9/10/2024

Sashiel Moonsamy

ST10028058

pROg6212 POE PArt 1

Sashiel Moonsamy

ST10028058

PROG6212

POE Part 1

Documentation

**Documentation**

**1.Design Choices**

The design of the Contract Monthly Claim System (CMCS) was driven by the need to simplify and streamline the process of claim submissions and approvals for Independent Contractor (IC) lecturers. Each design decision was made with careful consideration of scalability, security, and ease of use, ensuring that the system meets real-world business needs effectively.

**System Architecture**

(“Effective Project Documentation Best Practice,” n.d.)

I chose the **Model-View-Controller (MVC)** architecture for this system because it enforces a clear separation of concerns, making the application more modular, easier to maintain, and scalable. Here's why:

* **Model:** Manages the business logic and application data. It ensures that all operations, such as claim calculations, document uploads, and user authentication, are handled efficiently. I chose to separate this from the view and controller to make future adjustments (like adding new types of users) easy to implement without affecting the entire system.
* **View:** Manages the user interface, where I opted for simplicity to enhance user experience. Since lecturers, coordinators, and managers will use the system, the design must cater to users with varying technical skills. I focused on creating an intuitive, easy-to-navigate interface.
* **Controller:** Handles requests from the user, communicates with the model, and updates the view. This decoupling from the model ensures that user input (e.g., claim submissions) is processed without compromising the system's core business logic.

**Why MVC?**  
The MVC architecture ensures that the CMCS can evolve and scale as more lecturers, claims, and functionalities are added. By separating concerns, it becomes easier to update individual components, reducing the risk of bugs and improving maintainability.

**Scalability and Flexibility**

In a system like CMCS, scalability is a crucial factor. As the number of lecturers, claims, and document uploads grows, the system must handle increasing data loads without performance degradation. The MVC architecture naturally supports scalability because it separates the core components:

* **Model**: The business logic and database interactions are independent of the user interface, which means the system can be scaled at the backend (e.g., adding more servers or optimizing databases) without requiring changes to the frontend.
* **View**: The user interface (UI) can be scaled or changed independently to meet growing demands, for instance, by adding new features like reports or dashboards, without affecting the backend.

**Security Considerations**

**Why Security Is Essential?**  
Since the system handles sensitive data, including financial claims and lecturer details, it is essential to implement security measures such as password hashing and encrypted communication. For this reason, I opted for:

* **Hashed Passwords**: All passwords are securely stored using a hashing algorithm to prevent unauthorized access in case of a data breach.
* **Data Validation and Sanitization**: To ensure that all inputs are safe, reducing the risk of SQL injections or malicious data entries.

**2. Database Structure Description**

(“Effective Project Documentation Best Practice,” n.d.)

The database structure for the Contract Monthly Claim System (CMCS) is designed to handle the submission, approval, and processing of monthly claims for independent contractor lecturers. This structure includes key entities and relationships to support various stakeholders such as Lecturers, Programme Coordinators, Academic Managers, and Human Resources. Each of these roles interacts with claims in different ways to ensure that the claims are efficiently managed from submission to approval. (“Database - Overview, Roles and Components, DBMS,” n.d.)

**Tables and Key Fields:**

1. **User**
   * **Key Fields:**
     + **User\_ID:** Primary Key (PK) that uniquely identifies each user.
     + **FirstName, LastName, Email, Password:** Shared personal and login details across all user types.
   * **Description:** The User table acts as a base class for all user types in the system, including Lecturers, Programme Coordinators, Academic Managers, and HR staff. This allows shared attributes like name, email, and password to be managed consistently.
2. **Lecturer (Inherits from User)**
   * **Key Fields:**
     + **LecturerID:** Primary Key (PK) identifying each lecturer.
     + **Department, Date\_Joined:** Additional attributes specific to lecturers.
   * **Description:** This table stores information specific to lecturers, who are responsible for submitting claims. Each lecturer can submit multiple claims, and they can track their claim status through the system.
3. **ProgrammeCoordinator (Inherits from User)**
   * **Key Fields:**
     + **CoordinatorID:** Primary Key (PK) for each Programme Coordinator.
   * **Description:** Programme Coordinators review claims submitted by lecturers. They manage multiple claims and decide whether to approve or reject them before sending them to Academic Managers.
4. **AcademicManager (Inherits from User)**
   * **Key Fields:**
     + **ManagerID**: Primary Key (PK) identifying each Academic Manager.
   * **Description:** Academic Managers provide additional oversight on claims that have been reviewed by Programme Coordinators. They have the authority to approve or reject claims before they are processed by HR.
5. **HumanResources (HR) (Inherits from User)**
   * **Key Fields:**
     + **HR\_ID:** Primary Key (PK) for each HR staff member.
   * **Description:** The HR department is responsible for processing claims that have been approved by the Academic Manager. HR manages the final step in the claim lifecycle by ensuring the claim is processed for payment and stored for record-keeping.
6. **Claim**
   * **Key Fields:**
     + **ClaimID:** Primary Key (PK) for each claim.
     + **LecturerID:** Foreign Key (FK) linking the claim to the lecturer who submitted it.
     + **CoordinatorID:** Foreign Key (FK) linking the claim to the Programme Coordinator reviewing it.
     + **ManagerID:** Foreign Key (FK) linking the claim to the Academic Manager approving it.
     + **HR\_ID:** Foreign Key (FK) linking the claim to the HR staff member processing it.
     + **HoursWorked, HourlyRate, TotalAmount:** Key fields for calculating the claim's total amount.
     + **Status:** Current status of the claim (submitted, approved, rejected).
     + **DateSubmitted:** Date the claim was submitted.
   * **Description:** This table records each claim submitted by lecturers. Claims are routed through Programme Coordinators, Academic Managers, and HR for approval and processing. It includes the calculation of the claim amount based on the hours worked and hourly rate.
7. **SupportingDocument**
   * **Key Fields:**
     + DocumentID: Primary Key (PK) for each document.
     + ClaimID: Foreign Key (FK) linking the document to its associated claim.
     + DocumentName, DateUploaded: Information about the document.
   * **Description:** This table stores documents that support claims, such as contracts or timesheets. Each claim can have multiple supporting documents attached, which can be uploaded by the lecturer and accessed by reviewers.

**Relationships and Multiplicities:**

1. **User to Lecturer/Coordinator/Manager/HR:**
   * The User table is inherited by the Lecturer, ProgrammeCoordinator, AcademicManager, and HumanResources tables. Each user type has its own role in the system and its own specific attributes.
   * Multiplicity: User 1..1 ⟶ Lecturer/Coordinator/Manager/HR 1..1
2. **Lecturer to Claim:**
   * A lecturer can submit multiple claims, but each claim is linked to a single lecturer.
   * Multiplicity: Lecturer 1..\* ⟶ Claim 1
3. **ProgrammeCoordinator to Claim:**
   * A Programme Coordinator can review multiple claims, but each claim is reviewed by only one coordinator.
   * Multiplicity: ProgrammeCoordinator 1..\* ⟶ Claim 1
4. **AcademicManager to Claim:**
   * An Academic Manager can review and approve many claims, but each claim is approved by only one manager.
   * Multiplicity: AcademicManager 1..\* ⟶ Claim 1
5. **HumanResources to Claim:**
   * HR can process multiple claims, but each claim is processed by only one HR staff member.
   * Multiplicity: HR 1..\* ⟶ Claim 1
6. **Claim to SupportingDocument:**
   * A claim can have multiple supporting documents, but each document is associated with only one claim.
   * Multiplicity: Claim 1..\* ⟶ SupportingDocument 1

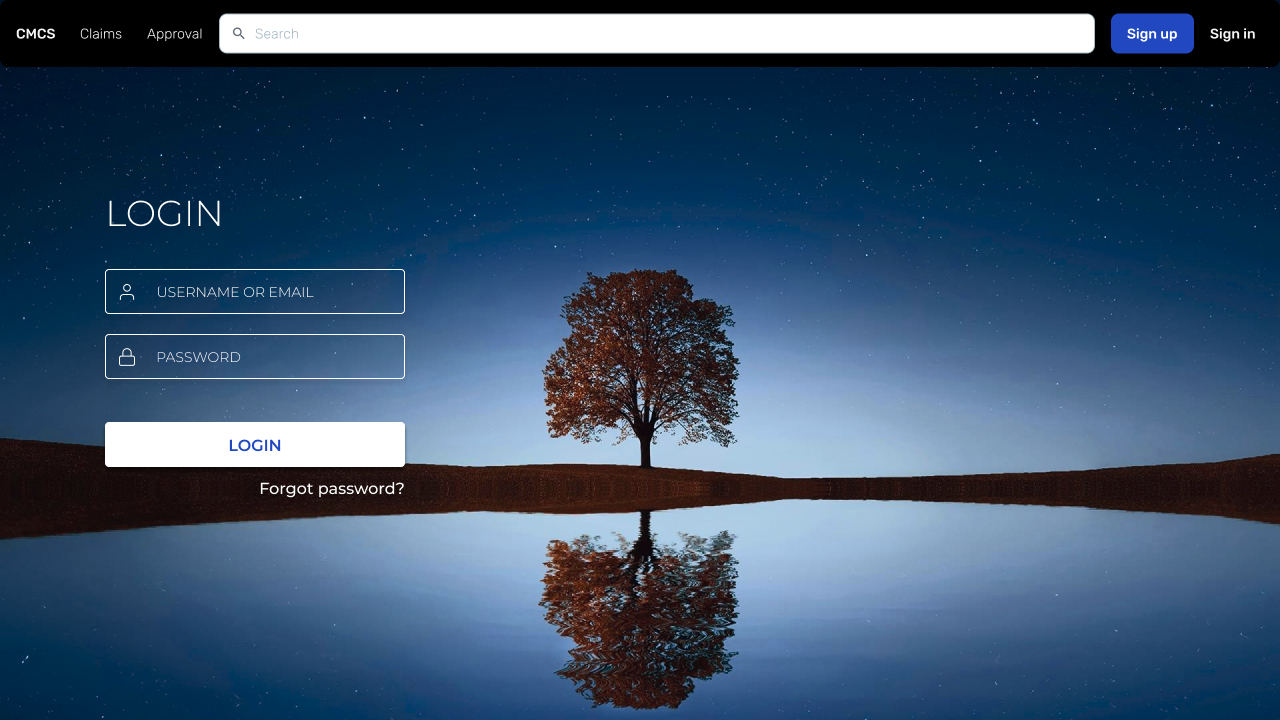
**How This Structure Supports CMCS Requirements:**

1. **Claim Submission and Tracking:**
   * Lecturers can submit claims through the Claim table, providing details such as hours worked and uploading supporting documents via the SupportingDocument table.
   * Each lecturer can track the status of their claims as they move through the approval process, with statuses such as "submitted," "approved," or "rejected."
2. **Multi-Level Claim Review and Approval:**
   * Claims move through several layers of approval. First, the ProgrammeCoordinator reviews and either approves or rejects the claim. Approved claims move to the AcademicManager for final approval. This ensures a thorough review process for each claim.
   * The relationships between claims and the different user roles (Coordinator, Manager, HR) ensure that claims are handled properly at each stage.
3. **Document Management:**
   * Supporting documents are linked to claims and can be accessed by Programme Coordinators, Academic Managers, and HR staff to ensure that claims are supported by the necessary documentation.
   * This helps to validate claims and streamline the review process by providing all necessary information in one place.
4. **Centralized and Scalable System:**
   * The database structure is designed to handle a large volume of users, claims, and supporting documents efficiently. By centralizing data into specific tables and establishing clear relationships between them, the system can scale as more users and claims are added over time.
   * The use of foreign keys ensures data integrity, linking claims to the correct lecturers and managing the flow of claims through the system.

**3. GUI Layout Explanation**(“Effective Project Documentation Best Practice,” n.d.)

* The **Graphical User Interface (GUI)** for the CMCS is designed to prioritize ease of use and functionality. The target users (lecturers, Programme Coordinators, and Academic Managers) vary in technical ability, so simplicity and clarity were paramount in the design choices. Figma (2016)

**Layout Features:**

* **Login and Register:** These pages are to login and register yourself in the application.
* **Lecturer Dashboard**: The dashboard for lecturers displays their submitted claims, with an option to filter by status (e.g., "Submitted," "Approved"). A prominent “Submit New Claim” button allows easy access to the claim submission form.
* **Claim Submission Form**: The form contains fields for hours worked, hourly rate, and a file upload option for supporting documents. Fields are clearly labeled, and validation is in place to ensure correct data entry.
* **Review Pages for Coordinators and Managers**: Both coordinators and managers have their own dashboards that display claims needing review or approval. The design allows them to easily access claim details and approve/reject claims with a few clicks.
* **Approval Workflow Integration**: The GUI includes clear visual indicators of claim progress, allowing users to track where their claim is in the approval process.
* 
* A tree reflected in water

  Description automatically generated
* A screenshot of a computer

  Description automatically generated
* A screenshot of a computer

  Description automatically generated
* A screenshot of a computer

  Description automatically generated
* Figma (2016)
* The GUI layout was designed to streamline the workflow, reduce the number of clicks needed to perform actions, and minimize confusion. The dashboard design gives users an immediate overview of relevant tasks (e.g., claims submitted or awaiting approval). The minimalistic design is intentional to reduce cognitive overload, especially for non-technical users like lecturers and administrators. This ensures a better user experience, fewer errors, and faster processing times. Figma (2016)

**4. Assumptions and Constraints**(“Effective Project Documentation Best Practice,” n.d.)

**Assumptions:**

1. **Lecturers Will Submit Claims Correctly**: It is assumed that lecturers will provide accurate data when submitting claims (e.g., correct hours worked and supporting documents). Input validation will ensure data integrity.
2. **Programme Coordinators and Academic Managers Will Follow the Approval Process**: It is assumed that all users responsible for reviewing claims will diligently follow the approval process, ensuring that no claim is left unreviewed.
3. **Supporting Documents Are in PDF Format**: It is assumed that all supporting documents, such as timesheets, will be uploaded as PDFs to maintain consistency and avoid compatibility issues.
4. **System Growth**: The system is designed to scale with the assumption that new lecturers, claims, and documents will be added regularly. It assumes an average increase in data over time without major spikes in usage.

**Constraints:**

1. **Database Performance**: The system’s performance could be affected by the number of claims and supporting documents stored in the database. To mitigate this, optimization techniques such as indexing and regular maintenance will be necessary.
2. **File Size Limits**: The system may impose size limits on uploaded files (e.g., supporting documents) to prevent performance degradation or excessive storage costs.
3. **Security Constraints**: Due to the sensitive nature of the data (financial and personal information), security measures such as encrypted communication and secure password storage must be in place. This adds complexity to the system, but it is a necessary tradeoff for ensuring data integrity.
4. **User Access Levels**: The system assumes distinct roles for users (Lecturers, Programme Coordinators, and Academic Managers), which requires well-defined access control mechanisms to prevent unauthorized actions.

**5.Conclusion**

Every design choice made for the CMCS was driven by the need to create a system that is secure, scalable, and easy to use. By implementing a structured database, an intuitive GUI, and adhering to assumptions and constraints, the system will support the submission and approval of claims in a reliable and efficient manner. Each decision is backed by industry best practices, ensuring that the system is not only functional but also adaptable for future enhancements.

**Reference List**

(“Database - Overview, Roles and Components, DBMS,” n.d.)

Database - Overview, Roles and Components, DBMS [WWW Document], n.d. URL **https://corporatefinanceinstitute.com/resources/data-science/database/** (accessed 3.14.24).

(“Effective Project Documentation Best Practice,” n.d.)

Effective Project Documentation Best Practice [WWW Document], n.d. URL **https://www.orangescrum.com/tutorial/project-documentation-and-reporting/effective-project-documentation** (accessed 9.10.24).

Figma (2016) ‘Figma for UX design’. **Figma.com**