Deliverable 2

PES1UG22CS510

PES1UG22CS516

PES1UG22CS522

1. Lifecycle Model Selection

Model: Waterfall Model

- Reasoning: The Waterfall model is suitable for a structured project like a banking system, where the requirements are clear from the start, and there's a linear progression of tasks. This model ensures that the stages such as requirement gathering, system design, implementation, testing, and maintenance happen sequentially, which suits the complexity and security needs of a banking system.

2. Tools to Use

Planning Tool:

- **Jira**: For task management and tracking.
- **Microsoft Project**: For Gantt charts and scheduling.

Design Tool:

- **Lucidchart or Microsoft Visio**: For ER diagrams and other design artifacts (like database design, and API flow).

Version Control:

- **Git (GitHub or GitLab)**: For managing code versions, especially with React and MySQL scripts.

Development Tools:

- **MySQL Workbench**: For database development and query writing.
- **VS Code**: For coding with extensions for React, JavaScript, and MySQL.
- **React**: For the frontend development of the banking system, offering a responsive and dynamic user interface. React will handle user interaction, dashboards, and user experience.
- **Node.js**: For building an API to connect the React frontend with the MySQL database.

Bug Tracking:

- **Jira**: For managing bugs and issues.

Testing Tools:

- **Selenium**: For automated end-to-end testing of the front-end.
- **Jest**: For unit testing React components.
- **Manual Testing**: Through MySQL Workbench for database testing.

3. Deliverables & Components

- **Reuse Components**:
    - **MySQL Pre-built Functions**: Instead of building functions like SUM(), COUNT(), etc., use pre-built MySQL functions to enhance efficiency.
    - **Libraries**: If using PHP or any other language for the front-end, reusable libraries like mysqli or PDO for database interaction.
- **Build Components**:

    - **Database Schema**: Building the customized tables for accounts, transactions, customers, etc.
    - **Stored Procedures**: Writing custom procedures for banking operations like deposit, withdrawal, balance check, etc.
    - **Triggers**: Writing triggers for auditing and compliance.
- **Justification**:
    - Reusing standard functions and libraries will reduce development time and ensure fewer errors, while building custom components is necessary to handle the specific functionalities required in the banking system.

4. Work Breakdown Structure (WBS) for Banking Management System:

1. Requirements

- **Functional Requirements**
    - Account creation and management
    - Money transfer functionality
    - Loan and interest management
    - Customer support services
- **Non-Functional Requirements**
    - Security and encryption standards (e.g., PCI-DSS)
    - Performance requirements (scalability, reliability)
- **Requirements Documentation**
    - Functional Specifications Document (FSD)
    - System Requirement Specifications (SRS)

2. Design

- **System Design**
    - High-level architecture
    - Design of microservices (if applicable)
- **System Architecture**
    - Database design (MySQL schemas for user accounts, transactions, etc.)
    - API design (for interacting between frontend and backend)
- **UI/UX Design**
    - User flow diagrams
    - Wireframes and prototyping
- **User Interface (UI)**
    - Customer-facing dashboard
    - Admin/Banker-facing dashboard

3. Development

- **Frontend Development (React)**
    - **Account Management Pages**
        - Create accounts, view balances, transaction history
    - **Loan Application Pages**
        - Apply for loans, check loan status
    - **Transaction Management**
        - Transfer money, schedule payments
- **Backend Development (MySQL, Node.js)**
    - **User Authentication and Role Management**

        - Customer, banker, and admin roles
    - **Transaction Processing**
        - Handle money transfers, deposits, withdrawals
    - **Loan and Interest Calculations**
        - Automatic interest calculation and management
    - **Admin Dashboard**
        - View system reports, manage users, oversee transactions

4. Testing

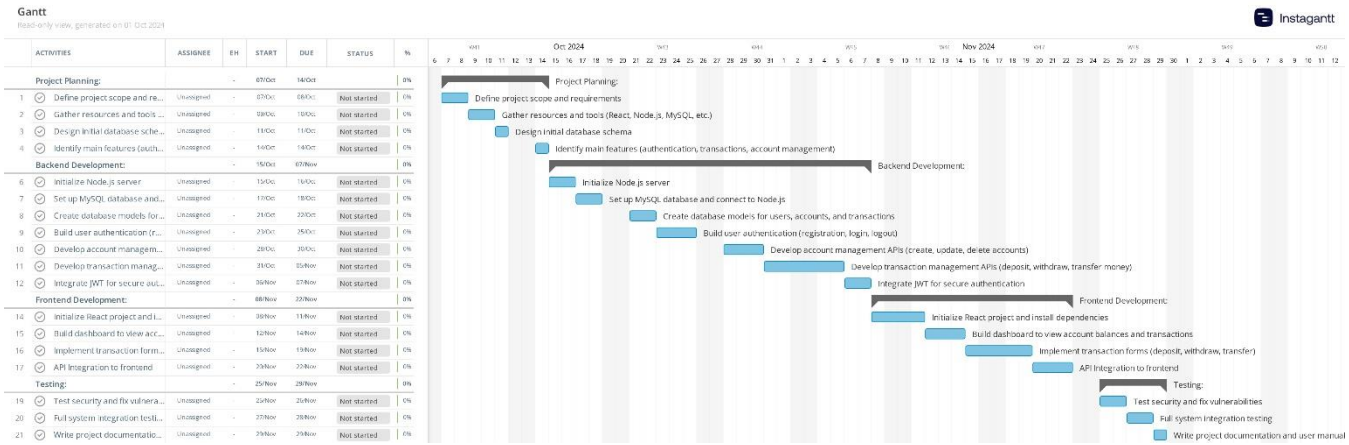
- **Unit Testing**
    - Test frontend components (React) and backend APIs (Node.js)
- **Integration Testing**
    - Test database (MySQL) interactions with frontend and backend services
- **User Acceptance Testing (UAT)**
    - Ensure system meets the needs of bank staff and customers

5. Deployment

- **Setup Hosting Environment**
    - Cloud hosting (AWS/GCP/Azure) – future work
    - Database server setup (MySQL)
- **Deployment of Backend and Frontend**
    - Deploy React frontend, Node.js backend
    - Setup database on the cloud

- **Post Deployment Checks**
    - Security audits
    - Load testing and performance optimization

5. Gantt Chart



6. Coding Details

- **Code Structure:**
    - Backend will follow an MVC (Model-View-Controller) architecture using Node JS.
    - The frontend will use HTML, CSS, JavaScript, and integrate responsive design with React.
    - Database: MySQL with tables for users, transaction, account details.
- **Testing Strategy:**
    - Unit testing of the backend logic.
    - Integration tests to validate the interaction between the frontend, backend, and the database.

- Use Selenium for automated UI testing to ensure all the processes work smoothly.