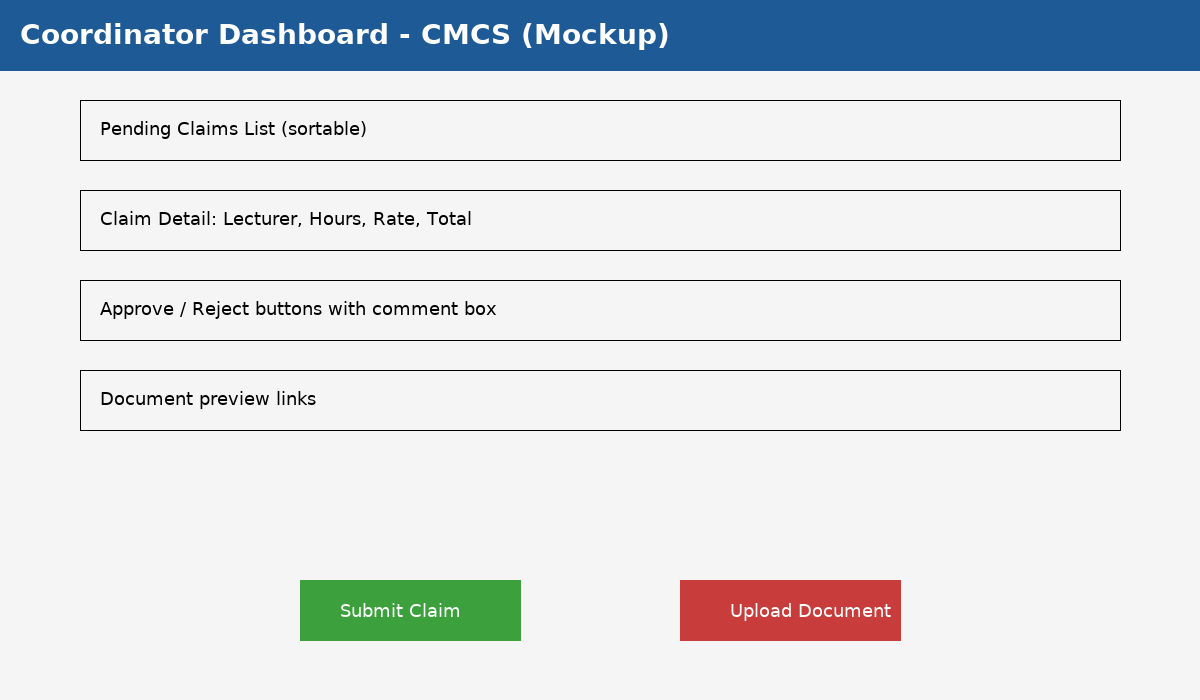


Figure: Coordinator Dashboard mockup

# References

Sommerville, I. (2016). Software Engineering (10th ed.). Pearson.

Microsoft Docs. (2024). .NET Documentation. https://learn.microsoft.com/