POE Part 1

# Programming 2B

PROG6212

ST10061509

Mohammed Moosa

[**Programming 2B 1**](#_j9rclgt54umv)

[**Documentation. 3**](#_2d2kjw63v3nh)

[**UML Diagram. 6**](#_u3ihmc6lfbhu)

[**Project Plan. 7**](#_72uzmlnc7az4)

[**GUI. 8**](#_eifeebozxmbb)

[Home Page: 8](#_unxd5njo3utd)

[Lecturer Dashboard: 8](#_cfbxiaet8rvj)

[PC Dashboard: 10](#_s1hghuhzkxle)

[AM Dashboard: 11](#_a0yuk55z8c1)

[**Reference List. 13**](#_emltcqdo73n1)

# **Documentation.**

**Contract Monthly Claim System (CMCS) – Prototype Design.**

In this prototype of the Contract Monthly Claim System (CMCS), a clear and user-friendly interface is provided to support the effective submission of monthly claims and a transparent approval process. The graphical user interface (GUI) will be developed using .NET Core MVC (Razor) to ensure portability, maintainability, and smooth integration with future functionality such as database connectivity. The primary stakeholders are Lecturers, Programme Coordinators (PCs), and Academic Managers (AMs), each with individual views. Claims will be created and tracked by lecturers, verified by PCs, and finally approved by AMs.

**Design Decisions and Justifications.**

The MVC pattern was selected to avoid overlap between user interactions, presentation, and business logic. This architecture separates Views, Controllers, and Models, ensuring that the system is modular and scalable. It also provides compatibility with Entity Framework and Azure SQL for future development.

The prototype is primarily focused on usability. It features a straightforward claim submission process, a file upload option for attaching supporting documents (PDF/JPG), a graphical claim tracking system (Submitted → Verified → Approved → Settled), and intuitive role-based navigation. These design decisions aim to improve accuracy, transparency, and ease of use.

**Assumptions and Constraints.**

Assumptions:

* Users will have unique accounts, although login functionality is not yet implemented.
* Hourly rates are fixed and linked to lecturers rather than manually entered with each claim.
* Supporting documents are assumed to be valid PDF or JPG files.
* Claims are submitted on a monthly basis, not weekly or daily.

Constraints:

* At this stage, only GUI components are developed (no backend logic or database connection).
* Role-based navigation is simulated but does not enforce real security.
* File uploads are represented as placeholders without actual storage.
* Business logic such as calculations, deductions, and reporting is documented but not implemented.

**Database Model Summary.**

The UML class diagram represents the following entities: Lecturer, Claim, ClaimItem, Rate, SupportingDocument, Approval, and Role. Relationships are normalised so that a single claim may have multiple ClaimItems and SupportingDocuments. Decisions made by PCs and AMs are recorded in Approval records, ensuring accountability and auditability.

**Trustworthiness and Informational Consistency.**

Forms will include key fields such as claim period, total hours, and rate reference. Calculated totals will be displayed as read-only fields, while a status tracker will provide visibility of progress. Consistent labelling, structured navigation, and input validation will ensure reliable and easy-to-use data.

**Prototype Scope.**

The output is a non-functional front-end prototype that includes:

* A claim creation page with submission flow.
* File upload mockups.
* Approval dashboards for PCs and AMs.
* A status-tracking interface.

Version control will be maintained with at least five descriptive commits, and the project will be hosted on GitHub.

**Graphical User Interface Layout Overview.**

Lecturer Dashboard: The primary menu features options such as Submit Claim, My Claims, and Upload Supporting Documents. The submission form comprises fields for the claim period, hours worked, and rate, along with an upload button.

Programme Coordinator Dashboard: This section presents a list of submitted claims equipped with filter/search capabilities, buttons for Verify or Return for Correction, and a detailed view of each claim.

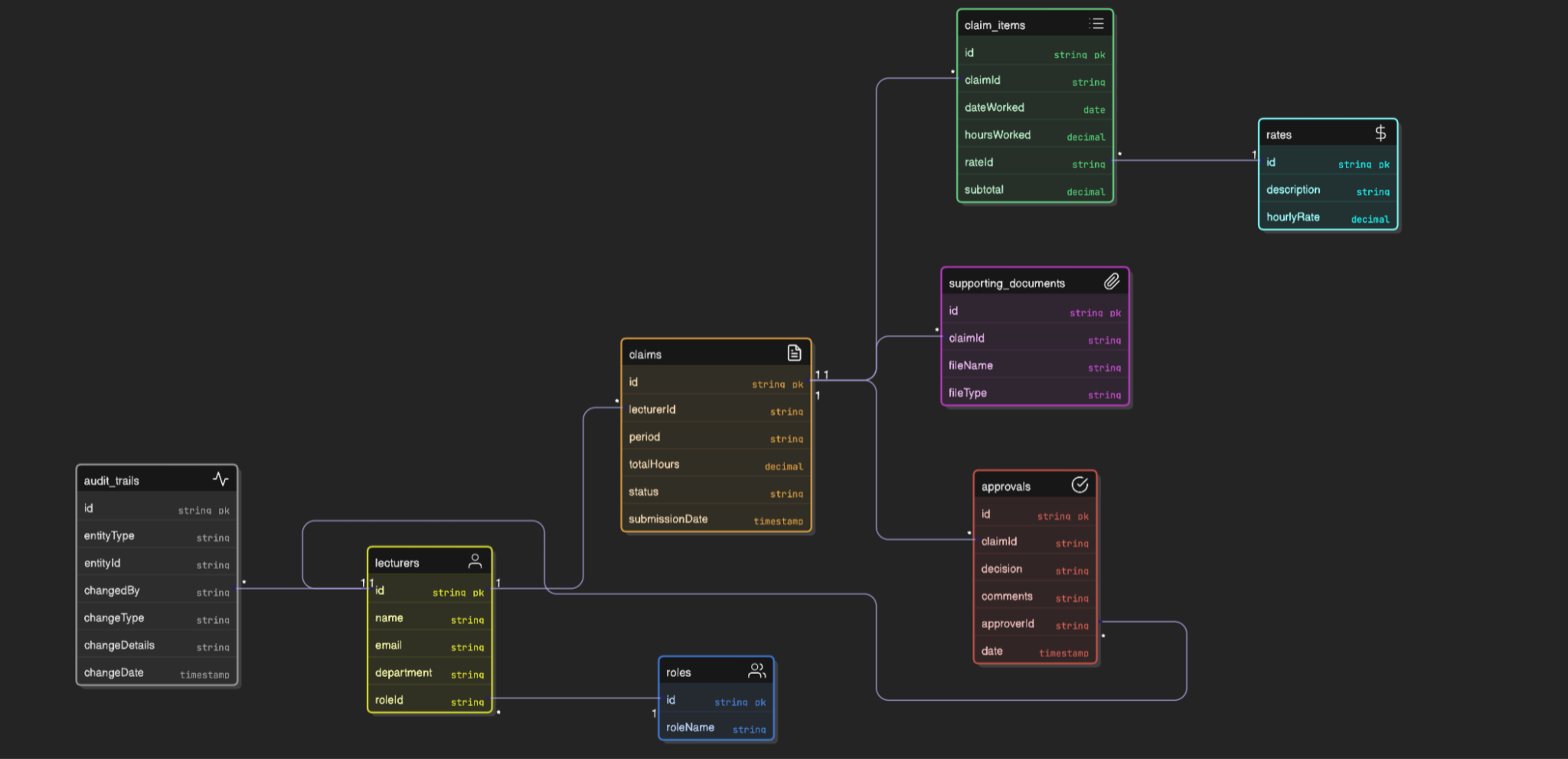
Academic Manager Dashboard: This dashboard mirrors the previous list but includes options to Approve or Reject claims, along with the ability to add comments.

Status Tracking Page: This page illustrates the status of claims through a progress bar, indicating the stages from Submitted to Verified, Approved, and finally Settled.

**Conclusion.**

This prototype achieves a balance of clarity, extensibility, and transparency. It defines the data model, workflows, and UI processes required for effective claims management and structured approvals while leaving room for future enhancement. In later POE stages, this MVC-based front-end will be developed into a fully functional application with database integration, authentication, and secure file storage.

# **UML Diagram.**



# **Project Plan.**

The project will be executed through a structured development plan to ensure successful delivery:

**Phase 1: Planning (Week 1)**

* Review system requirements.
* Identify stakeholders and user roles.
* Define assumptions, constraints, and project scope.

**Phase 2: Design (Weeks 2–3)**

* Develop UML diagrams (use case and class diagrams).
* Create wireframes and GUI layouts for dashboards and forms.
* Define role-based access simulation and navigation flow.

**Phase 3: Prototype Development (Weeks 4–6)**

* Implement the GUI using .NET Core MVC (Razor).
* Develop claim creation and tracking pages.
* Build test interfaces for file uploads and approval processes.
* Conduct front-end usability and navigation testing.

**Phase 4: Documentation and Version Control (Ongoing)**

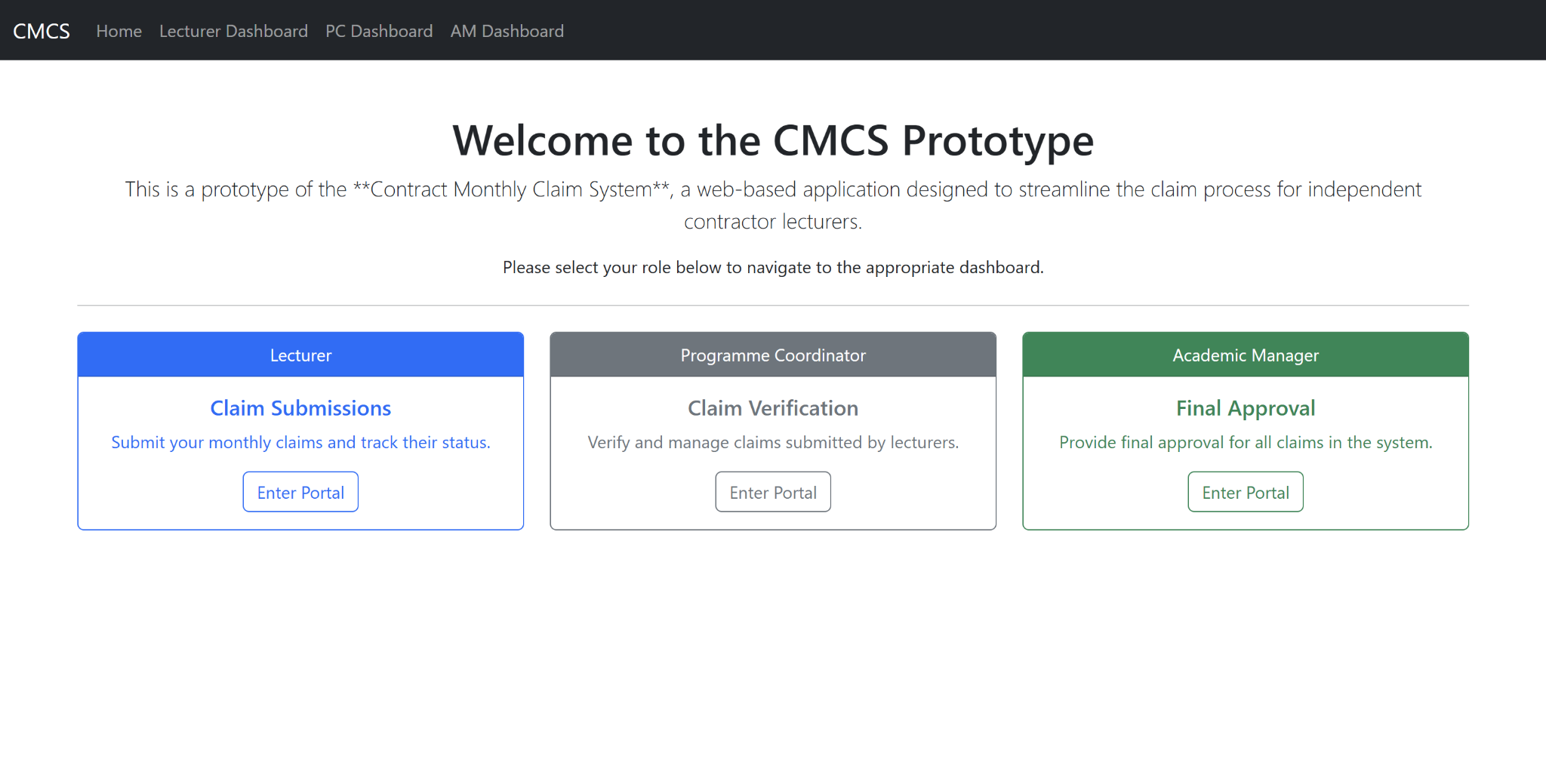
* Maintain a GitHub repository with a minimum of five descriptive commits.
* Document the database model, GUI layouts, and design decisions.

**Phase 5: Submission and Review (Final Week)**

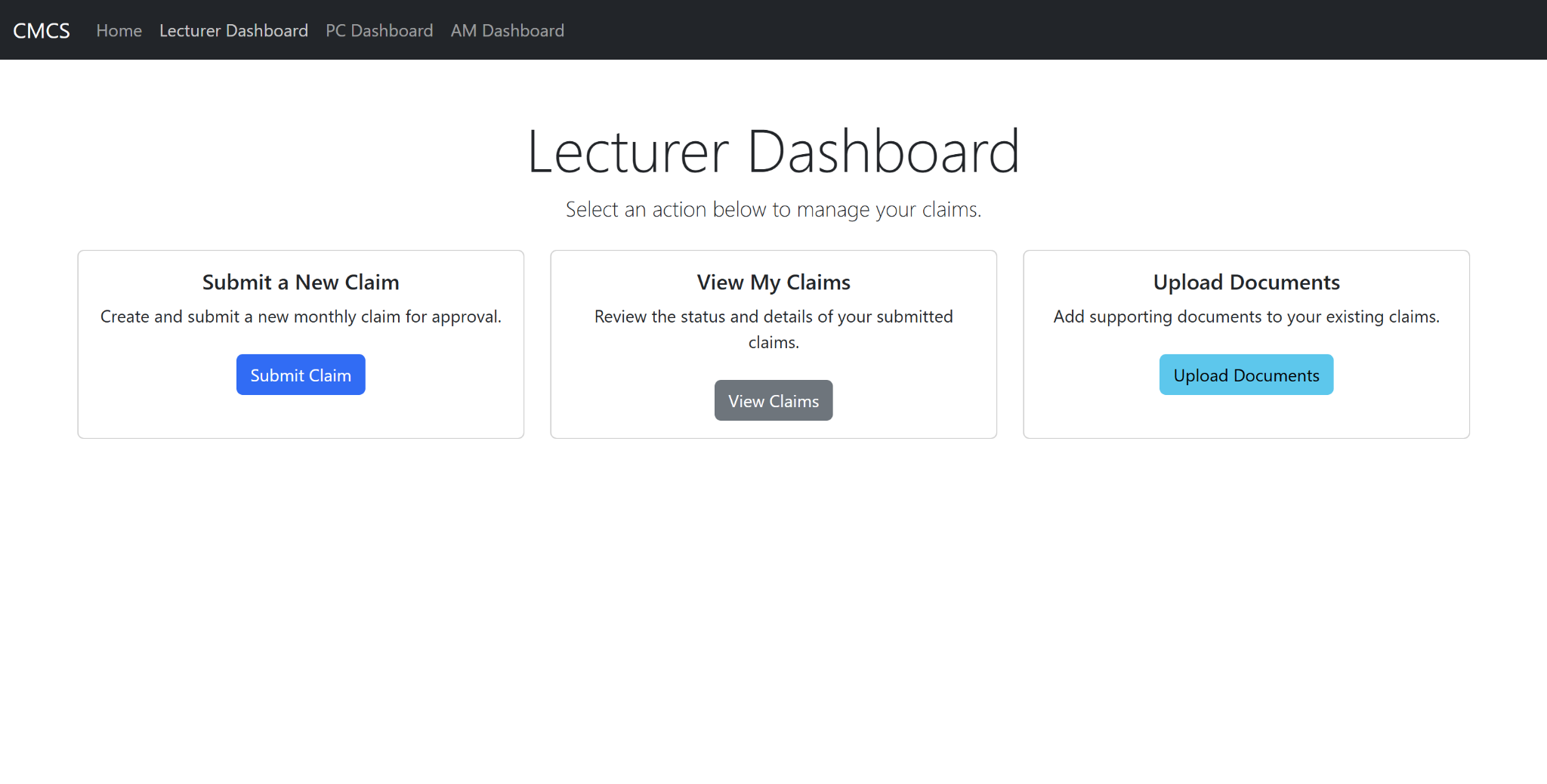
* Finalize the non-functional prototype.
* Compile the report including design, assumptions, GUI layout, and project plan.
* Submit the completed work for evaluation.

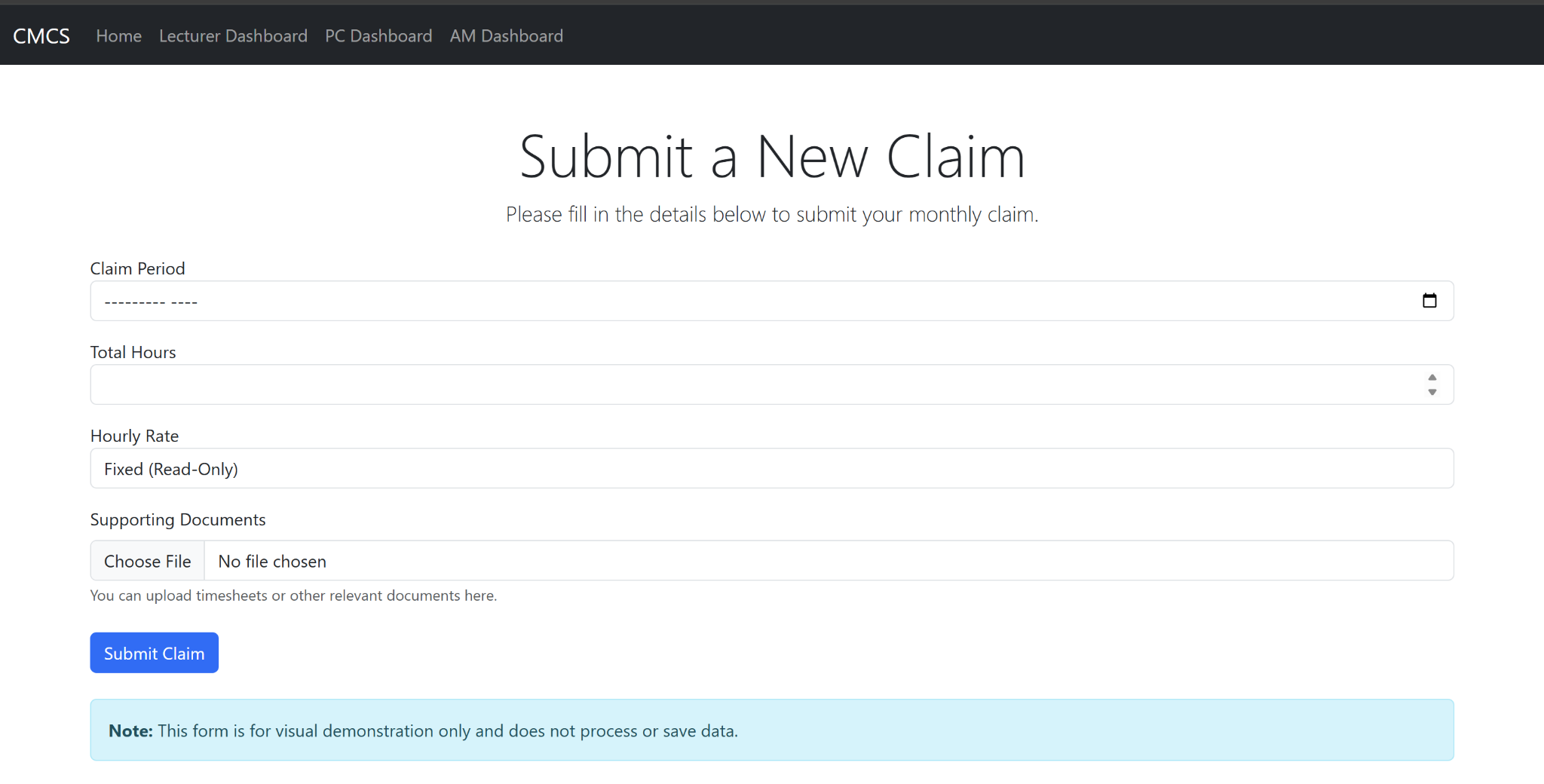
# **GUI.**

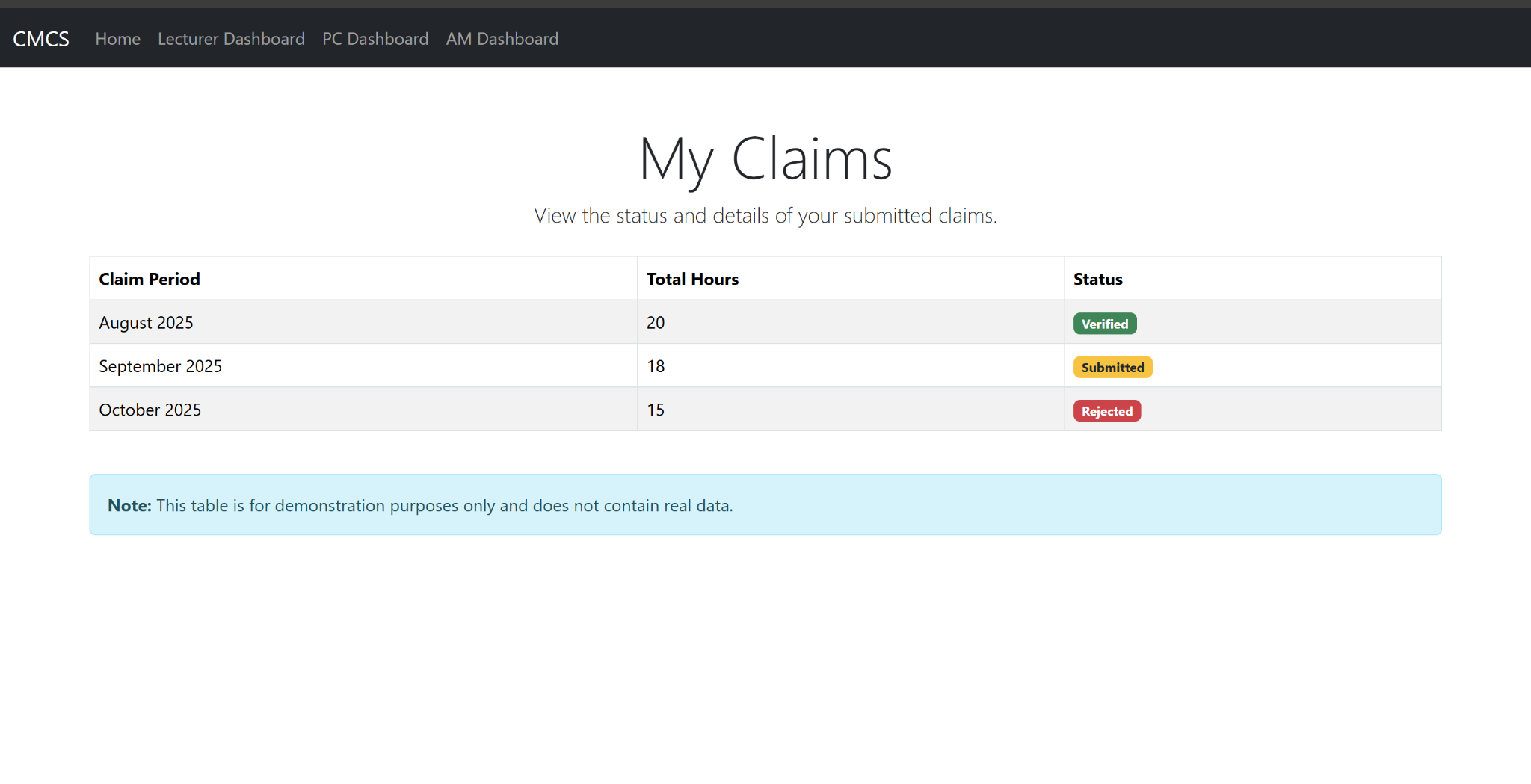
## **Home Page:**

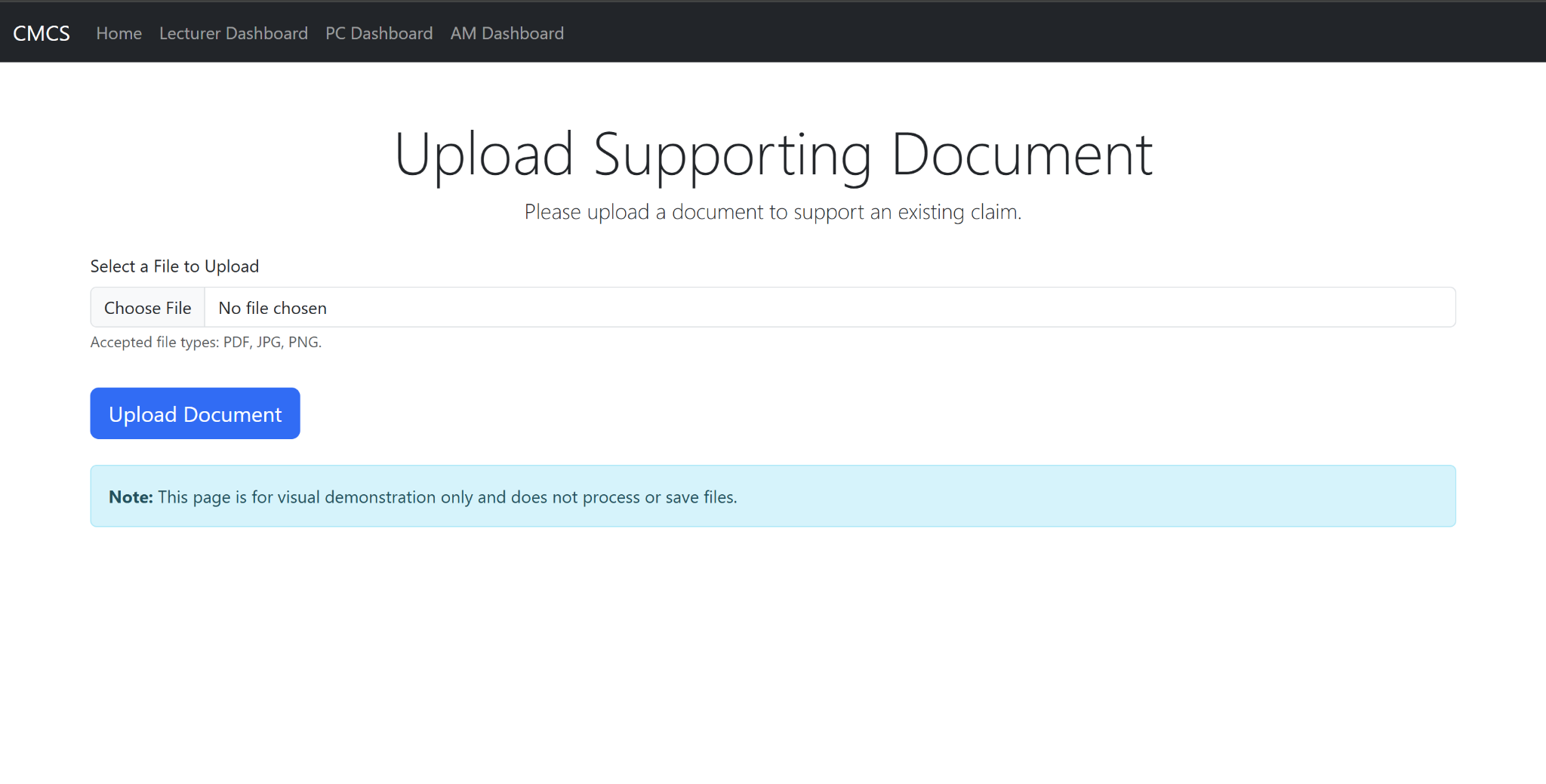
****

## **Lecturer Dashboard:**

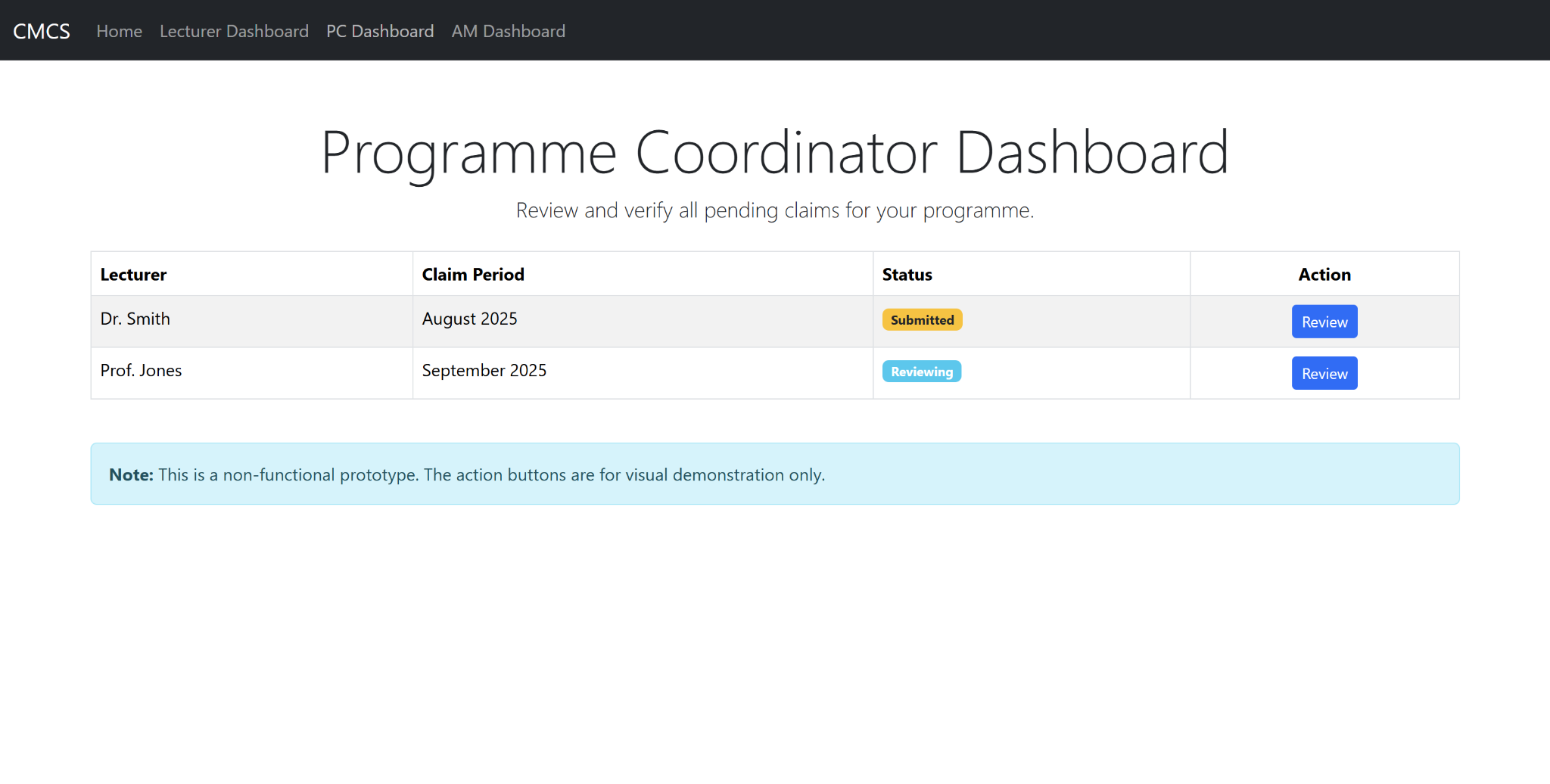
****

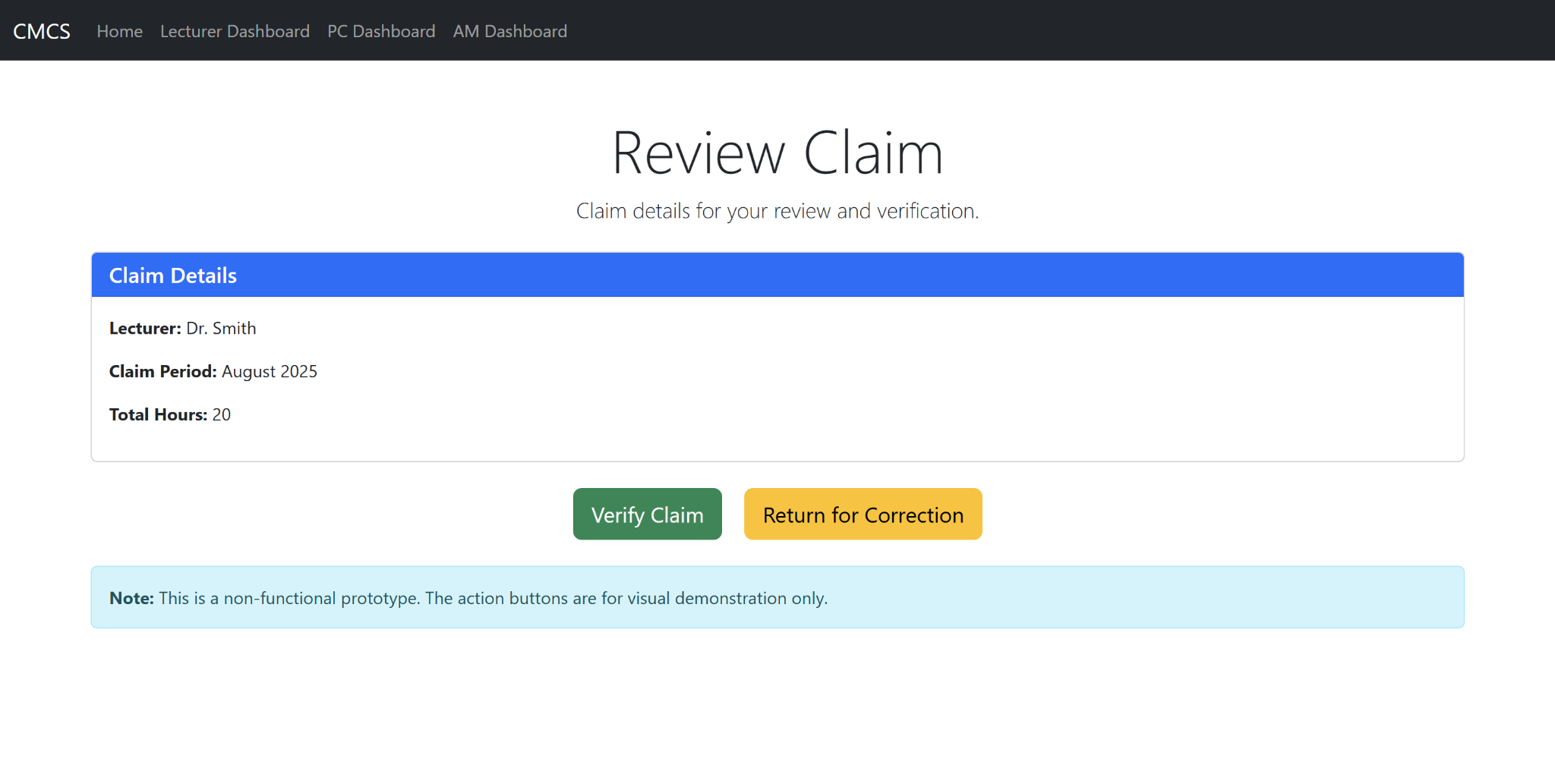
****

****

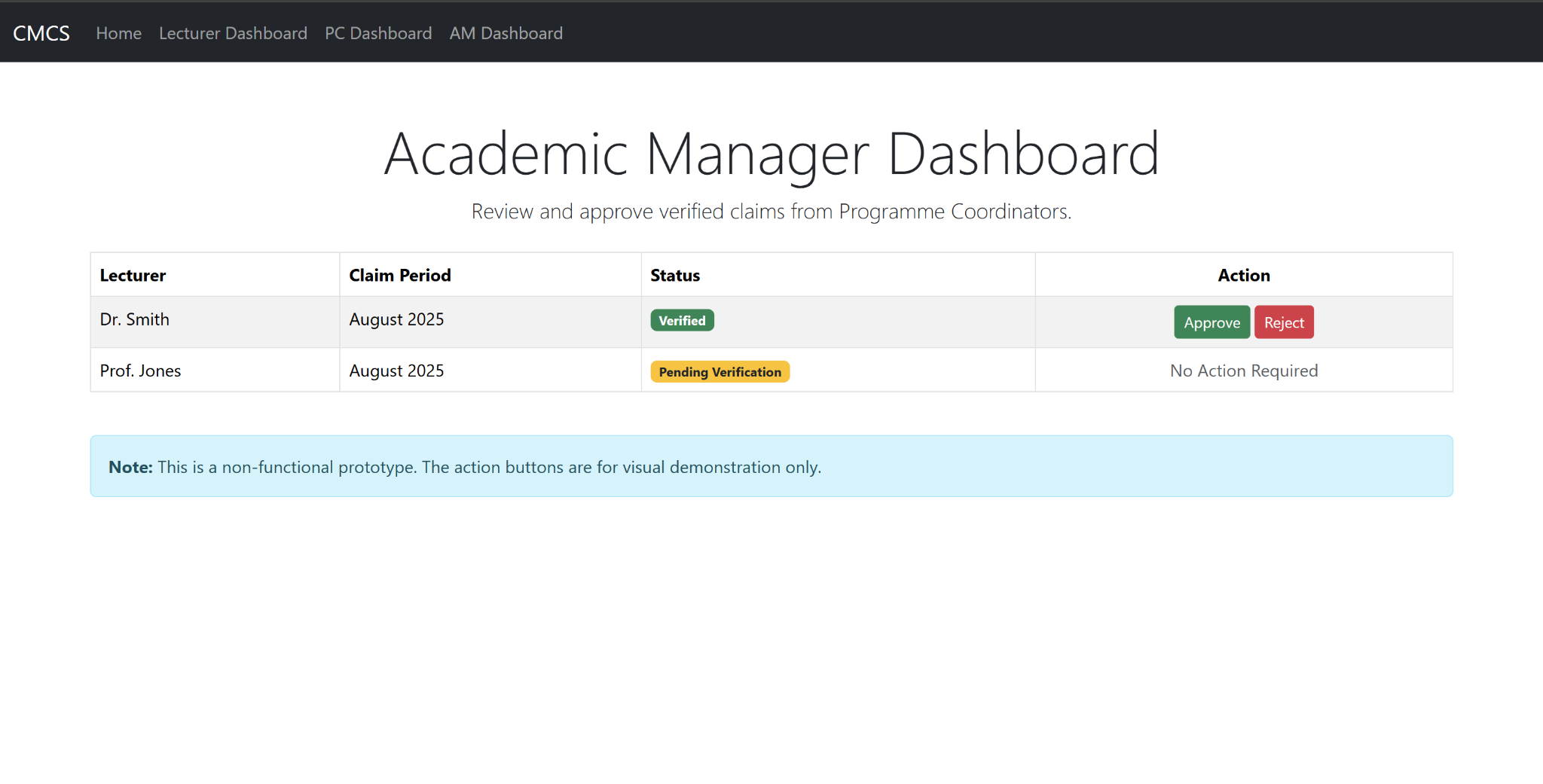
****

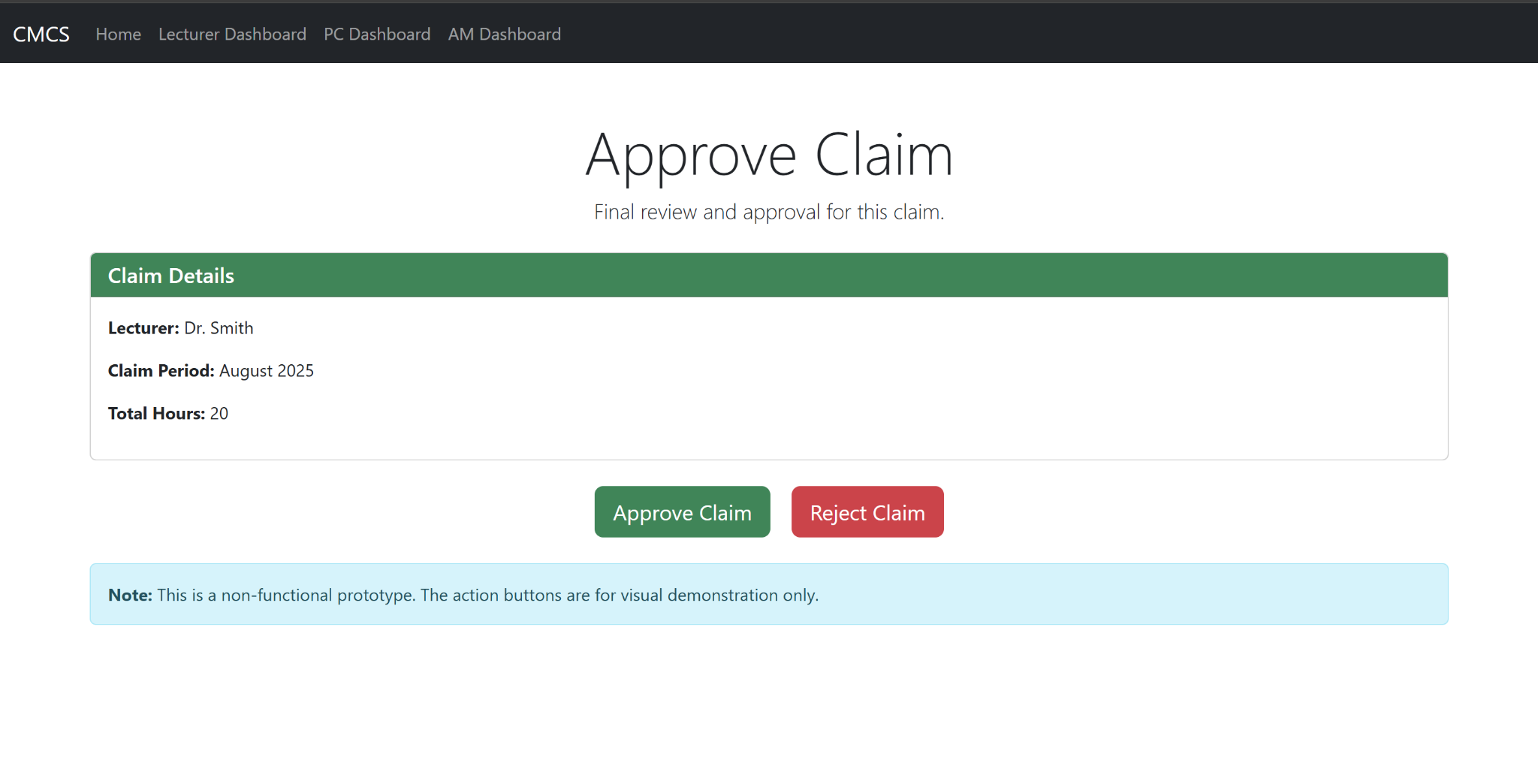
## **PC Dashboard:**





## **AM Dashboard:**





# **Reference List.**

Microsoft Docs (2025) ‘Overview of ASP.NET Core’. Available at: https://learn.microsoft.com/en-us/aspnet/core/overview (Accessed: 17 September 2025).

Microsoft Docs (2025) ‘MVC in ASP.NET Core’. Available at: https://learn.microsoft.com/en-us/aspnet/core/mvc/overview (Accessed: 17 September 2025).

Microsoft Docs (2025) ‘Introduction to Razor Pages in ASP.NET Core’. Available at: https://learn.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-9.0 (Accessed: 17 September 2025).

Lucidchart Blog (2021) ‘UML Class Diagrams Explained with Examples’. Available at: https://www.lucidchart.com/blog/uml-class-diagram (Accessed: 17 September 2025).

ProjectManager.com (2020) ‘How to Create a Project Plan: Steps and Templates’. Available at: https://www.projectmanager.com/project-plan (Accessed: 17 September 2025).

Guru99 (2021) ‘MVC Architecture in .NET’. Available at: https://www.guru99.com/mvc-tutorial.html (Accessed: 17 September 2025).

Tutorialspoint (2025) ‘.NET Core - Introduction’. Available at: https://www.tutorialspoint.com/asp.net\_core/asp.net\_core\_introduction.htm (Accessed: 17 September 2025).

Interaction Design Foundation (2020) ‘Principles of Good User Interface (UI) Design’. Available at: https://www.interaction-design.org/literature/topics/ui-design (Accessed: 17 September 2025).