SOFTWARE ENGINEERING & CONCEPTS – LAB MANUAL

[Document Subtitle]

<<Author Name>> <<Author ID>>

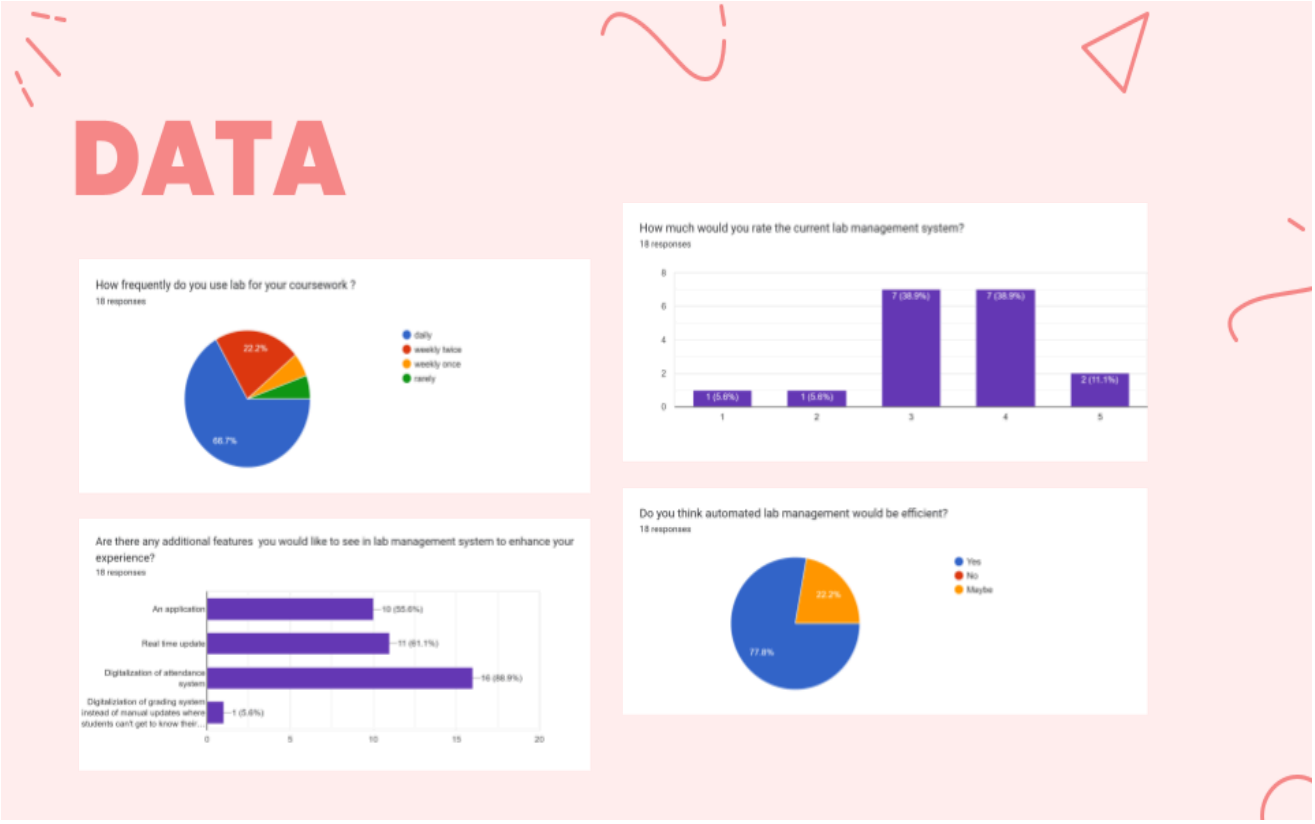
<<Degree>> <<Year>> <<Section>>

Software Concepts & Engineering - Lab Manual

Overview of the Project

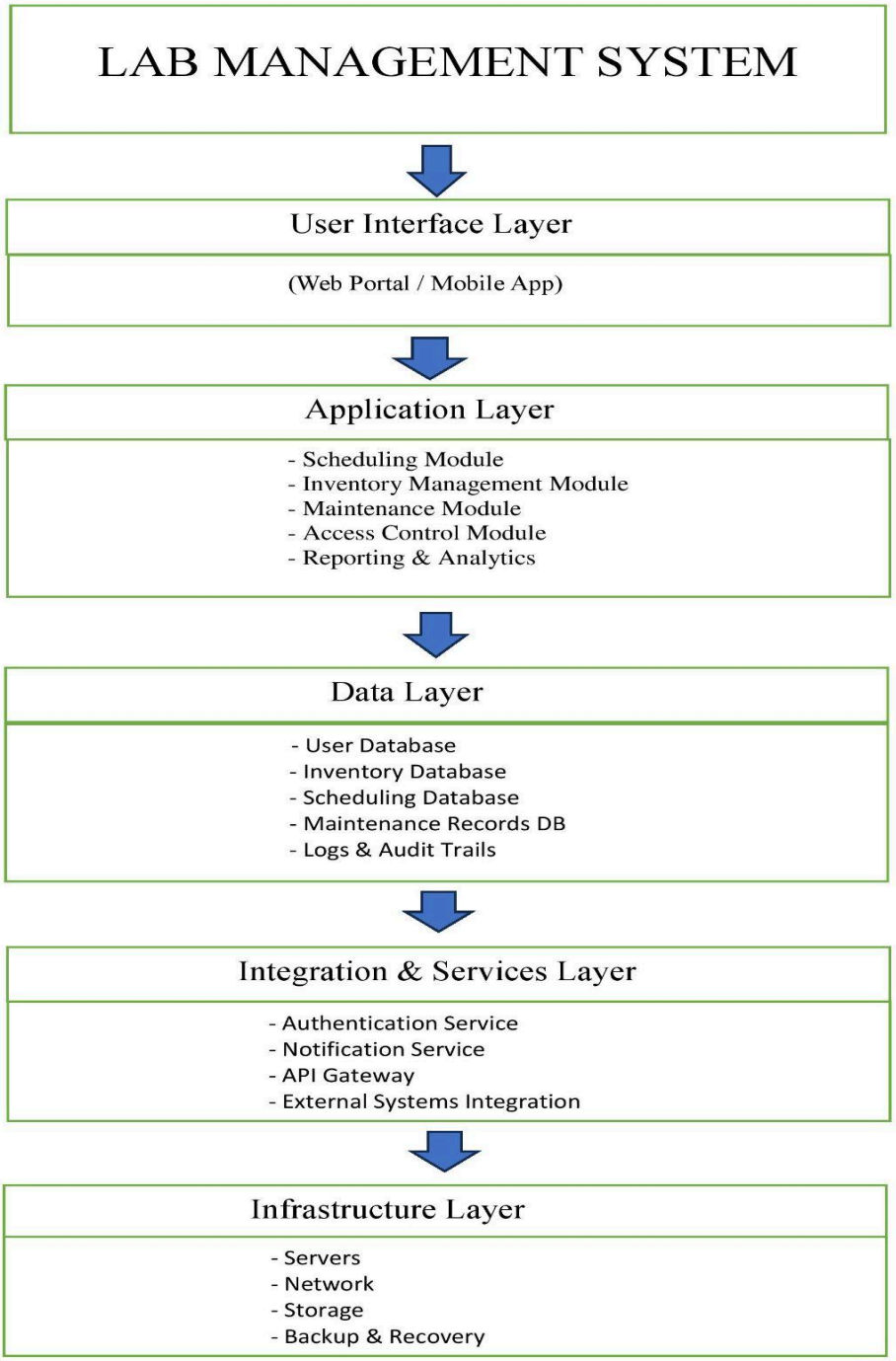
LAB MANAGEMENT SYSTEM

The lab management system aims to resolve **inefficiencies in lab scheduling, inventory management, user management, and communication**. It addresses issues like **double-booking of lab resources, manual inventory tracking, inconsistent user access control, and delayed notifications**, ultimately improving lab operations and resource utilization.



Implementing the lab management system will streamline lab operations, reduce administrative burdens, and enhance user satisfaction. Users can easily book lab resources, receive timely notifications, and access accurate inventory information. The system ensures secure and efficient lab management, ultimately improving productivity and user experience.

Business Architecture Diagram



Business Need of our project:

The main business need for creating a College Lab Management System is to streamline and optimize laboratory operations within a college environment.
This system will enhance data accuracy, improve workflow efficiency, ensure regulatory compliance, and provide a centralized platform for managing lab resources.

Current Process (manual process):

In many institutions, lab management is still a largely manual process involving the following steps:

1. Scheduling: Lab schedules are maintained in physical registers or basic digital tools like spreadsheets.
2. Resource Allocation: Equipment and materials are tracked manually, often leading to discrepancies in availability.
3. Maintenance: Regular maintenance logs are kept in paper files, making it difficult to track the history of equipment service.
4. Access Control: Keys to the lab are issued manually, with a sign-in/sign-out process for students and staff.

Personas and How the Current Process Works for Them:

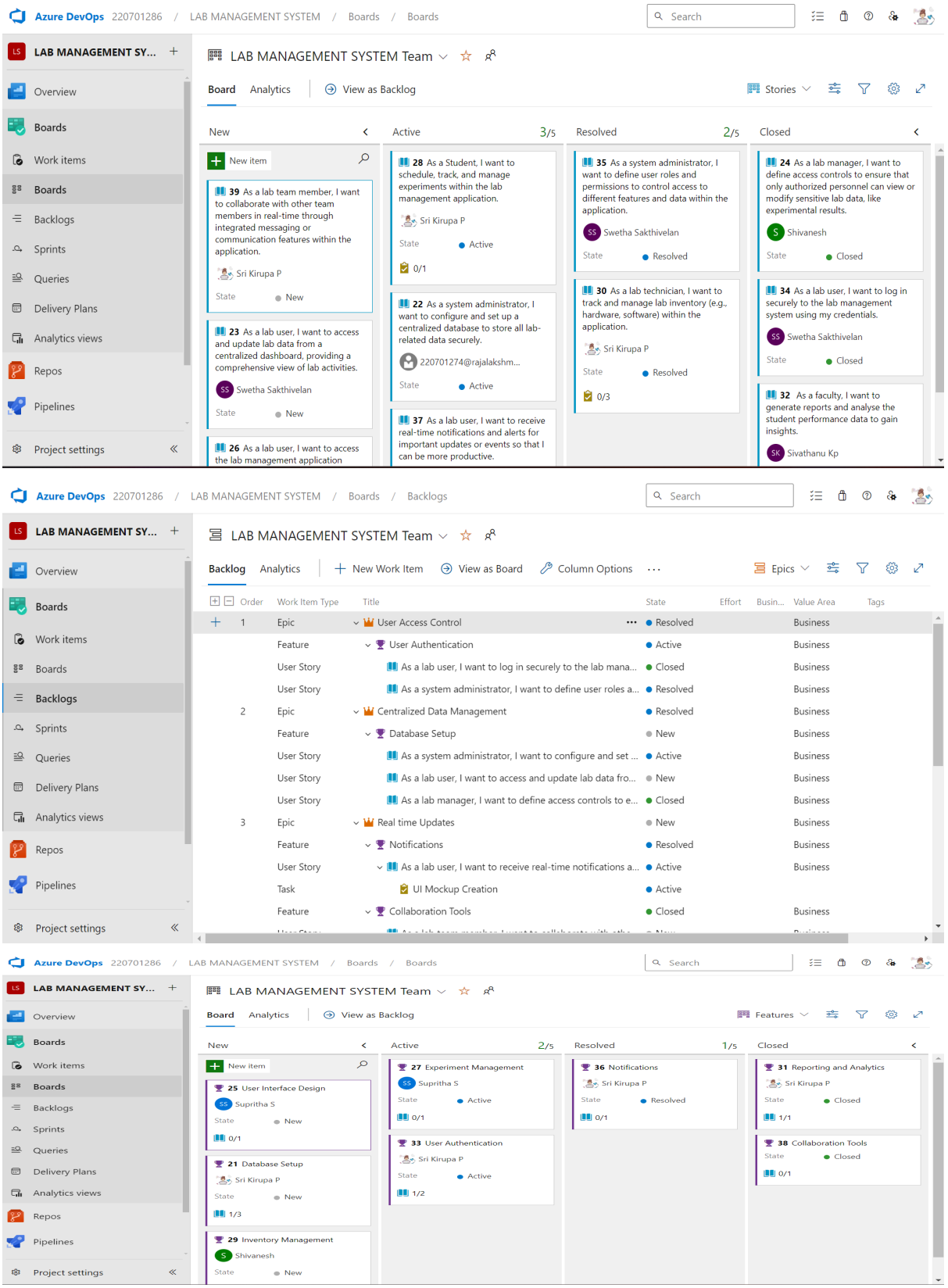
1. Lab Administrators
    - Manual Process: Lab administrators spend significant time scheduling, managing inventory, and handling maintenance records manually. This is time-consuming and prone to errors.
    - Automatic Process: Some tasks are automated, but administrators still need to input data and manage systems, often dealing with multiple unintegrated tools.
2. Instructors
    - Manual Process: Instructors manually check lab availability, book slots, and ensure that necessary equipment is available for their sessions. This can lead to conflicts and inefficiencies.
    - Automatic Process: They use basic scheduling software but may still face issues with resource allocation and last-minute changes.

3. Students
    - Manual Process: Students have to sign up for lab sessions through physical sign-up sheets or by contacting administrators directly. Access to lab schedules and available resources is limited.
    - Automatic Process: Students may use simple digital systems to view schedules and book lab time, but these systems often lack real-time updates and comprehensive resource information.
4. Maintenance Staff
    - Manual Process: Maintenance staff rely on physical logs and direct communication for their tasks. Tracking maintenance history and scheduling tasks is inefficient.
    - Automatic Process: Digital logs are used, but often these are not integrated with other systems, making it hard to get a holistic view of equipment status.

Business Problems:

1. Inefficiency and Time Consumption
2. Resource Mismanagement
3. Scheduling Conflicts
4. Limited Accessibility and Transparency
5. Inadequate Maintenance Tracking

Requirements as User Stories



- Poker planning methodology

As a Student, I want to schedule, track, and manage experiments within the lab management application.

- Estimate: 8 points

- Rationale: Implementing scheduling, tracking, and management features involves both frontend and backend development, including UI design, database setup, and user interaction functionalities.

As a System Administrator, I want to configure and set up a centralized database to store all lab-related data securely.

- Estimate: 13 points

- Rationale: Setting up a secure, scalable database requires significant planning, database design, implementation, and testing, as well as ensuring compliance with security standards.

As a Lab User, I want to receive real-time notifications and alerts for important updates or events so that I can be more productive.

- Estimate: 5 points

- Rationale: Implementing real-time notifications involves setting up a notification system, integrating it with the application, and handling user preferences, which can be moderately complex.

As a Lab Team Member, I want to collaborate with other team members in real-time through integrated messaging or communication features within the application.

- Estimate: 8 points

- Rationale: Integrating real-time messaging features requires implementing communication protocols, UI design, backend infrastructure, and ensuring data privacy and security.

As a Lab User, I want to access and update lab data from a centralized dashboard, providing a comprehensive view of lab activities.

- Estimate: 8 points

- Rationale: Developing a centralized dashboard involves frontend design, backend integration, data visualization, and ensuring data accuracy and accessibility.

As a Lab User, I want to access the lab management application through an intuitive and user-friendly web interface.

- Estimate: 5 points

- Rationale: Developing an intuitive web interface involves UI/UX design, frontend development, and usability testing, which can be moderately complex.

As a System Administrator, I want to define user roles and permissions to control access to different features and data within the application.

- Estimate: 8 points

- Rationale: Implementing role-based access control (RBAC) requires defining roles, permissions, and access levels, as well as integrating them with the application's authentication system.

As a Lab Technician, I want to track and manage lab inventory (e.g., hardware, software) within the application.

- Estimate: 8 points

- Rationale: Implementing inventory management features involves database design, frontend development, barcode/RFID integration (if applicable), and ensuring data accuracy and integrity.

As a System Administrator, I need to generate customizable reports to audit user access and permissions within the lab management software to ensure data security, privacy, and regulatory compliance.

- Estimate: 13 points

- Rationale: Developing customizable reporting features requires designing report templates, querying data, generating dynamic reports, and ensuring compliance with regulatory standards.

As a Lab Administrator, I want to optimize resource allocation based on analytics insights to ensure efficient use of lab facilities, equipment, and personnel.

- Estimate: 13 points

- Rationale: Implementing analytics features involves data collection, analysis, visualization, and providing actionable insights, which can be complex and require collaboration with data scientists or analysts.

Non Functional Requirements:

Security:

Our system ensures the confidentiality, integrity, and availability of data by implementing encryption, multi-factor authentication, and regular security audits.

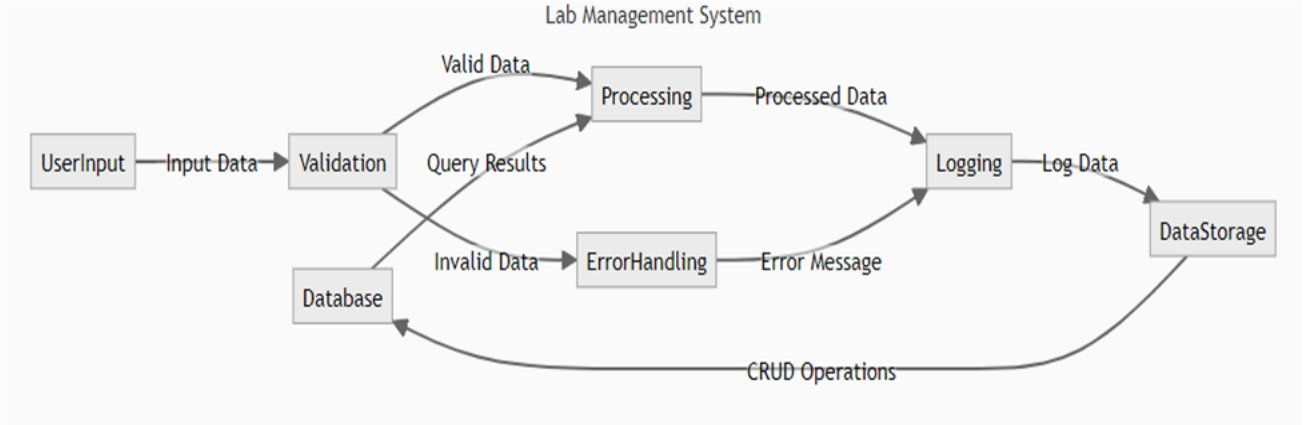
Performance:

Our system performs efficiently under expected load conditions, handling up to 500 concurrent users with response times not exceeding 2 seconds for any query or transaction.

Usability:

Our system is user-friendly and easy to navigate, featuring a clean, intuitive interface, context-sensitive help.

Architecture Diagram



1. **Client Apps:**
   Interfaces for users to interact with the system via web, mobile, or API.
2. **API Gateway:**
   Acts as a single entry point for all client requests. Handles routing, authentication, and request/response transformation.
3. **Auth Service:**
   Manages authentication and authorization. Utilizes OAuth2/JWT tokens for secure access.
4. **User Management Service:**
   Manages user profiles, roles, and permissions.
5. **Booking Service:**
   Handles lab booking, scheduling, and cancellations.
6. **Inventory Service:**
   Manages lab inventory, equipment check-in/check-out, and stock levels.
7. **Notification Service:**
   Sends notifications (email, SMS) for bookings, cancellations, and alerts.
8. **Report Service:**
   Generates and manages reports on lab usage, inventory, and user activity.
9. **Database Service:**
   Interfaces with data storage systems, including SQL/NoSQL databases and cache.
10. **Logging Service:**
    Collects and stores logs for auditing, monitoring, and debugging purposes.
11. **Data Storage:**
    Backend storage systems including relational databases (SQL), NoSQL databases (MongoDB, etc.), and file storage systems.

Architecture Pattern

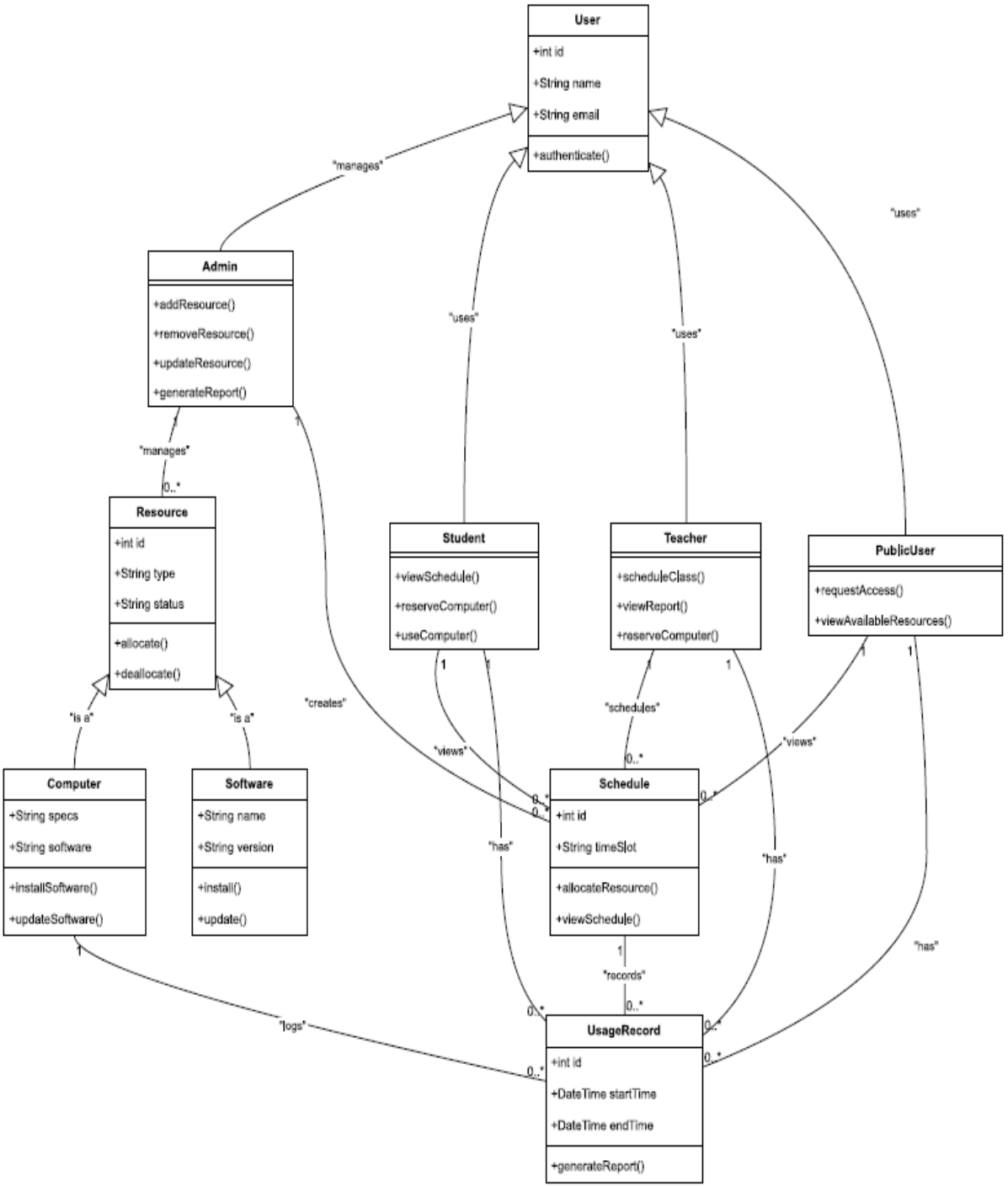
Microservices Architecture

- Why:
  - Scalability: Services can be scaled independently based on demand.
  - Maintainability: Each service is isolated, making it easier to update or replace without affecting the entire system.
  - Flexibility: Different technologies and languages can be used for different services.
  - Resilience: Failure in one service does not necessarily bring down the entire system.

Design Principles

- Single Responsibility Principle (SRP):
  - Each microservice handles a specific business function, making the system easier to maintain and extend.
- Separation of Concerns:
  - Different aspects of the application are managed by different services (e.g., authentication, booking, inventory).
- Loose Coupling:
  - Services interact through well-defined APIs, reducing dependencies and allowing independent deployment and scaling.
- High Cohesion:
  - Related functionalities are encapsulated within the same service, enhancing clarity and maintainability.
- API Gateway:
  - Centralized point for routing and aggregating requests, improving security and efficiency.
- Resilience and Fault Tolerance:
  - Implementing error handling and retry mechanisms ensures the system remains operational despite individual service failures.
- Scalability:
  - Each microservice can be scaled independently based on its load and performance requirements.
- Observability:
  - Comprehensive logging, monitoring, and error tracking are integrated to provide insights into system performance and issues.

Class diagrams



Classes and Their Relationships

1. User Class
    - Attributes: id, name, email
    - Methods: authenticate()
    - Description: This is the base class representing a user in the system. It includes basic user information and an authentication method.
2. Admin Class
    - Inherits from User
    - Methods: addResource(), removeResource(), updateResource(), generateReport()
    - Description: Represents an administrator with permissions to manage resources, schedules, and generate reports.
3. Student Class
    - Inherits from User
    - Methods: viewSchedule(), reserveComputer(), useComputer()
    - Description: Represents a student user who can view schedules, reserve, and use computers.
4. Teacher Class
    - Inherits from User
    - Methods: scheduleClass(), viewReport(), reserveComputer()
    - Description: Represents a teacher user who can schedule classes, view reports, and reserve computers.
5. PublicUser Class
    - Inherits from User
    - Methods: requestAccess(), viewAvailableResources()
    - Description: Represents a public user who can request access and view available resources.
6. Resource Class
    - Attributes: id, type, status
    - Methods: allocate(), deallocate()
    - Description: This is a general class for resources in the system. It includes methods to allocate and deallocate resources.
7. Computer Class
    - Inherits from Resource
    - Attributes: specs, software
    - Methods: installSoftware(), updateSoftware()
    - Description: Represents a computer resource with specific attributes and methods related to software management.
8. Schedule Class
    - Attributes: id, timeSlot
    - Methods: allocateResource(), viewSchedule()
    - Description: Represents a schedule for allocating resources and viewing schedules.
9. UsageRecord Class

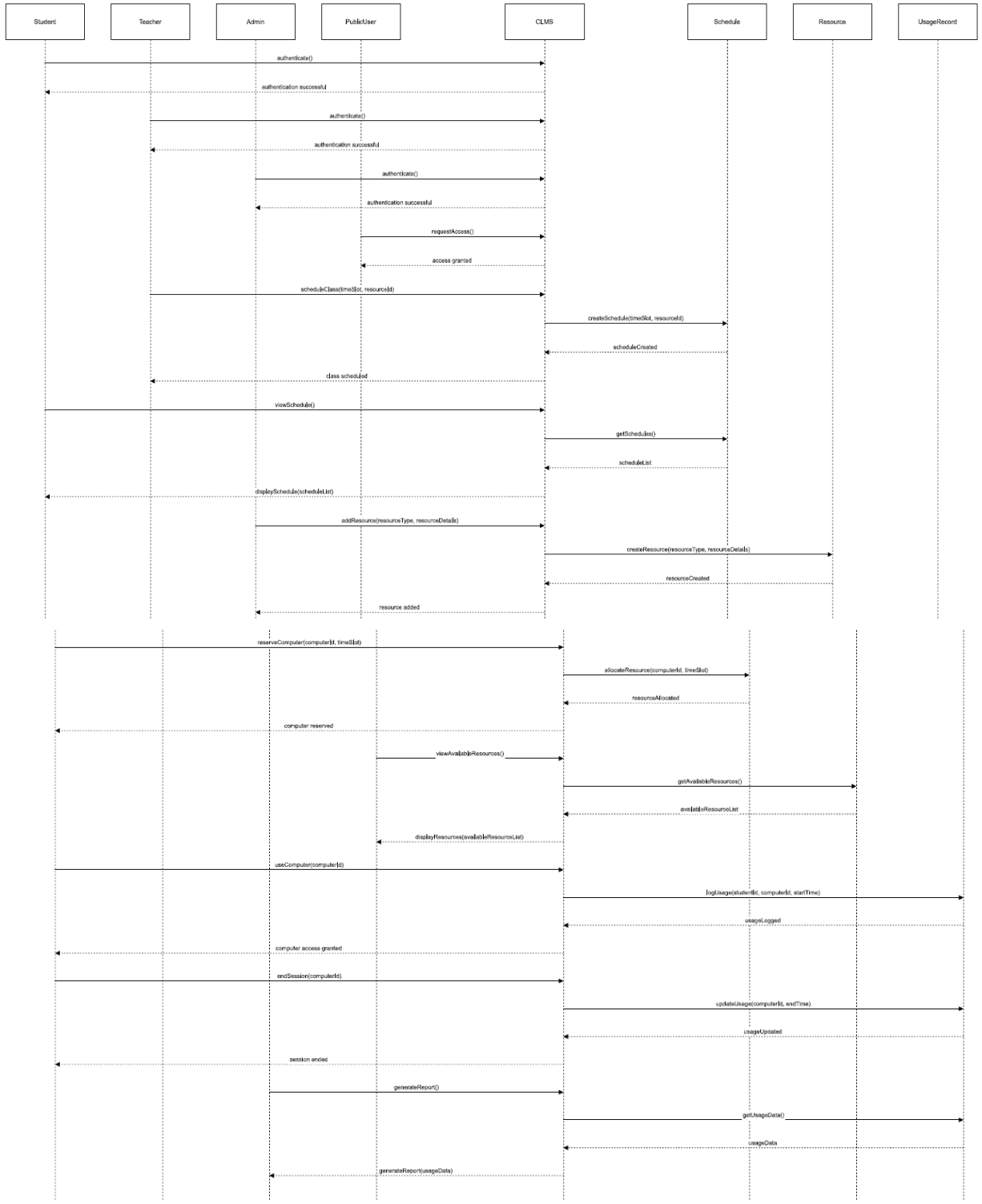
- Attributes: id, startTime, endTime
- Methods: generateReport()
- Description: Represents a record of resource usage, including methods to generate reports on usage.

10. Software Class
    - Inherits from Resource
    - Attributes: name, version
    - Methods: install(), update()
    - Description: Represents software resources with attributes for name and version, and methods for installation and updating.

Relationships

- Inheritance Relationships
    - Admin, Student, Teacher, and PublicUser inherit from User, indicating they are all types of users with some common attributes and behaviors.
    - Computer and Software inherit from Resource, indicating they are specific types of resources with additional properties and methods.
- Association Relationships
    - Admin manages Resources and creates Schedules: An admin can manage various resources and create schedules.
    - Computer logs UsageRecords: A computer resource can have multiple usage records.
    - Schedule records UsageRecords: A schedule can have multiple usage records.
    - Students, Teachers, and PublicUsers interact with Schedules and UsageRecords: They can view schedules and have usage records associated with them.
    - Teachers schedule classes and view reports: Teachers have permissions related to scheduling classes and viewing reports.

- Sequence diagram



The sequence diagram illustrates the sequence of interactions between different actors (Student, Teacher, Admin, PublicUser) and the CLMS (Computer Lab Management System) in various scenarios within the system. Here's the breakdown of each step:

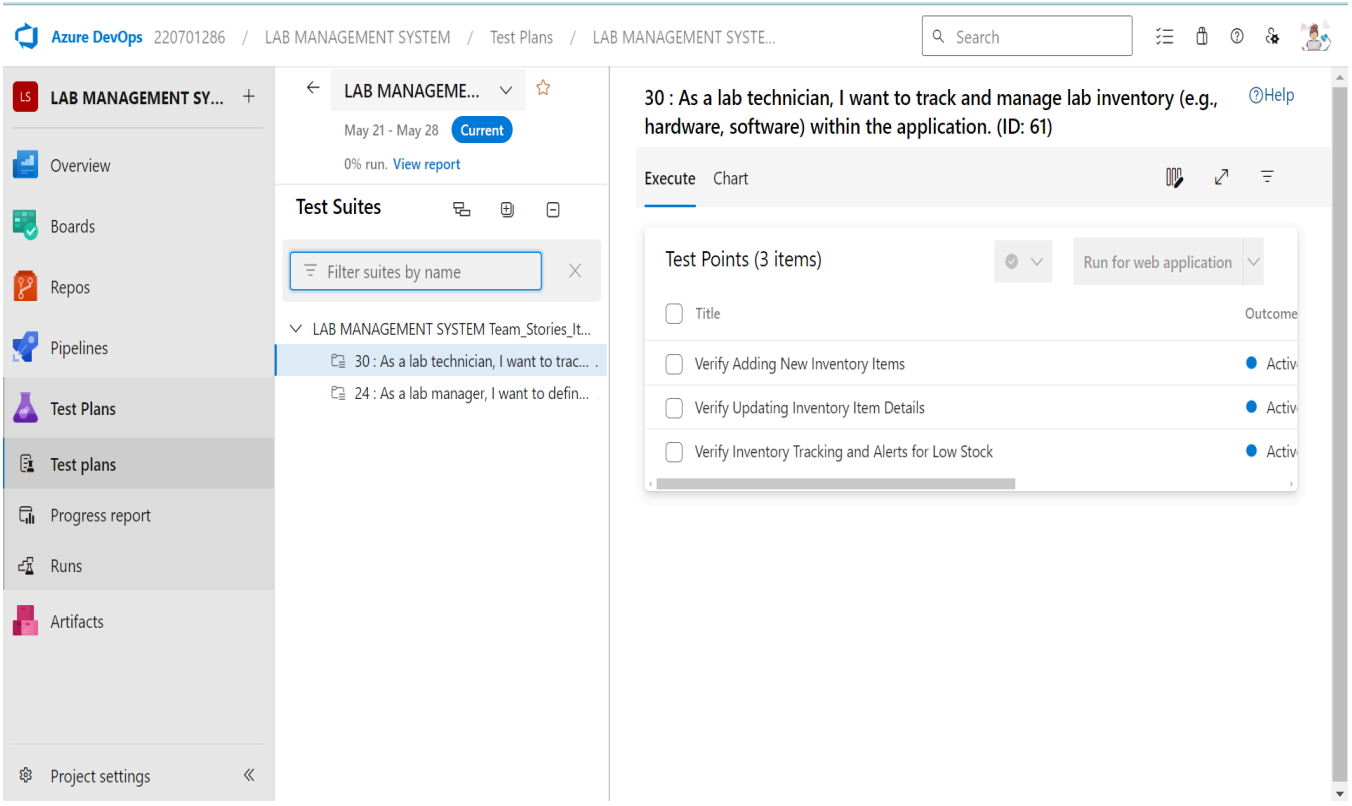
1. Authentication
    - Each type of user (Student, Teacher, Admin, PublicUser) initiates an authentication process with the CLMS by calling the `authenticate()` method.
    - The CLMS verifies the credentials, and upon successful authentication, it sends a confirmation back to the respective user.
2. Request Access
    - A PublicUser requests access to the system by calling the `requestAccess()` method.
    - Upon receiving the request, the CLMS grants access to the PublicUser.
3. Scheduling a Class
    - A Teacher schedules a class by invoking the `scheduleClass(timeSlot, resourceId)` method.
    - The CLMS creates a schedule for the specified time slot and resource (e.g., a computer lab).
    - Upon successful creation, the CLMS informs the Teacher about the scheduled class.
4. Viewing Schedule
    - A Student requests to view the schedule by calling the `viewSchedule()` method.
    - The CLMS retrieves the list of schedules from the Schedule component.
    - It then displays the schedule to the Student.
5. Adding a Resource
    - An Admin adds a new resource to the system by calling the `addResource(resourceType, resourceDetails)` method.
    - The CLMS creates the resource with the provided details.
    - Upon successful addition, the CLMS notifies the Admin about the resource being added.
6. Reserving a Computer
    - A Student reserves a computer for a specified time slot by invoking the `reserveComputer(computerId, timeSlot)` method.
    - The CLMS allocates the computer for the specified time slot in the schedule.
    - Upon successful reservation, the CLMS confirms the reservation to the Student.
7. Viewing Available Resources
    - A PublicUser requests to view available resources by calling the `viewAvailableResources()` method.
    - The CLMS retrieves the list of available resources.
    - It then displays the list of available resources to the PublicUser.
8. Using a Computer
    - A Student initiates using a computer by calling the `useComputer(computerId)` method.
    - The CLMS logs the usage record for the student and the computer.

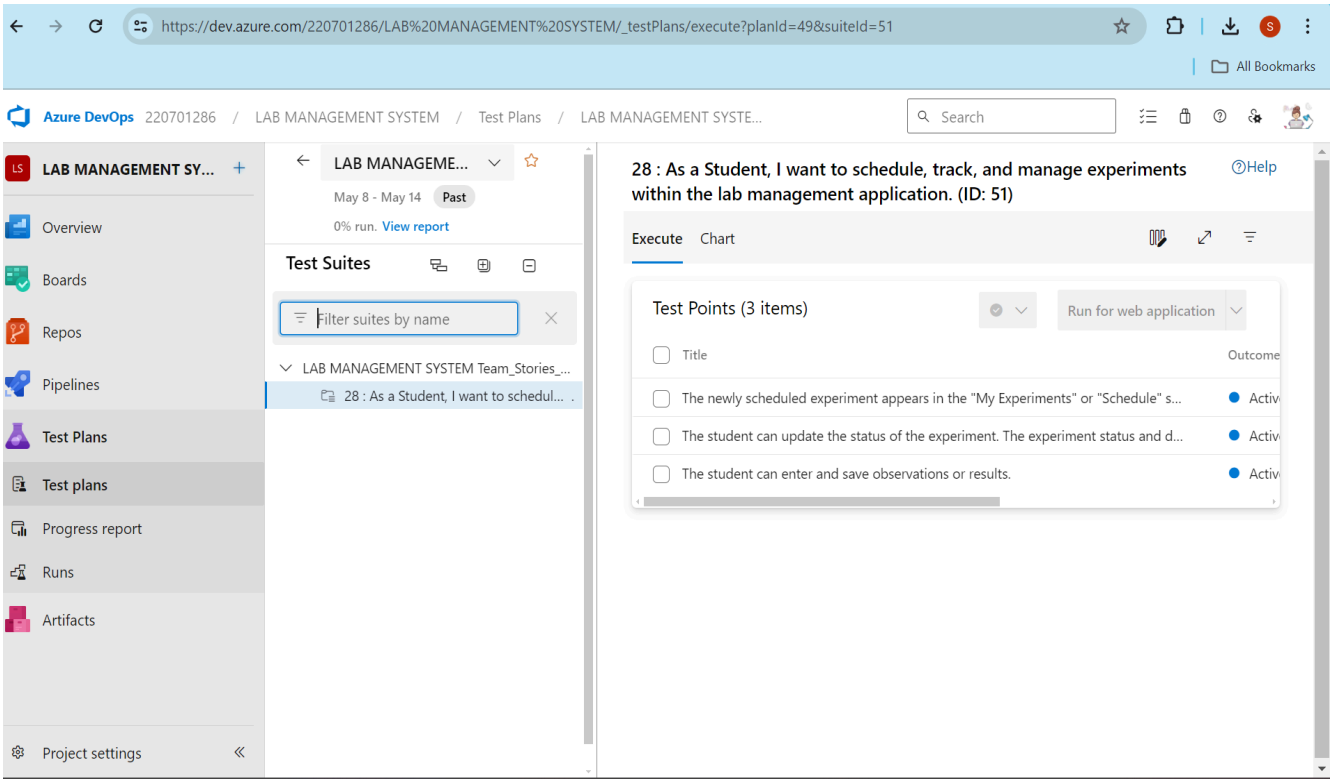
- Upon successful logging, the CLMS grants access to the computer to the Student.

9. Ending Session
    - A Student ends the session on a computer by calling the `endSession(computerId)` method.
    - The CLMS updates the usage record with the end time of the session.
    - Upon successful update, the CLMS notifies the Student about the session ending.
10. Generating Report
    - An Admin generates a report by calling the `generateReport()` method.
    - The CLMS retrieves the usage data from the UsageRecord component.
    - It then generates the report based on the usage data and sends it to the Admin.

Test Strategy

- Document the Test Plans





- Test cases for at least 5 user stories showcasing the Happy Path and the Error Scenarios

Test Cases for User Stories

User Story: As a Student, I want to schedule, track, and manage experiments within the lab management application.

Test Case 1: Schedule Experiment (Happy Path)

- **Precondition:** Student is logged into the system.
- **Steps:**
  1. Navigate to the "Schedule Experiment" page.
  2. Select a date and time.
  3. Enter experiment details (title, description).
  4. Click "Save."
- **Expected Result:** Experiment is successfully scheduled and appears on the student's dashboard.

Test Case 2: Schedule Experiment (Error Scenario)

- **Precondition:** Student is logged into the system.
- **Steps:**
  1. Navigate to the "Schedule Experiment" page.
  2. Select a past date and time.
  3. Enter experiment details (title, description).
  4. Click "Save."
- **Expected Result:** Error message is displayed: "Cannot schedule an experiment in the past."

User Story: As a System Administrator, I want to configure and set up a centralized database to store all lab-related data securely.

Test Case 1: Configure Database (Happy Path)

- **Precondition:** Admin is logged into the system.
- **Steps:**
  1. Navigate to the "Database Configuration" page.
  2. Enter database credentials and connection details.
  3. Click "Test Connection."
  4. Click "Save."

- **Expected Result:** Database is successfully configured, and a confirmation message is displayed.

Test Case 2: Configure Database (Error Scenario)

- **Precondition:** Admin is logged into the system.
- **Steps:**
  1. Navigate to the "Database Configuration" page.
  2. Enter incorrect database credentials.
  3. Click "Test Connection."
- **Expected Result:** Error message is displayed: "Database connection failed. Please check your credentials and try again."

User Story: As a Lab User, I want to receive real-time notifications and alerts for important updates or events so that I can be more productive.

Test Case 1: Receive Real-Time Notifications (Happy Path)

- **Precondition:** User is logged into the system.
- **Steps:**
  1. Schedule an experiment.
  2. Receive a notification 10 minutes before the scheduled time.
- **Expected Result:** Notification is received in real-time.

Test Case 2: Receive Real-Time Notifications (Error Scenario)

- **Precondition:** User is logged into the system.
- **Steps:**
  1. Turn off notifications in user settings.
  2. Schedule an experiment.
  3. Wait for notification time.
- **Expected Result:** No notification is received as per user settings.

User Story: As a Lab Team Member, I want to collaborate with other team members in real-time through integrated messaging or communication features within the application.

Test Case 1: Real-Time Messaging (Happy Path)

- **Precondition:** Team member is logged into the system.
- **Steps:**
  1. Open the messaging feature.
  2. Select a team member.

3. Send a message.

- **Expected Result:** Message is delivered in real-time and a response can be seen instantly.

Test Case 2: Real-Time Messaging (Error Scenario)

- **Precondition:** Team member is logged into the system.
- **Steps:**
  1. Open the messaging feature.
  2. Select a team member who is offline.
  3. Send a message.
- **Expected Result:** Message is queued and delivered when the team member comes online.

User Story: As a Lab User, I want to access and update lab data from a centralized dashboard, providing a comprehensive view of lab activities.

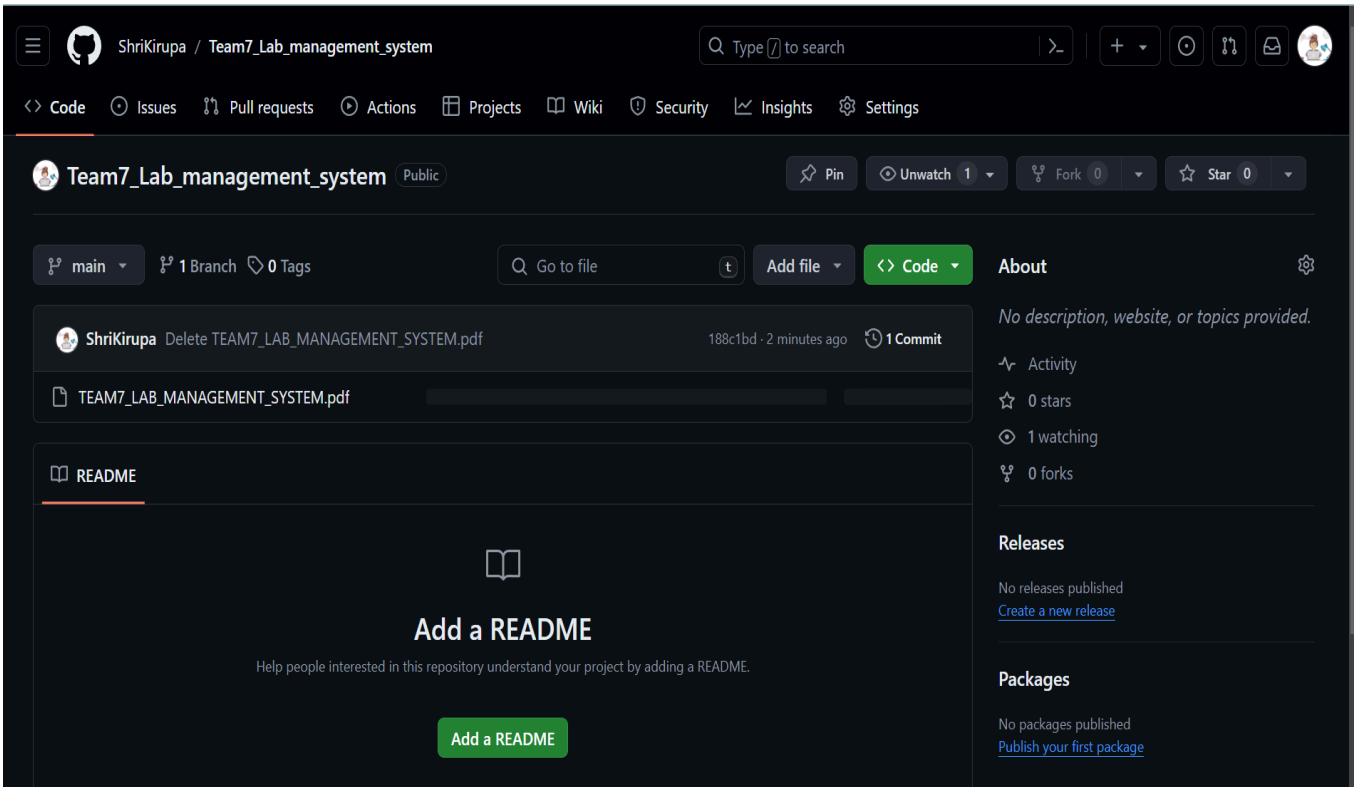
Test Case 1: Access and Update Data (Happy Path)

- **Precondition:** User is logged into the system.
- **Steps:**
  1. Navigate to the dashboard.
  2. View lab activities.
  3. Update lab data (e.g., change experiment details).
  4. Save changes.
- **Expected Result:** Data is updated successfully and reflects on the dashboard.

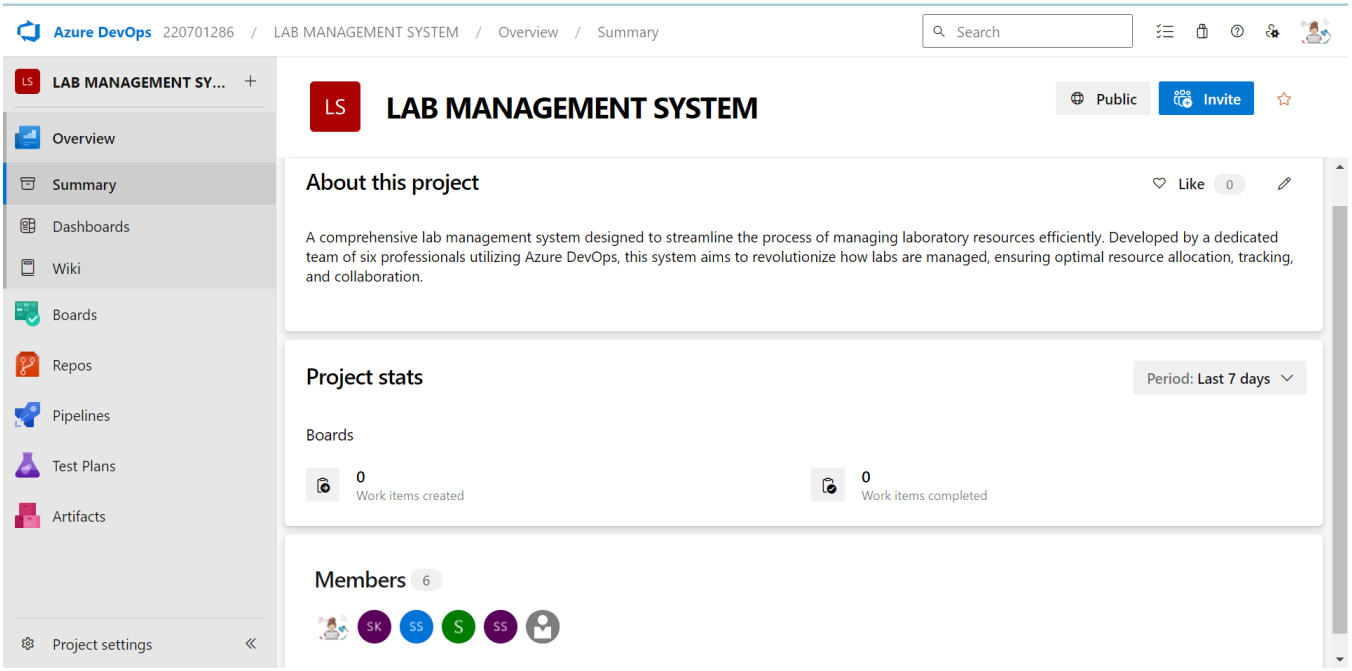
Test Case 2: Access and Update Data (Error Scenario)

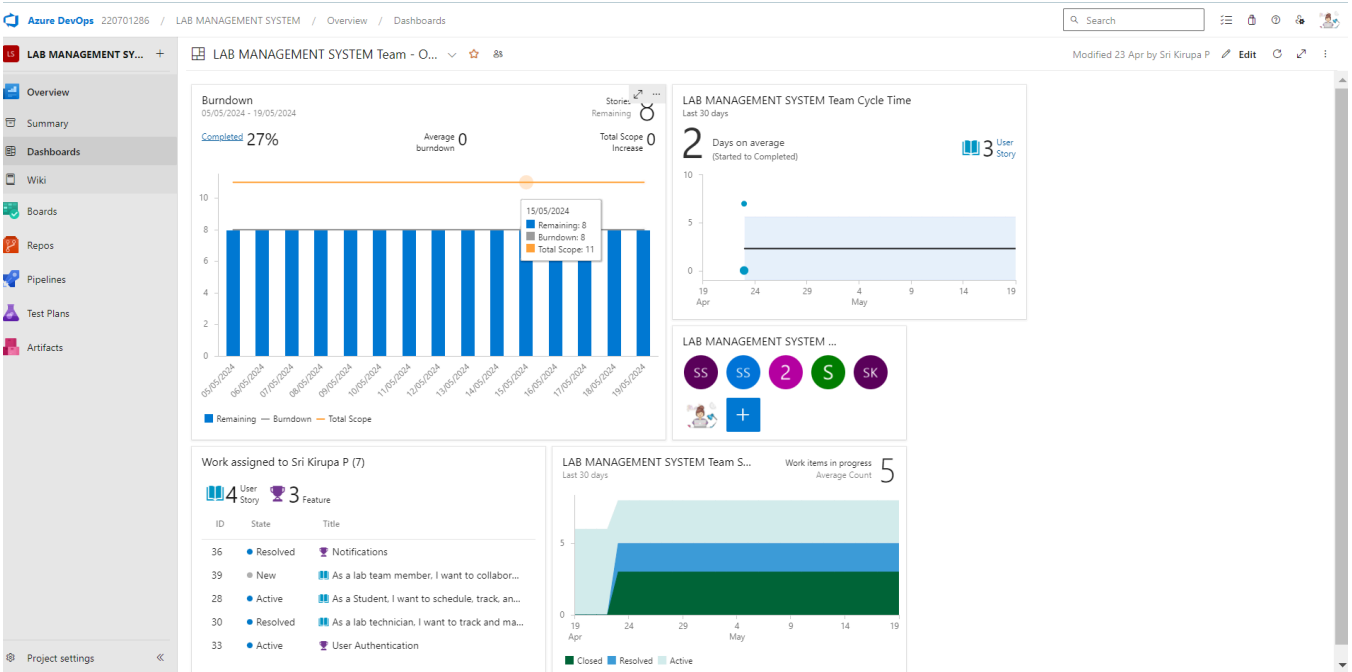
- **Precondition:** User is logged into the system.
- **Steps:**
  1. Navigate to the dashboard.
  2. Try to update lab data with invalid inputs (e.g., missing required fields).
  3. Click "Save."
- **Expected Result:** Error message is displayed: "Please fill in all required fields."

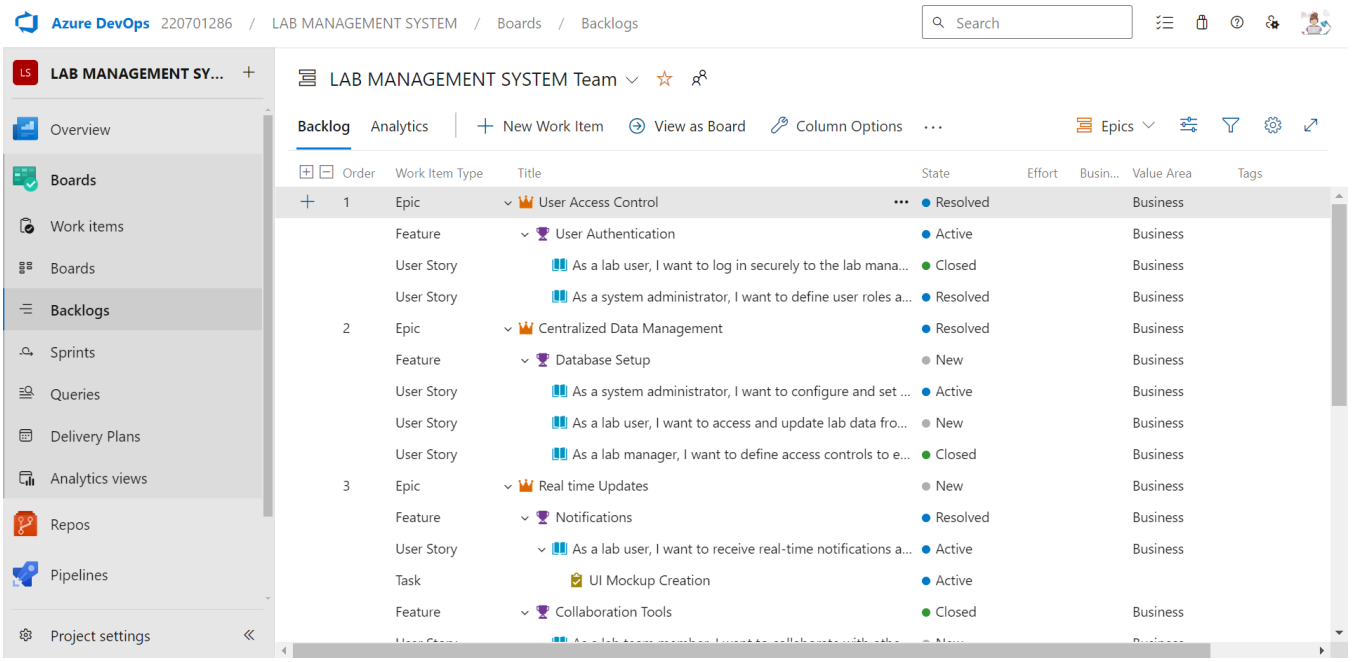
- A view of the github repository showcasing the Project structure, their naming conventions

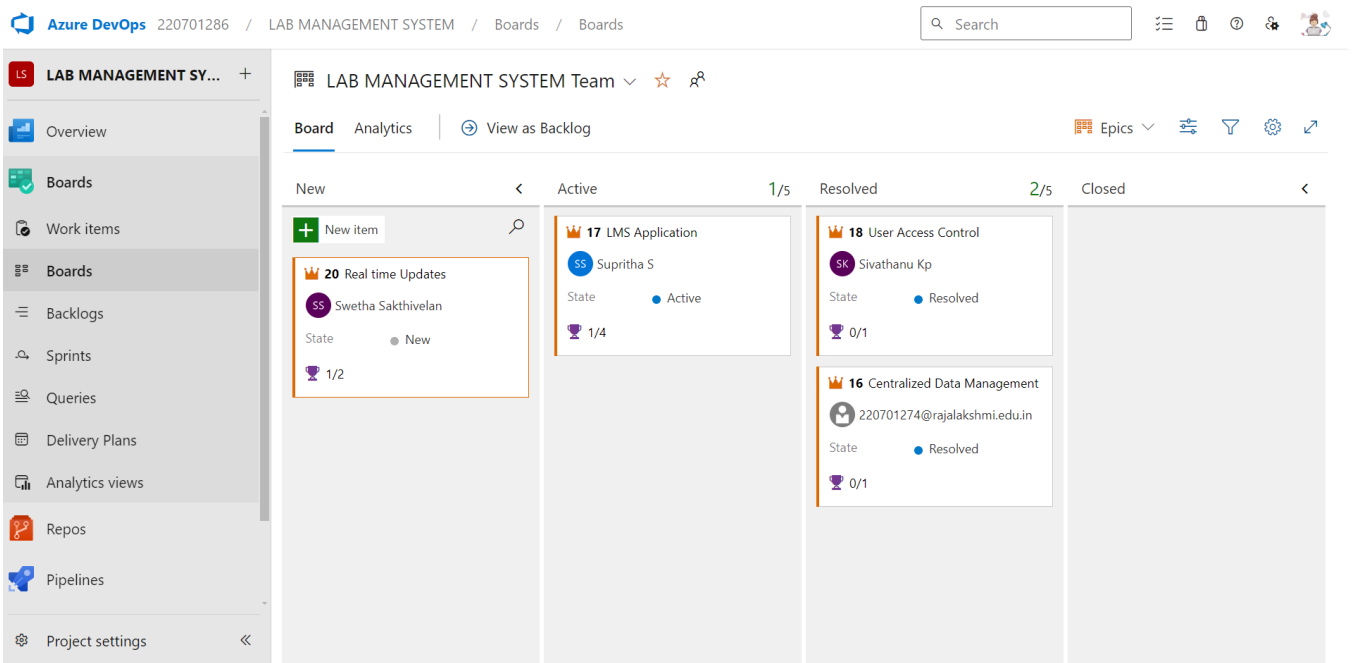


- A view of their DevOps Architecture for their respective project and the associated tools used in Azure






Azure DevOps 220701286 / LAB MANAGEMENT SYSTEM / Boards / Backlogs
Search
LAB MANAGEMENT SY...
Overview
Boards
Work items
Boards
Backlogs
Sprints
Queries
Delivery Plans
Analytics views
Repos
Pipelines
Project settings
LAB MANAGEMENT SYSTEM Team
Backlog
Analytics
New Work Item
View as Board
Column Options
Epics
Order
Work Item Type
Title
State
Effort
Busin...
Value Area
Tags
1 Epic User Access Control Resolved Business
Feature User Authentication Active Business
User Story As a lab user, I want to log in securely to the lab mana... Closed Business
User Story As a system administrator, I want to define user roles a... Resolved Business
2 Epic Centralized Data Management Resolved Business
Feature Database Setup New Business
User Story As a system administrator, I want to configure and set ... Active Business
User Story As a lab user, I want to access and update lab data fro... New Business
User Story As a lab manager, I want to define access controls to e... Closed Business
3 Epic Real time Updates New Business
Feature Notifications Resolved Business
User Story As a lab user, I want to receive real-time notifications a... Active Business
Task UI Mockup Creation Active
Feature Collaboration Tools Closed Business


Azure DevOps 220701286 / LAB MANAGEMENT SYSTEM / Boards / Boards
Search
LAB MANAGEMENT SY...
Overview
Boards
Work items
Boards
Backlogs
Sprints
Queries
Delivery Plans
Analytics views
Repos
Pipelines
Project settings
LAB MANAGEMENT SYSTEM Team
Board
Analytics
View as Backlog
Epics
New
Active 1/5
Resolved 2/5
Closed
New item
20 Real time Updates
Swetha Sakthivelan
State New
1/2
17 LMS Application
Supritha S
State Active
1/4
18 User Access Control
Sivathanu Kp
State Resolved
0/1
16 Centralized Data Management
220701274@rajalakshmi.edu.in
State Resolved
0/1

Deployment Architecture of the application

Designing a deployment architecture for a Lab Management System (LMS) involves outlining the various components and how they interact within the infrastructure. This typically includes the front-end, back-end, database, and additional services like authentication and monitoring. Here's a comprehensive deployment architecture for a Lab Management System:

1. Architecture Overview

Components:

- **Web Client:** User interface for accessing the system.
- **API Gateway:** Manages API requests and routes them to appropriate services.
- **Application Servers:** Handle the business logic.
- **Database Server:** Stores persistent data.
- **Authentication Service:** Manages user authentication and authorization.
- **Load Balancer:** Distributes incoming requests across multiple servers.
- **Monitoring & Logging:** Keeps track of system performance and logs events.
- **Storage Service:** Manages file storage for lab reports, documents, etc.
- **Notification Service:** Manages email/SMS notifications.

2. Infrastructure Layers

Frontend Layer:

- **Web Client (Browser/SPA):** Built using frameworks like React, Angular, or Vue.js. Communicates with the API Gateway.

Backend Layer:

- **API Gateway:** Tools like AWS API Gateway, Nginx, or Kong.
- **Application Servers:** Hosted on platforms like AWS EC2, Azure VMs, or containerized using Docker and managed with Kubernetes.
    - **Business Logic Microservices:** Developed in languages like Node.js, Java, Python, etc.
    - **Authentication Service:** Using OAuth, JWT, or third-party services like Auth0.
- **Notification Service:** Integrates with services like Twilio, SendGrid, or Amazon SNS.

Data Layer:

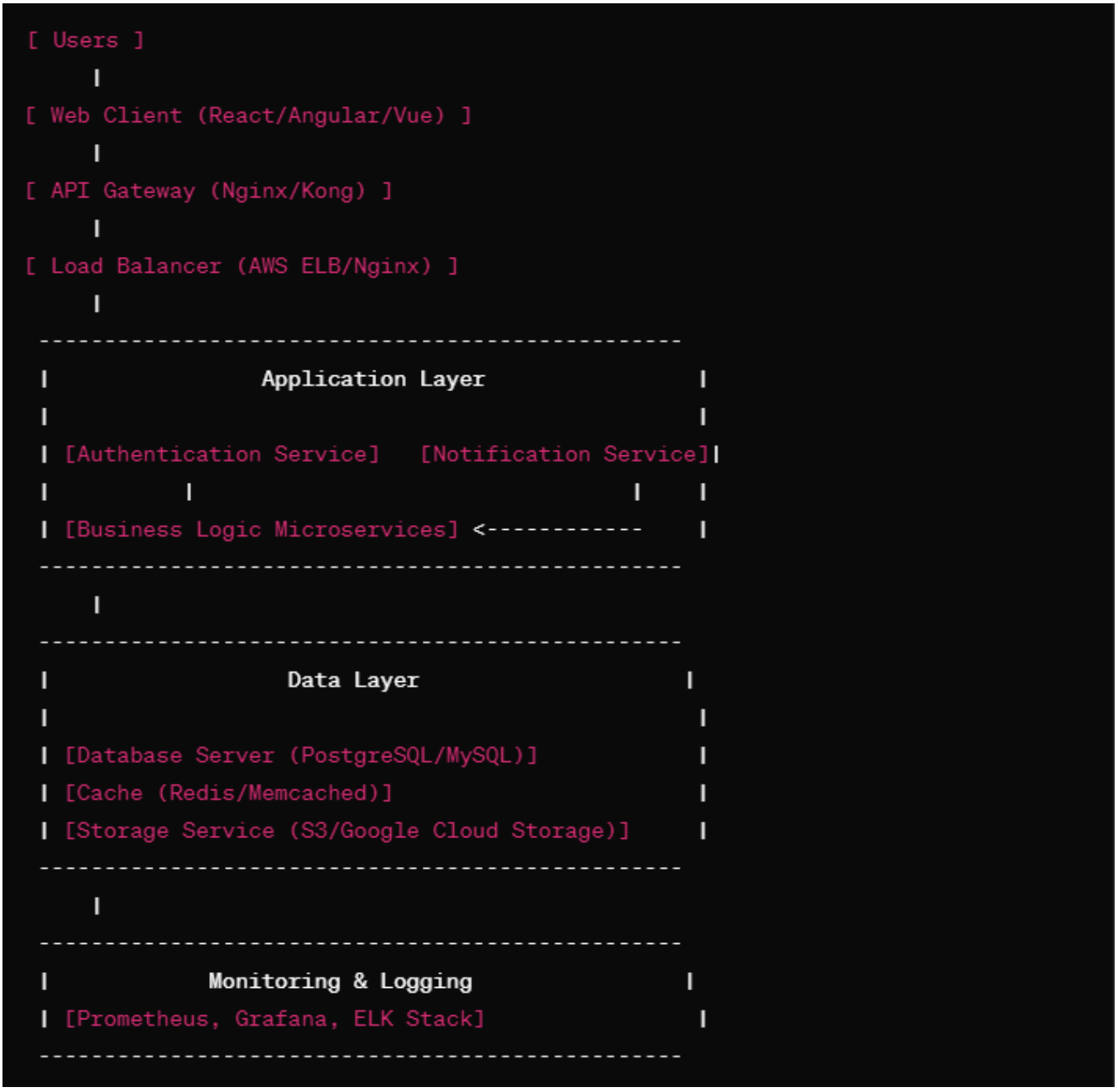
- **Database Server:** SQL (PostgreSQL, MySQL) or NoSQL (MongoDB, Cassandra).
- **Storage Service:** Amazon S3, Google Cloud Storage for document storage.
- **Cache:** Redis or Memcached for caching frequently accessed data.

Infrastructure Services:

- **Load Balancer:** AWS ELB, Nginx, or HAProxy to distribute traffic.

- **Monitoring & Logging:** Prometheus, Grafana, ELK Stack (Elasticsearch, Logstash, Kibana), or cloud services like AWS CloudWatch.

3. Deployment Diagram



4. High-Level Workflow

1. **User Interaction:**
    - Users interact with the LMS through a web client.
    - Requests are sent to the API Gateway.
2. **API Gateway:**
    - The API Gateway routes requests to the appropriate microservice.
    - It may also handle request validation and rate limiting.
3. **Application Layer:**
    - The Load Balancer distributes the incoming requests across multiple application servers to ensure high availability and reliability.
    - Application servers process the requests:
        - Business Logic Microservices handle specific functionalities of the LMS.
        - Authentication Service verifies user credentials and manages session tokens.
        - Notification Service sends notifications as needed.
4. **Data Layer:**
    - Application servers interact with the Database Server for persistent data storage.
    - Cache is used to store and retrieve frequently accessed data quickly.
    - Storage Service manages file storage for documents and reports.
5. **Monitoring & Logging:**
    - System performance is monitored using tools like Prometheus and Grafana.
    - Logs are collected and analyzed using the ELK stack.

5. Security Considerations

- **Authentication & Authorization:** Secure user authentication and role-based access control.
- **Data Encryption:** Encrypt data in transit (TLS) and at rest.
- **Regular Security Audits:** Regularly audit the system for vulnerabilities.
- **Network Security:** Use VPCs, firewalls, and security groups to restrict access.

6. Scalability & Availability

- **Auto-scaling:** Automatically scale application servers based on demand.
- **Redundancy:** Deploy multiple instances across different regions/zones.
- **Backup & Disaster Recovery:** Regular backups and a disaster recovery plan.

This architecture provides a robust, scalable, and secure deployment environment for a Lab Management System, ensuring high performance and reliability.