SOFTWARE ENGINEERING CONCEPTS - LAB MANUAL

220701299 - TAMILINI D K
220701306 - UDHAYA SHANKAR J
220701312 - VEDHA VIGHESHWAR
220701318 - VIGNESH S
220701324 - VISHAL P
220701330 - YUVEN SENTHIL KUMAR

Software Concepts & Engineering - Lab Manual

Software Concepts & Engineering - Lab Manual

Overview of the Project

Problem Statement:

Despite the advancements in software testing methodologies, many organizations struggle with:

- efficiently managing and
- resolving the vast array of issues that can arise during testing phases.

The challenges include:

- prioritizing critical issues,
- assigning the right resources, and
- ensuring timely resolution without compromising software quality.

The need for a robust, scalable, and systematic testing management system is evident, one that can adapt to the dynamic nature of software development and ensure that all problems, regardless of their priority, are addressed effectively.

Data Perspective:

From a data standpoint, the project involves collecting, organizing, and analyzing data related to inventory. This includes the detailed representation of the data entities, their attributes, and relationships. Below is an outline of the data perspective, including the entities and their attributes, relationships, and database schema.

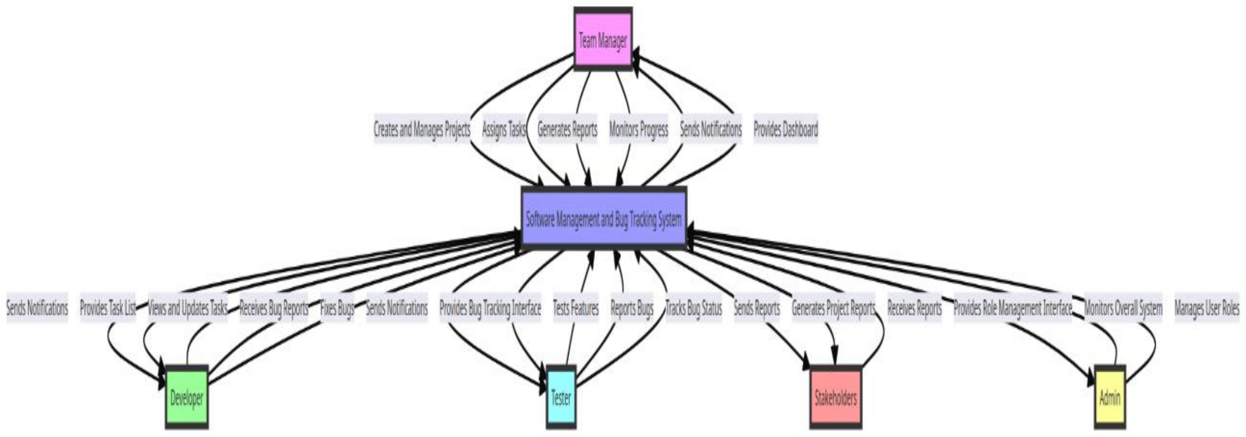
Benefits of implementing:

1. **Centralized Project Management**:
   - Simplifies project tracking and management, reducing administrative overhead and enabling better focus on strategic tasks.
2. **Task Assignment and Monitoring**:
   - Ensures optimal utilization of resources, improves productivity, and helps in balancing the team's workload effectively.
3. **Comprehensive Reporting**:
   - Enhances transparency, keeps stakeholders informed, and builds trust by providing regular updates on project progress and performance.
4. **Clear Task Management**:
   - Reduces confusion, ensures developers focus on the most critical tasks first, and improves overall workflow efficiency.
5. **Efficient Bug Tracking and Resolution**:
   - Accelerates bug resolution process, minimizes time spent on clarifying issues, and improves the quality of code.
6. **Structured Bug Reporting**:
   - Ensures that bugs are reported consistently and accurately, making it easier for developers to understand and address issues.

7. **Efficient Bug Tracking**:
    - Keeps testers informed about the progress of bug fixes, allowing them to re-test and close issues promptly.

8. **Enhanced Efficiency and Productivity**:
    - Reduces manual effort, speeds up task completion, and improves the overall productivity of the development team.

Business Architecture Diagram



1. **Business Need:** The primary business need for developing a software management and bug tracking system arises from the need to efficiently manage software development projects, ensure high-quality deliverables, and improve collaboration among team members. This system serves multiple stakeholders including developers, testers, managers, and administrators by streamlining project management, tracking bugs, and generating comprehensive reports.
2. **Current Process:**
    - **Manual Process:** Currently, software management and bug tracking system might involve manual methods such as spreadsheet tracking, asking each team member about their task, or basic software systems. This process is labor-intensive, error-prone, and lacks real-time visibility into software development phase and bug tracking.
3. **Different Personas and Their Use of the System:**
    - **Team Managers:** The team manager uses the system to oversee the entire project lifecycle from inception to completion. They create new software projects, assign tasks to developers and testers, and track the progress of these tasks through detailed dashboards and reports. The manager utilizes performance data to allocate tasks effectively, ensuring that developers and testers are assigned work that matches their expertise and previous performance. They also generate and share comprehensive project reports with stakeholders, highlighting key milestones, progress updates, and any potential issues. This allows the manager to maintain a high-level view of all ongoing projects, ensuring that deadlines are met and quality standards are maintained.

- **Developers:** Developers use the system to receive and manage their assigned tasks, which include developing new features and fixing reported bugs. Upon logging into the system, they can view their task list, which is prioritized based on urgency and their previous performance. For each task, developers can access detailed requirements and documentation, communicate with testers and managers through integrated messaging, and update the task status as they progress. The system also allows developers to log time spent on each task, providing insights into their productivity and helping managers make data-driven decisions for future task assignments. By using this system, developers can efficiently manage their workload, collaborate effectively with testers, and ensure that their work aligns with project goals and timelines.
- **Tester:** Testers use the system to track the quality of software by thoroughly testing new features and identifying bugs. They receive tasks related to specific features or modules, conduct tests, and log any bugs they find in the system. Each bug report includes detailed information such as the severity, steps to reproduce, and any relevant screenshots or logs. The system automatically notifies the assigned developer of the new bug, allowing for prompt resolution. Testers can also track the status of bugs they have reported, ensuring that they are fixed in a timely manner. By utilizing this system, testers can maintain a structured and efficient testing process, contribute to the overall quality of the software, and ensure that issues are communicated clearly and resolved quickly.

Possible Business Problems Addressed by the Software Management and Bug Tracking System

1. **Inefficient Project Management**

Problem: Difficulty in tracking project progress, assigning tasks, and ensuring timely completion.
Solution: The system provides a centralized platform for creating and managing projects, assigning tasks, and monitoring progress through dashboards and reports.

2. **Poor Collaboration and Communication**

Problem: Lack of effective communication and collaboration among team members leads to misunderstandings and delays.

Solution: Integrated messaging and notification features facilitate clear communication between developers, testers, and managers, ensuring everyone is aligned.

3. **Unclear Role Assignments**

Problem: Ambiguity in role assignments can lead to overlapping responsibilities or neglected tasks.

Solution: The system clearly defines roles (developer, tester, manager) and assigns tasks based on these roles and individual performance metrics.

4. **Delayed Bug Resolution**

Problem: Bugs are often reported late or not addressed promptly, affecting software quality and release schedules.

Solution: A structured bug reporting and tracking system ensures that bugs are logged, prioritized, assigned, and resolved efficiently, minimizing delays**.**

5. **Lack of Performance Insights**

Problem: Managers lack visibility into individual and team performance, making it difficult to allocate resources effectively.

Solution: Detailed performance tracking and reporting enable managers to make data-driven decisions, improving resource allocation and project outcomes.

6. **Quality Assurance Challenges**

Problem: Ensuring consistent quality across software releases is challenging without a robust testing and bug tracking process**.**
Solution: Testers can systematically report bugs, and the status of these bugs can be tracked until resolution, ensuring that quality issues are addressed before release.

7. **Inadequate Reporting for Stakeholders**

Problem: Difficulty in generating comprehensive reports for stakeholders can lead to a lack of transparency and informed decision-making.

Solution: The system enables managers to generate detailed reports on project status, progress, and performance, providing stakeholders with the necessary insights.

8. **Resource Mismanagement**

Problem: Inefficient use of human and technical resources can lead to project delays and increased costs.

Solution: The system optimizes resource utilization by tracking task assignments and performance, ensuring that resources are used effectively and efficiently**.**

9. Scalability Issues

Problem: As projects grow in size and complexity, managing them becomes increasingly difficult without the right tools.

Solution: The system scales with the organization, allowing for multiple projects to be managed concurrently with support for complex workflows and large teams.

10. Inconsistent Documentation and Knowledge Sharing

Problem: Lack of proper documentation and knowledge sharing can lead to repeated mistakes and inefficiencies**.**

Solution: The system provides a centralized repository for project documentation, bug reports, and feature descriptions, ensuring that knowledge is preserved and accessible.

User Stories for a Software Testing and Management System

1. **As a project manager, I want to create new test projects so that I can manage multiple testing efforts simultaneously.**
   - **Estimate**: 5 story points
2. **As a tester, I want to create and assign test cases to specific test projects so that I can organize tests according to project requirements.**
   - **Estimate**: 8 story points
3. **As a tester, I want to execute test cases and record the results so that I can track the status of each test.**
   - **Estimate**: 8 story points
4. **As a project manager, I want to generate detailed test reports for each project so that I can review testing progress and results.**
   - **Estimate**: 8 story points
5. **As a tester, I want to update and modify test cases so that I can ensure they reflect the most current requirements and conditions.**
   - **Estimate**: 5 story points
6. **As a developer, I want to receive notifications when a test case fails so that I can address issues promptly.**
   - **Estimate**: 3 story points
7. **As a tester, I want to link test cases to specific requirements so that I can ensure full coverage of all requirements.**
   - **Estimate**: 8 story points
8. **As a project manager, I want to assign testers to specific test cases so that I can distribute workload effectively.**
   - **Estimate**: 5 story points
9. **As a project manager, I want to track the time spent on each test case so that I can monitor productivity and efficiency.**
   - **Estimate**: 8 story points
10. **As a tester, I want to mark test cases as automated or manual so that I can differentiate between them easily.**

   - **Estimate**: 3 story points
11. **As a project manager, I want to view a dashboard with the overall status of all testing projects so that I can get a quick overview of testing progress.**
   - **Estimate**: 8 story points

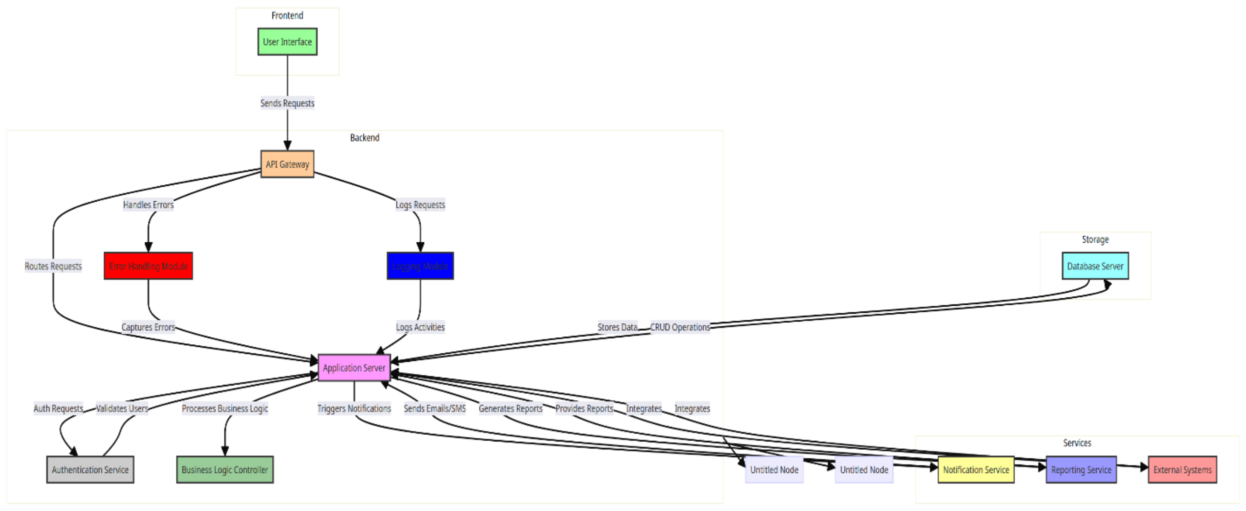
Non-Functional Requirements (NFRs)

1. **Performance**: The system should handle up to 1000 concurrent users without significant performance degradation, ensuring that the response time for any action is less than 2 seconds under normal load conditions.
2. **Security**: The system must ensure secure data transmission by implementing SSL/TLS encryption and should support role-based access control (RBAC) to restrict access to sensitive functionalities and data.
3. **Reliability**: The system should have an uptime of 99.9% and should be able to recover from failures within 5 minutes to ensure minimal disruption to testing activities.

Poker Planning

Sure, let's do a poker planning session for these user stories and non-functional requirements. We'll use the Fibonacci sequence for estimation, with 1 being the smallest and 13 being the largest.

Architecture Diagram



Architecture Pattern Used: Model-View-Controller (MVC)

Reasons for Using MVC Architecture:

1. **Separation of Concerns**:
   - **Modularization**: MVC separates the application into three interconnected components—Model, View, and Controller—each responsible for different aspects of the application, leading to a more organized codebase.
   - **Maintenance**: Makes the application easier to maintain and update because changes in one component have minimal impact on others.
2. **Parallel Development**:
   - **Team Collaboration**: Different developers can work on the Model, View, and Controller simultaneously, which speeds up the development process.
   - **Specialization**: Developers can specialize in one part of the application, improving productivity and code quality.
3. **Reusability**:
   - **Code Reuse**: Models and Views can be reused across different parts of the application or even in different projects.
   - **Component Reuse**: Views can be shared and reused, reducing duplication and inconsistencies.
4. **Flexibility**:
   - **Adaptability**: Changes in the business logic (Model) do not require changes in the user interface (View) and vice versa.
   - **Testing**: Components can be tested independently, making it easier to identify and fix bugs.
5. **Scalability**:
   - **Component Scaling**: MVC architecture allows individual components to be scaled independently based on their needs, improving performance and resource utilization.

Software Concepts & Engineering - Lab Manual

Components in MVC Architecture for Software Management and Bug Tracking System

1. **Model**:
   - **Definition**: Represents the data and the business logic of the application.
   - **Components**:
     - **Project**: Handles data related to software projects.
     - **Task**: Manages task data.
     - **Bug**: Tracks bugs and their status.
     - **User**: Manages user information and roles.
     - **Report**: Generates data for reporting purposes.
2. **View**:
   - **Definition**: Responsible for presenting data to the user.
   - **Components**:
     - **Project Dashboard**: Displays project status and progress.
     - **Task List**: Shows tasks assigned to developers and testers.
     - **Bug Tracker**: Lists bugs reported and their status.
     - **Reports**: Visual representation of project reports for stakeholders.
3. **Controller**:
   - **Definition**: Acts as an intermediary between Model and View, handling user input and updating the Model.
   - **Components**:
     - **Project Controller**: Manages project-related actions (create, update, delete).
     - **Task Controller**: Handles task assignments and status updates.
     - **Bug Controller**: Manages bug reporting and resolution.
     - **User Controller**: Oversees user authentication and role management.
     - **Report Controller**: Controls report generation and distribution.

Design Principles Used and Their Importance

1. **Single Responsibility Principle (SRP)**

   Definition: A class should have only one reason to change, meaning it should have only one job or responsibility.

   Usage in the Project:

   - Each class in the system, such as `Project`, `Task`, `Bug`, `User`, and `Report`, has a distinct responsibility. For example, the `Bug` class is responsible solely for managing bug-related data and operations.
   - **Benefits**:
     - **Maintainability**: Makes the system easier to understand and maintain, as each class focuses on a specific aspect of the application.
     - **Scalability**: Allows individual components to be modified or extended without affecting other parts of the system.
2. **Open/Closed Principle (OCP)**

Definition: Software entities (classes, modules, functions, etc.) should be open for extension but closed for modification.

Usage in the Project:

- The design allows for adding new features (e.g., additional notification methods or new types of reports) without altering existing code. For instance, the `NotificationService` can be extended to include new notification channels.
- **Benefits**:
  - **Extensibility**: Facilitates adding new functionalities with minimal impact on existing code.
  - **Reliability**: Reduces the risk of introducing bugs when extending the system.

3. **Liskov Substitution Principle (LSP)**

   Definition: Objects of a superclass should be replaceable with objects of a subclass without affecting the correctness of the program.

   Usage in the Project:

   - If the system has a base class `User` and derived classes like `Manager`, `Developer`, and `Tester`, these subclasses can be used interchangeably wherever a `User` object is expected.
   - **Benefits**:
     - **Interchangeability**: Enhances the flexibility of the system, allowing for the use of more specific types while maintaining general functionality.
     - **Robustness**: Ensures that substituting subclasses does not compromise system behavior.

4. **Interface Segregation Principle (ISP)**

   Definition: Clients should not be forced to depend on interfaces they do not use.

   Usage in the Project:

   - Different interfaces are designed for specific roles. For example, `IDeveloper` might include methods specific to developers, while `ITester` includes methods specific to testers.
   - **Benefits**:
     - **Cohesion**: Increases the cohesion of interfaces, making them more specific to the clients' needs.
     - **Reduced Complexity**: Simplifies the implementation of interfaces and the classes that use them.

5. **Dependency Inversion Principle (DIP)**

   Definition: High-level modules should not depend on low-level modules. Both should depend on abstractions. Abstractions should not depend on details. Details should depend on abstractions.

Usage in the Project:

- High-level modules like `ProjectController` depend on abstract services like `INotificationService` rather than concrete implementations.
- **Benefits**:
  - **Decoupling**: Reduces the coupling between high-level and low-level modules, promoting more flexible and maintainable code.
  - **Testability**: Improves the testability of the system by allowing the use of mock implementations for testing purposes.

6. **Don't Repeat Yourself (DRY)**

   Definition: Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.

   Usage in the Project:

   - Common functionality is abstracted into reusable methods or services, such as `AuthenticationService` or `NotificationService`, to avoid duplication.
   - **Benefits**:
     - **Maintainability**: Reduces redundancy, making the system easier to maintain and update.
     - **Consistency**: Ensures consistency across the application by centralizing common functionality.

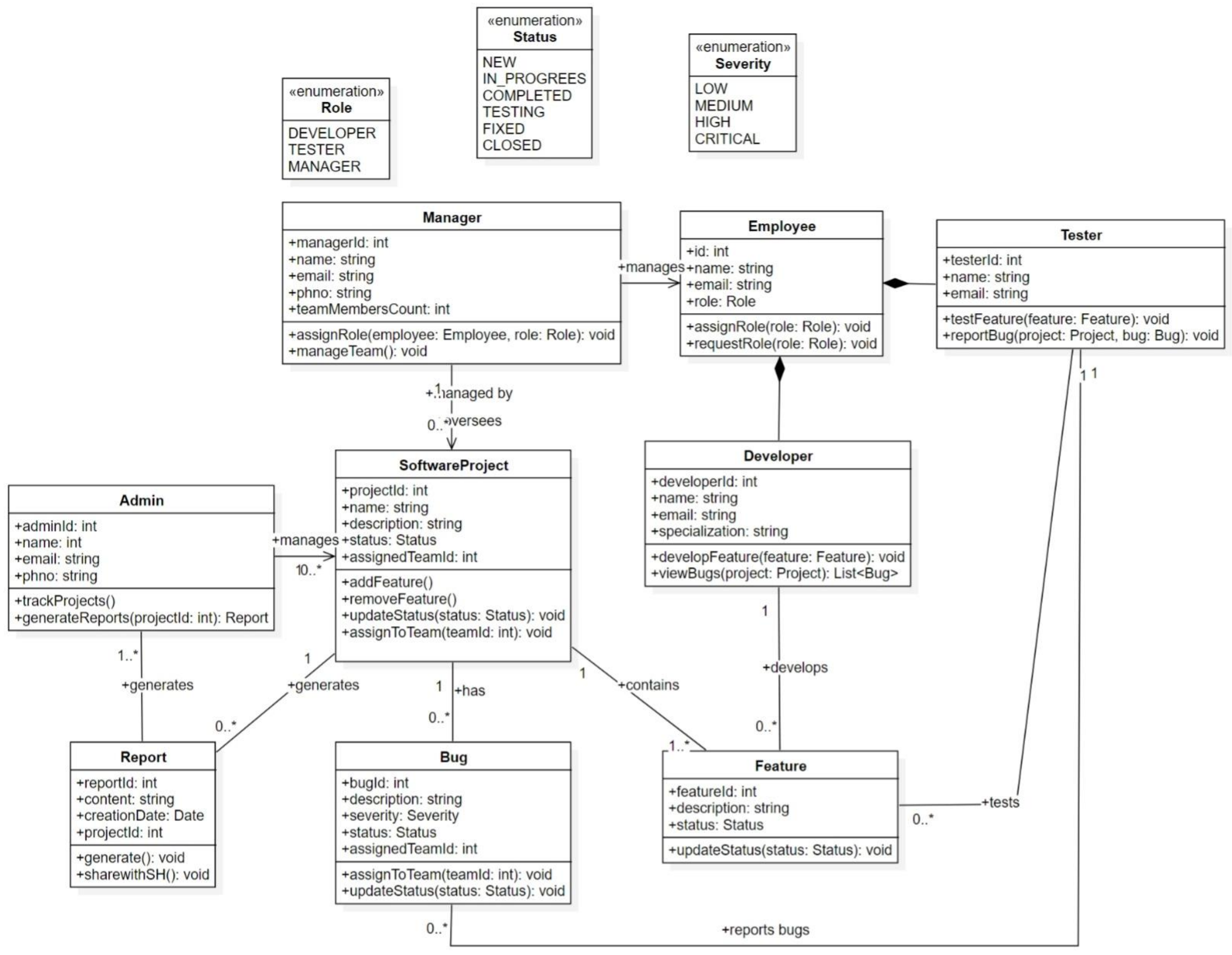
7. **Separation of Concerns (SoC)**

   Definition: Different parts of a program should have distinct and separate responsibilities.

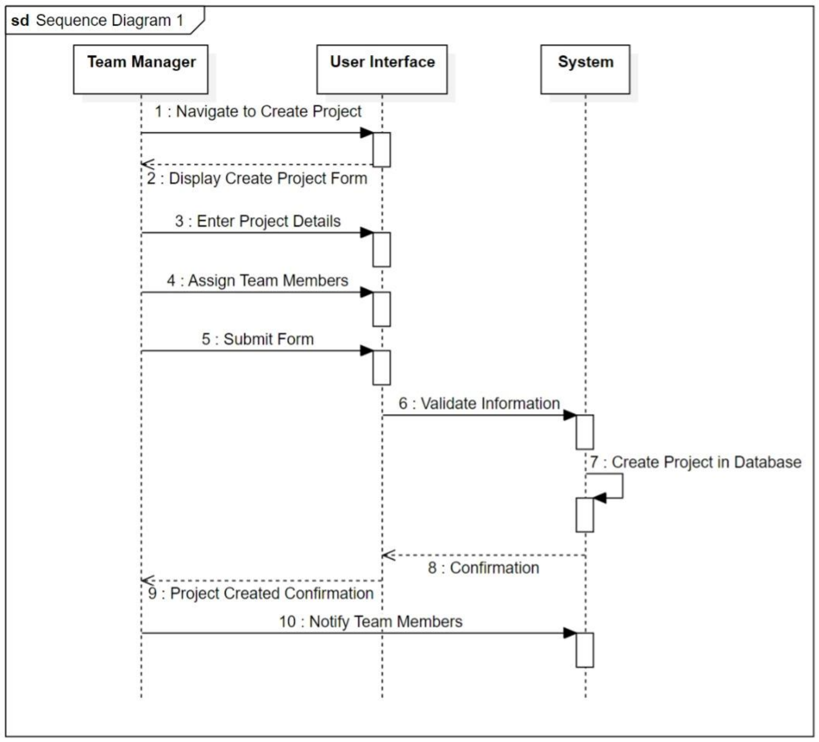
   Usage in the Project:

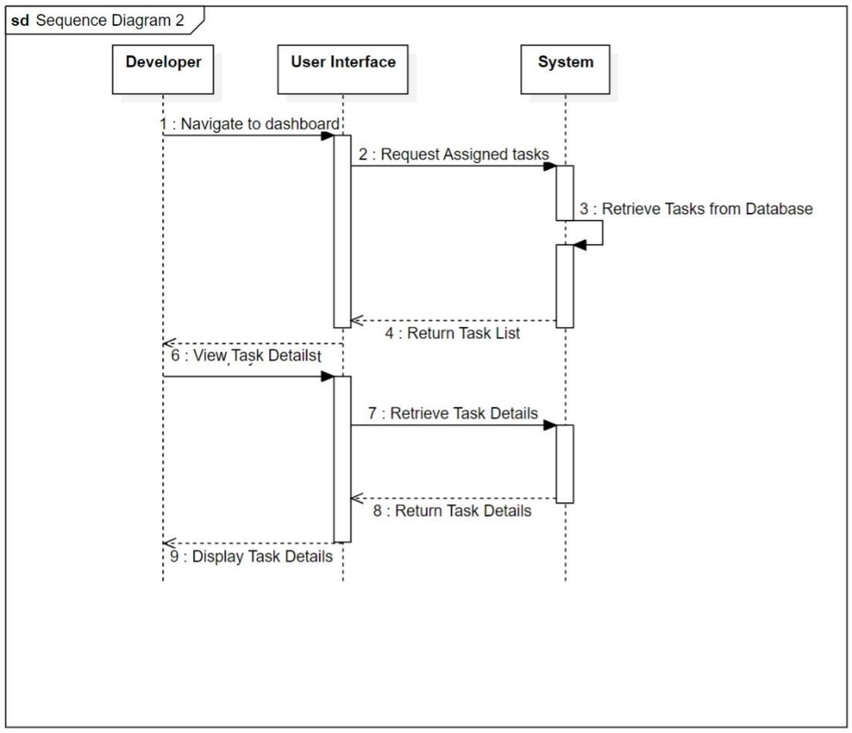
   - The MVC architecture itself embodies SoC by separating the user interface (View), business logic (Controller), and data management (Model).
   - **Benefits**:
     - **Clarity**: Enhances the clarity of the system by dividing it into distinct sections, each responsible for a specific aspect of the functionality.
     - **Modularity**: Promotes modularity, allowing for easier maintenance and extension.

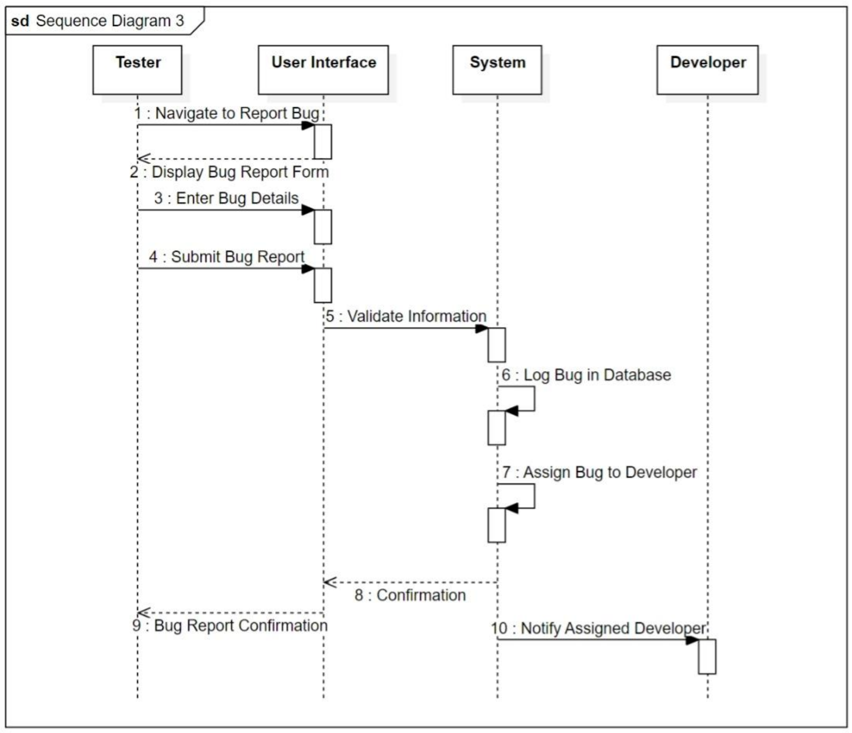
CLASS DIAGRAM

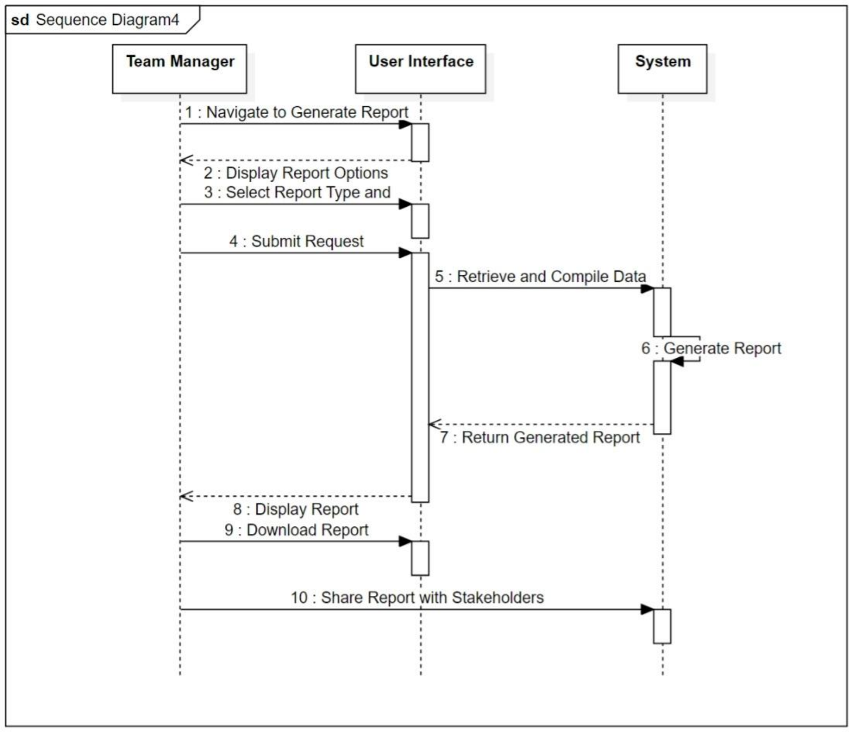


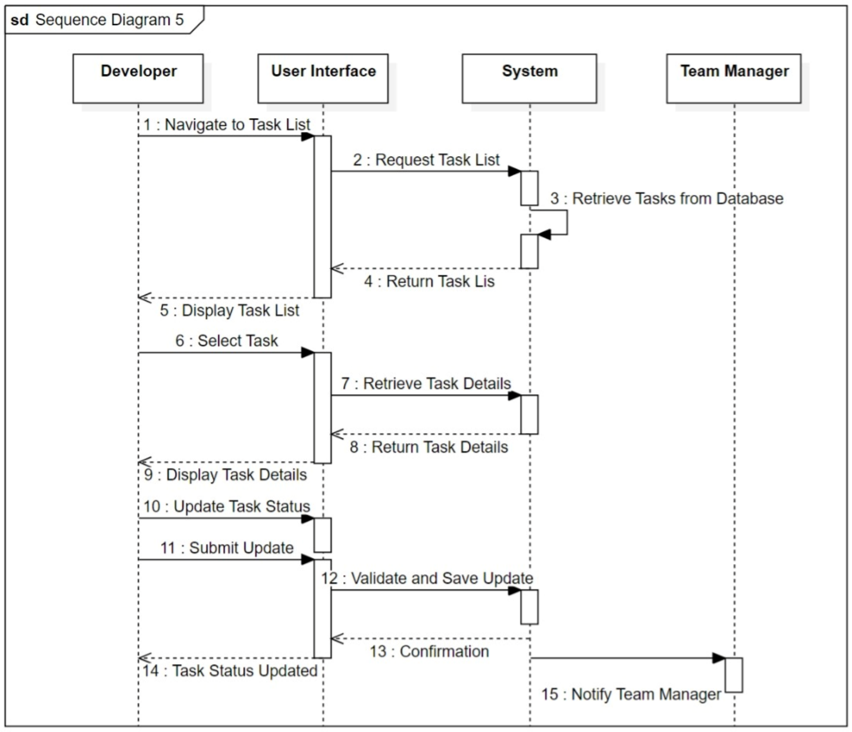
SEQUENCE DIAGRAMS






sd Sequence Diagram 3
Tester
User Interface
System
Developer
1 : Navigate to Report Bug
2 : Display Bug Report Form
3 : Enter Bug Details
4 : Submit Bug Report
5 : Validate Information
6 : Log Bug in Database
7 : Assign Bug to Developer
8 : Confirmation
9 : Bug Report Confirmation
10 : Notify Assigned Developer


sd Sequence Diagram4
Team Manager
User Interface
System
1 : Navigate to Generate Report
2 : Display Report Options
3 : Select Report Type and
4 : Submit Request
5 : Retrieve and Compile Data
6 : Generate Report
7 : Return Generated Report
8 : Display Report
9 : Download Report
10 : Share Report with Stakeholders


sd Sequence Diagram 5
Developer
User Interface
System
Team Manager
1 : Navigate to Task List
2 : Request Task List
3 : Retrieve Tasks from Database
4 : Return Task Lis
5 : Display Task List
6 : Select Task
7 : Retrieve Task Details
8 : Return Task Details
9 : Display Task Details
10 : Update Task Status
11 : Submit Update
12 : Validate and Save Update
13 : Confirmation
14 : Task Status Updated
15 : Notify Team Manager

Test Strategy

Test Plans

Objective: To ensure the software testing and management system functions as expected, is user-friendly, secure, and scalable.

Scope:

- Functional Testing
- Non-Functional Testing (Performance, Security, Usability)
- Integration Testing
- System Testing
- Regression Testing

Test Environments:

- Development Environment
- Staging Environment
- Production Environment

Tools:

- **Test Management**: Jira
- **Automation**: Selenium, JUnit
- **Performance Testing**: JMeter
- **Security Testing**: OWASP ZAP
- **CI/CD**: Azure DevOps
- **Version Control**: GitHub

Test Cases

User Story 1: Create a New Test Project

Happy Path:

- **Test Case ID**: TC001
- **Description**: Verify that a new test project can be created successfully.
- **Preconditions**: User is logged in.
- **Steps**:
    1. Navigate to the "Create Project" page.
    2. Enter valid project details.

3. Click "Submit".

- **Expected Result**: Project is created and visible in the project list.

Error Scenario:

- **Test Case ID**: TC002
- **Description**: Verify that an error message is displayed when creating a project with missing required fields.
- **Preconditions**: User is logged in.
- **Steps**:
    1. Navigate to the "Create Project" page.
    2. Leave required fields empty.
    3. Click "Submit".
- **Expected Result**: Error message "Required fields cannot be empty" is displayed.

User Story 2: Create and Assign Test Cases

Happy Path:

- **Test Case ID**: TC003
- **Description**: Verify that a test case can be created and assigned to a project.
- **Preconditions**: User is logged in, and a project exists.
- **Steps**:
    1. Navigate to the "Create Test Case" page.
    2. Enter valid test case details.
    3. Select a project to assign.
    4. Click "Submit".
- **Expected Result**: Test case is created and assigned to the selected project.

Error Scenario:

- **Test Case ID**: TC004
- **Description**: Verify that an error message is displayed when creating a test case with invalid data.
- **Preconditions**: User is logged in, and a project exists.
- **Steps**:
    1. Navigate to the "Create Test Case" page.

2. Enter invalid data (e.g., extremely long name).
3. Click "Submit".

- **Expected Result**: Error message "Invalid data entered" is displayed. **User**

Story 3: Execute Test Cases and Record Results

Happy Path:

- **Test Case ID**: TC005
- **Description**: Verify that a test case can be executed and the result recorded.
- **Preconditions**: User is logged in, and a test case exists.
- **Steps**:
    1. Navigate to the test case.
    2. Click "Execute".
    3. Enter results.
    4. Click "Submit".
- **Expected Result**: Test case result is recorded and updated.

Error Scenario:

- **Test Case ID**: TC006
- **Description**: Verify that an error message is displayed when submitting results with missing fields.
- **Preconditions**: User is logged in, and a test case exists.
- **Steps**:
    1. Navigate to the test case.
    2. Click "Execute".
    3. Leave result fields empty.
    4. Click "Submit".
- **Expected Result**: Error message "Results cannot be empty" is displayed.

User Story 4: Generate Detailed Test Reports

Happy Path:

- **Test Case ID**: TC007
- **Description**: Verify that a detailed test report can be generated.
- **Preconditions**: User is logged in, and test cases have been executed.
- **Steps**:

1. Navigate to the "Reports" page.
2. Select a project.
3. Click "Generate Report".

- **Expected Result**: Detailed report is generated and displayed.

Error Scenario:

- **Test Case ID**: TC008
- **Description**: Verify that an error message is displayed when generating a report for a project with no test cases.
- **Preconditions**: User is logged in.
- **Steps**:
  1. Navigate to the "Reports" page.
  2. Select a project with no test cases.
  3. Click "Generate Report".
- **Expected Result**: Error message "No test cases found for this project" is displayed.

User Story 5: Receive Notifications on Test Case Failures

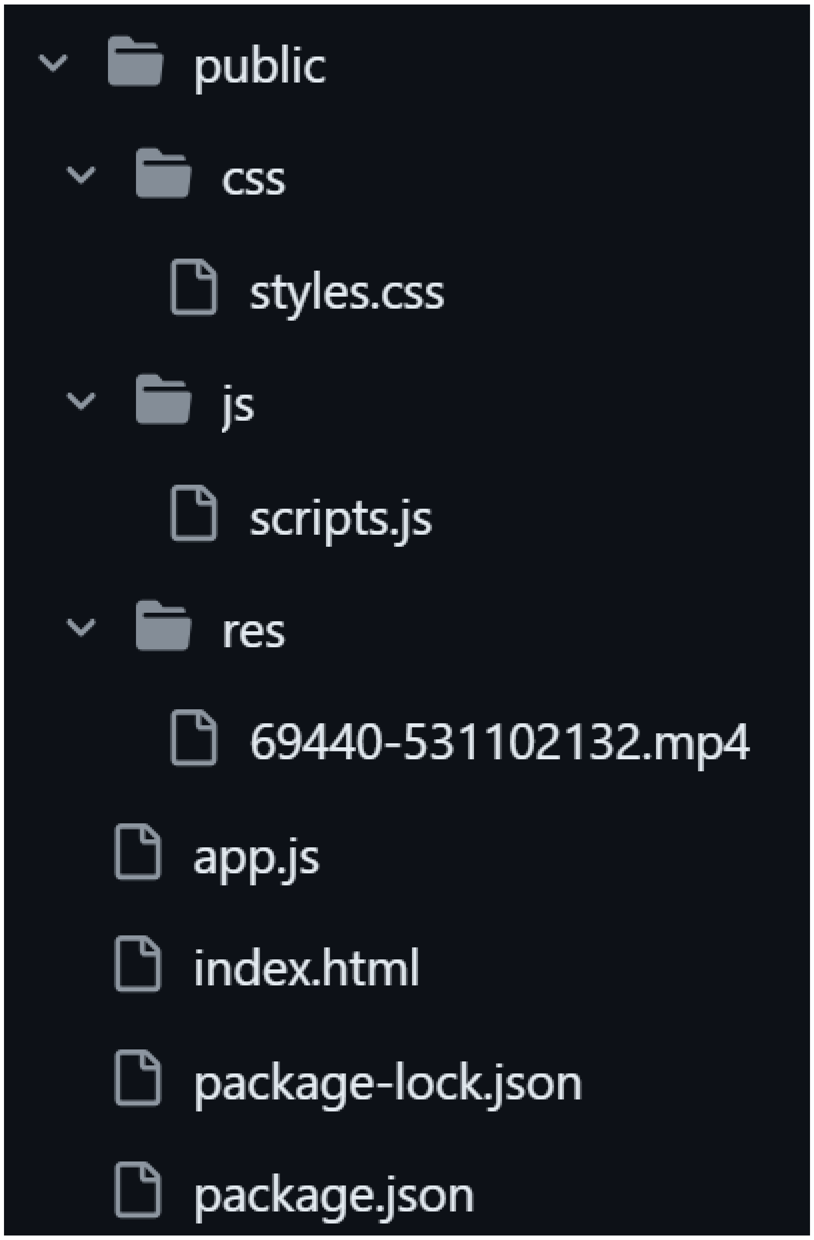
Happy Path:

- **Test Case ID**: TC009
- **Description**: Verify that a notification is sent when a test case fails.
- **Preconditions**: User is logged in, and a test case exists.
- **Steps**:
  1. Execute a test case and mark it as failed.
- **Expected Result**: Notification is sent to the assigned users.

Error Scenario:

- **Test Case ID**: TC010
- **Description**: Verify that no notification is sent when a test case passes.
- **Preconditions**: User is logged in, and a test case exists.
- **Steps**:
  1. Execute a test case and mark it as passed. ● **Expected Result**: No notification is sent.

GitHub Repository Structure



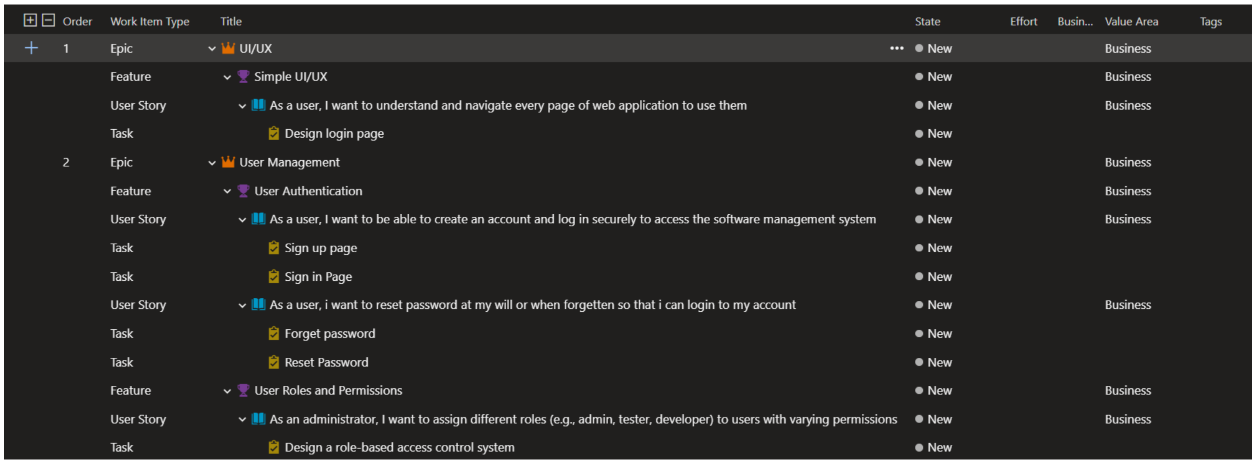
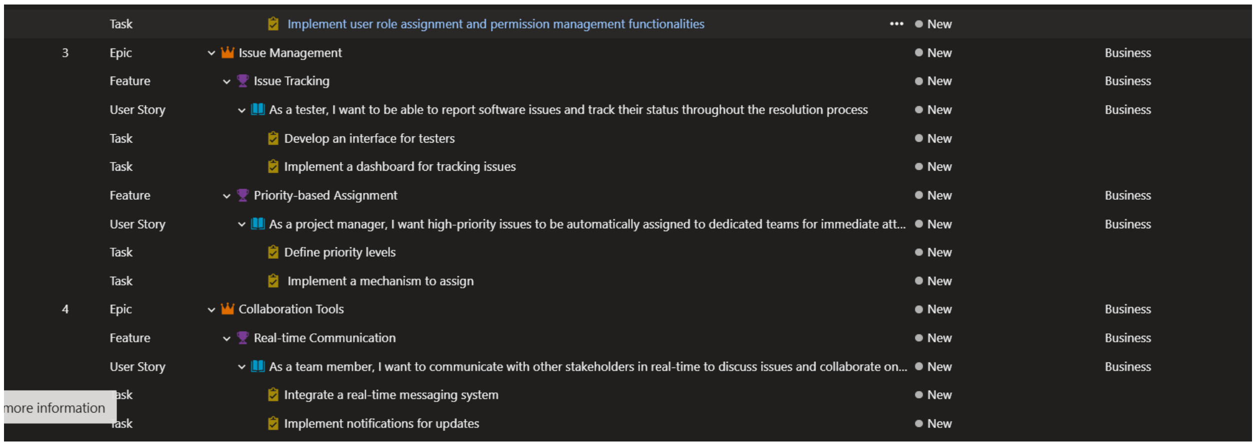
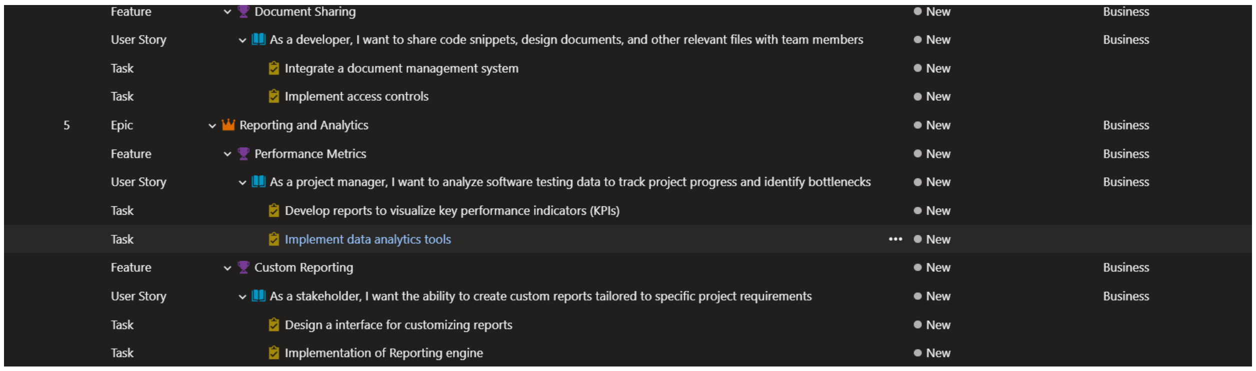
```
public
  css
    styles.css
  js
    scripts.js
  res
    69440-531102132.mp4
  app.js
  index.html
  package-lock.json
  package.json
```

Naming Conventions:

- **Packages**: Use lowerCamelCase.
- **Classes**: Use UpperCamelCase.
- **Methods**: Use lowerCamelCase.
- **Configuration Files**: Use descriptive names.

Software Concepts & Engineering - Lab Manual

A view of their DevOps Architecture for their respective project

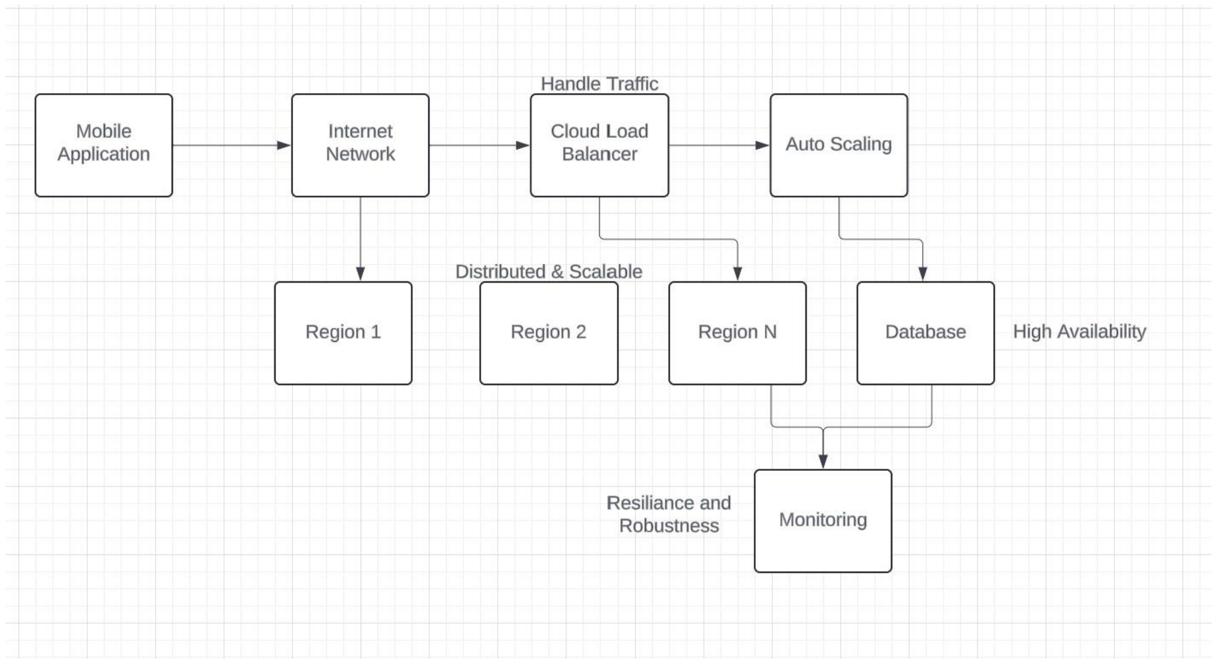




Order	Work Item Type	Title	State	Effort	Busin...	Value Area	Tags
1	Epic	UI/UX	New			Business	
	Feature	Simple UI/UX	New			Business	
	User Story	As a user, I want to understand and navigate every page of web application to use them	New			Business	
	Task	Design login page	New				
2	Epic	User Management	New			Business	
	Feature	User Authentication	New			Business	
	User Story	As a user, I want to be able to create an account and log in securely to access the software management system	New			Business	
	Task	Sign up page	New				
	Task	Sign in Page	New				
	User Story	As a user, i want to reset password at my will or when forgetten so that i can login to my account	New			Business	
	Task	Forget password	New				
	Task	Reset Password	New				
	Feature	User Roles and Permissions	New			Business	
	User Story	As an administrator, I want to assign different roles (e.g., admin, tester, developer) to users with varying permissions	New			Business	
	Task	Design a role-based access control system	New				
	Task	Implement user role assignment and permission management functionalities	New				
3	Epic	Issue Management	New			Business	
	Feature	Issue Tracking	New			Business	
	User Story	As a tester, I want to be able to report software issues and track their status throughout the resolution process	New			Business	
	Task	Develop an interface for testers	New				
	Task	Implement a dashboard for tracking issues	New				
	Feature	Priority-based Assignment	New			Business	
	User Story	As a project manager, I want high-priority issues to be automatically assigned to dedicated teams for immediate att...	New			Business	
	Task	Define priority levels	New				
	Task	Implement a mechanism to assign	New				
4	Epic	Collaboration Tools	New			Business	
	Feature	Real-time Communication	New			Business	
	User Story	As a team member, I want to communicate with other stakeholders in real-time to discuss issues and collaborate on...	New			Business	
more information	ask	Integrate a real-time messaging system	New				
	ask	Implement notifications for updates	New				
	Feature	Document Sharing	New			Business	
	User Story	As a developer, I want to share code snippets, design documents, and other relevant files with team members	New			Business	
	Task	Integrate a document management system	New				
	Task	Implement access controls	New				
5	Epic	Reporting and Analytics	New			Business	
	Feature	Performance Metrics	New			Business	
	User Story	As a project manager, I want to analyze software testing data to track project progress and identify bottlenecks	New			Business	
	Task	Develop reports to visualize key performance indicators (KPIs)	New				
	Task	Implement data analytics tools	New				
	Feature	Custom Reporting	New			Business	
	User Story	As a stakeholder, I want the ability to create custom reports tailored to specific project requirements	New			Business	
	Task	Design a interface for customizing reports	New				
	Task	Implementation of Reporting engine	New				

Tools Used:

- **CI/CD Pipeline**: Azure DevOps
- **Code Repository**: GitHub
- **Containerization**: Docker
- **Orchestration**: Kubernetes
- **Monitoring**: Azure Monitor
- **Logging**: Azure Log Analytics
- **Infrastructure as Code**: Terraform

Deployment Architecture of the application



- **User Devices (Web/Mobile App):**Users interact with the application through their devices.
- **Internet:** Public network connection for users to access the application.
- **Cloud Load Balancer:** Distributes incoming traffic across multiple instances of the messaging system in different regions for optimal performance and scalability.
- **Auto Scaling:** These groups manage instances of your messaging system across different Availability Zones (AZs) within a cloud region. They automatically scale the number of instances up or down based on traffic demands.
- **Messaging System:** This represents the core application logic deployed across multiple instances within ASGs for redundancy and scalability.
- **Database:** A highly available database service deployed across multiple AZs within regions ensures data persistence and accessibility.
- **Monitoring:** Continuously monitors the health and performance of the messaging system and database, providing alerts and logs for troubleshooting and scaling decisions.

Benefits of this Architecture:

- High Availability
- Scalability
- Resilience
- Performance
- Observability