UNIVERSITY OF SRI JAYEWARDENEPURA

Faculty of Technology

Department of Information and Communication Technology

ITC3081 - ICT Project

FOT-MAIL-TRACK

Group 15

FOT-MAIL-TRACK

Revolutionizing Mails Tracking System of FOT

Details of Academic Supervisor:

Name : Mr. Bathiya Seneviratne.

Institution : University of Sri Jayewardenepura

Contact details: bathiyaseneviratne@sjp.ac.lk

Details of team members

Name	Specialized area	Index number	Email	Mobile Number
H.G.S.N Dilshan	Software Technology	ICT/21/832	Ict21832@fot.sjp.ac.lk	076-0890910
A.G.G.M.L Kumara	Software Technology	ICT/21/875	Ict21875@fot.sjp.ac.lk	078-2633528
P.M.M.N Gunasekara	Network Technology	ICT/21/841	Ict21841@fot.sjp.ac.lk	075-2303455
N.W.J.N Lakmal.	Multimedia Technology	ICT/21/878	Ict21878@fot.sjp.ac.lk	077-0170982
S.C Ediriweera	Software Technology	ICT/21/836	Ict21836@fot.sjp.ac.lk	078-2571795

Table of Contents

1. Introduction and Background

Introduction

The Faculty of Technology Mail Tracking System aims to modernize and streamline the process of handling and tracking physical mail within the faculty. With the advancement of technology, traditional mail management systems are becoming obsolete, often resulting in inefficiencies, delays, and lack of transparency. This project seeks to address these issues by developing a comprehensive system that integrates modern web and mobile technologies, ensuring secure, efficient, and transparent mail handling.

Background

Current Challenges

- **Manual Processes:**

The current system relies heavily on manual processes, leading to inefficiencies and errors in mail handling and tracking.

Manual logging and tracking of mail can result in misplaced or delayed letters, affecting communication and administrative processes.

- **Lack of Transparency:**

There is no real-time visibility into the status of mail as it passes through various departments and authorities.

Stakeholders often have to rely on follow-ups and inquiries to determine the location and status of their mail.

- **Security Concerns:**

Physical mail can be easily tampered with or lost, compromising the integrity and confidentiality of important documents.

Unauthorized access to sensitive mail can lead to data breaches and other security issues.

Objectives of the System

The Faculty of Technology Mail Tracking System aims to address these challenges by introducing an automated and secure system for managing mail. The key objectives include:

- **Automation and Efficiency:**

Automate the process of mail creation, tracking, and verification to reduce manual effort and errors.

Enable efficient handling and processing of mail within the faculty.

- **Real-time Tracking:**

Provide real-time visibility into the status and location of mail as it moves through different departments and authorities.

Enable stakeholders to track the progress of their mail through a user-friendly interface.

- **Enhanced Security:**

Ensure the integrity and confidentiality of mail by implementing secure QR codes for tracking and verification.

Restrict access to authorized personnel only, ensuring that sensitive mail is handled securely.

- **Technological Solution**

To achieve these objectives, the project will leverage modern web and mobile technologies, specifically the MERN stack for the web application and Flutter for the mobile application. The MERN stack (MongoDB, Express.js, React.js, Node.js) provides a robust and scalable framework for developing dynamic web applications, while Flutter allows for the creation of cross-platform mobile applications with a single codebase.

- **Web Application:**

Developed using the MERN stack, the web application will enable users to create and manage mail, generate QR codes, assign authorities, and track the status of mail.

Crystal Reports will be used to generate letterheads with unique QR codes for tracking.

- **Mobile Application:**

Developed using Flutter, the mobile application will allow authorized personnel to scan QR codes, verify mail, and update its status as it passes through different stages.

- **Database:**

MongoDB will serve as the central database, storing all relevant data, including user information, mail details, and tracking logs.

MongoDB Atlas, a cloud-based database service, will be used to ensure high availability, scalability, and secure access from both the web and mobile applications.

- **Deployment:**

Docker will be used to containerize the applications, ensuring consistent deployment across different environments and facilitating scalability.

The system will be deployed on a cloud platform to provide accessibility and reliability.

- **Conclusion**

The Faculty of Technology Mail Tracking System represents a significant advancement in the management and tracking of physical mail within the faculty. By leveraging modern technologies and addressing the current challenges, the system aims to enhance efficiency, transparency, and security, ultimately improving communication and administrative processes within the faculty

2. System Requirements

2.1 Project Objectives and Scope

2.1.1. Clearly Defined Objective

The Faculty of Technology Mail Tracking System aims to streamline and enhance the process of managing and tracking physical mail within the faculty. This system will consist of a web application developed using the MERN stack (MongoDB, Express.js, React.js, Node.js) and a mobile application developed using Flutter. The objective is to create a system that generates letters with unique QR codes, assigns authorities for the mail to pass through, and manages the overall workflow. The mobile application will allow users to scan QR codes to verify the passage of mail through designated authorities. Docker will be used for deployment to ensure scalability and manageability.

2.1.2 Detailed Scope of the Project, Including Deliverables

- Web Application:
  - Generate letters with unique QR codes using Crystal Reports.
  - Assign authorities for mail passage.
  - Manage and track the workflow of mail.
  - Store data in a MongoDB database.
  - Deploy using Docker.

- Mobile Application:
  - Scan QR codes to verify mail passage.
  - Update the status of mail in real-time.
  - User-friendly interface for ease of use.

- Deliverables:
  - Functional web application for generating and managing mail.
  - Mobile application for scanning and updating mail status.
  - Documentation detailing the system's design, development, and deployment process.
  - Deployment on cloud services for web hosting and mobile app publishing on Google Play Store.

2.2 Budget and Cost Estimates

2.2.1 Detailed Budget Breakdown

- **Who are involved in the project?**
    - **Project Manager:** Madusha Lakshan
    - **Developers:**
        - Shanuka Dilshan (Front-end and Back-end)
        - Nipuna Lakmal (Front-end and Back-end)
        - Madupa Nimsara (Front-end and Back-end)
    - **DevOps Engineer:** Samoda Ediriweera
    - **Consultant:** Mr. Bathiya Senawirathna

All individuals involved in this project are fellow university students, and this project is part of a university assignment, hence no one is receiving any payment.

- **Development Costs**
    - Software Licenses/Tools and Utilities: Integrated Development Environment (IDE): Visual Studio Code (Free)
    - MERN Technologies: MongoDB, Express.js, React.js, Node.js (All Open Source)

- **Infrastructure Costs**
    - Hardware: Personal laptops and computers (owned by team members)
    - Cloud Services: Hosting the website: Estimated at $50
    - Publishing the mobile app on Google Play Store: One-time fee of $25

2.2.2 Cost-Benefit Analysis

Conducting a cost-benefit analysis (CBA) for our software project involves comparing the total expected costs against the total expected benefits to evaluate the financial viability of the project. Here's a step-by-step guide on our CBA:

- **Identify Costs:**
    - **Development Costs:**
        - All development is done using free tools and IDEs, so there are no additional costs for software licenses.
    - **Infrastructure Costs:**
        - Hosting the website: $50
        - Cloud Services (AWS): $100
        - Google Play Store publishing fee: $25
    - **Miscellaneous Costs:**
        - Potential additional costs for unexpected requirements or minor expenses: $25

      Total Estimated Costs: $200

- **Identify Benefits:**
  - **Improved Efficiency:** The automated mail tracking system will significantly reduce the time and effort required to track physical mail, leading to better efficiency and productivity within the faculty.
  - **Enhanced Accuracy:** By using QR codes and a digital system, the accuracy of mail tracking will improve, reducing errors and misplacements.
  - **Educational Value:** The project provides practical experience in using modern web and mobile development technologies, valuable for the academic and professional growth of all team members.

- **Compare Costs and Benefits:**
  - While the total estimated cost is $100, the non-monetary benefits in terms of improved efficiency, accuracy, and educational value are substantial. The project is highly viable given its minimal costs and significant expected benefits.

2.3 Innovation and Competitiveness

2.3.1 Innovation

Our Faculty of Technology Mail Tracking System introduces several innovative elements that distinguish it from traditional mail tracking methods:

- **QR Code Integration:**
  - By embedding QR codes in physical mail, we streamline the tracking process, making it quick and error-free. Each QR code is unique and can be easily scanned to update the mail's status.

- **Real-time Tracking:**
  - The system allows for real-time tracking of mail as it moves through various departments. This ensures transparency and accountability, significantly reducing the chances of misplacement or delay.

- **Seamless Mobile Integration:**
  - Utilizing Flutter for mobile app development, our system enables users to conveniently scan QR codes and update the mail's status on the go. This mobile integration is particularly useful for staff who are frequently on the move.

- **Automated Notifications:**
  - The system sends automated notifications to relevant parties at each stage of the mail's journey. This ensures that everyone involved is aware of the mail's current status, further enhancing communication and efficiency.

- **User-friendly Interface:**
  - Both the web and mobile applications are designed with a focus on user experience. The intuitive interfaces make it easy for users of all technical backgrounds to navigate and utilize the system effectively.

2.3.2 Competitiveness

Our mail tracking system offers several competitive advantages over existing solutions:

- **Cost-effectiveness:**
  - By leveraging open-source technologies and free tools like Visual Studio Code and the MERN stack, our system is highly cost-effective. This makes it an attractive option for educational institutions and organizations with limited budgets.

- **Scalability:**
  - The use of Docker for deployment ensures that our system can easily scale to accommodate increased usage or expansion to other faculties or departments. This scalability makes it a sustainable solution for long-term use.

- **Customization:**
  - The system is highly customizable, allowing it to be tailored to the specific needs and workflows of different departments or institutions. This flexibility sets it apart from one-size-fits-all solutions.

- **Enhanced Security:**
  - The system incorporates robust security measures, including secure authentication and encrypted communication, to protect sensitive information. This focus on security enhances trust and reliability.
- **Educational Impact:**
  - Beyond its practical applications, the project provides invaluable learning opportunities for the students involved. By working with modern technologies and methodologies, the team gains hands-on experience that is highly relevant in the current job market.

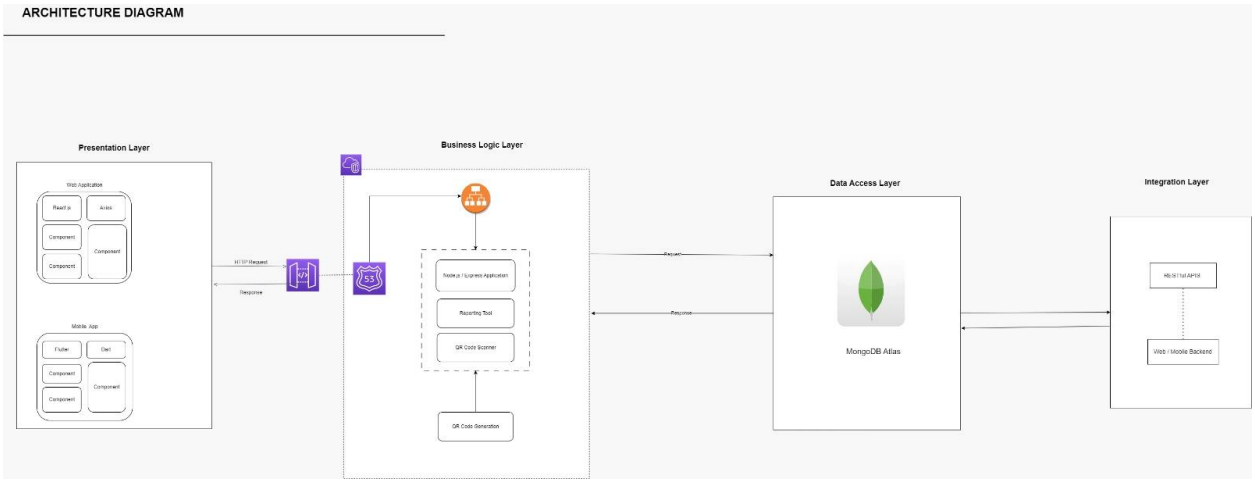
3. System Design

3.1 Outputs of the System Design Phase

3.1.1 Design Documents

3.1.1.1 High-Level Design (HLD)

- System Architect Design Architectural Pattern:
    - Type: Layered Architecture



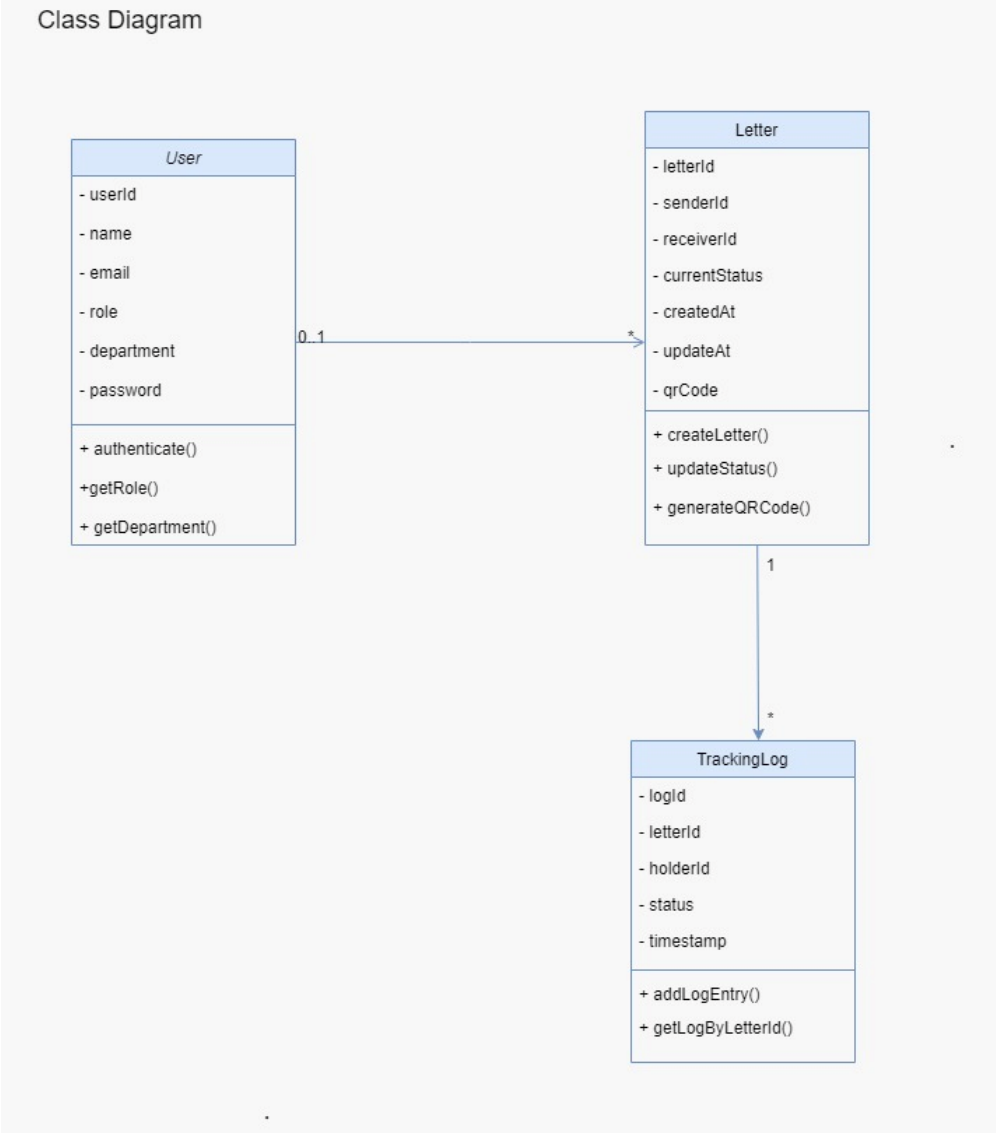
.

- Main Components:
    - Web Application (MERN Stack):
        - Frontend: Developed using React.js
        - Backend: Developed using Node.js and Express.js
        - Database: MongoDB
        - QR Code Generation: External API for generating QR codes
        - Reporting Tool: Crystal Reports or an alternative for generating PDF letterheads
- Mobile Application (Flutter):
    - Login Module: Authentication for users (lecturers, deans, department heads)
    - QR Code Scanner: Scan QR codes to verify and update mail status
    - Integration Module: Communication with the backend API
- Database:
    - MongoDB Atlas: Cloud database for storing user data, mail tracking details, and QR codes
- Deployment:
    - Docker: Containerization of the web and mobile applications for consistent deployment
    - Cloud Platform: Deployment on a cloud service (e.g., AWS, Google Cloud)
- Component Interactions
    - User Interactions:

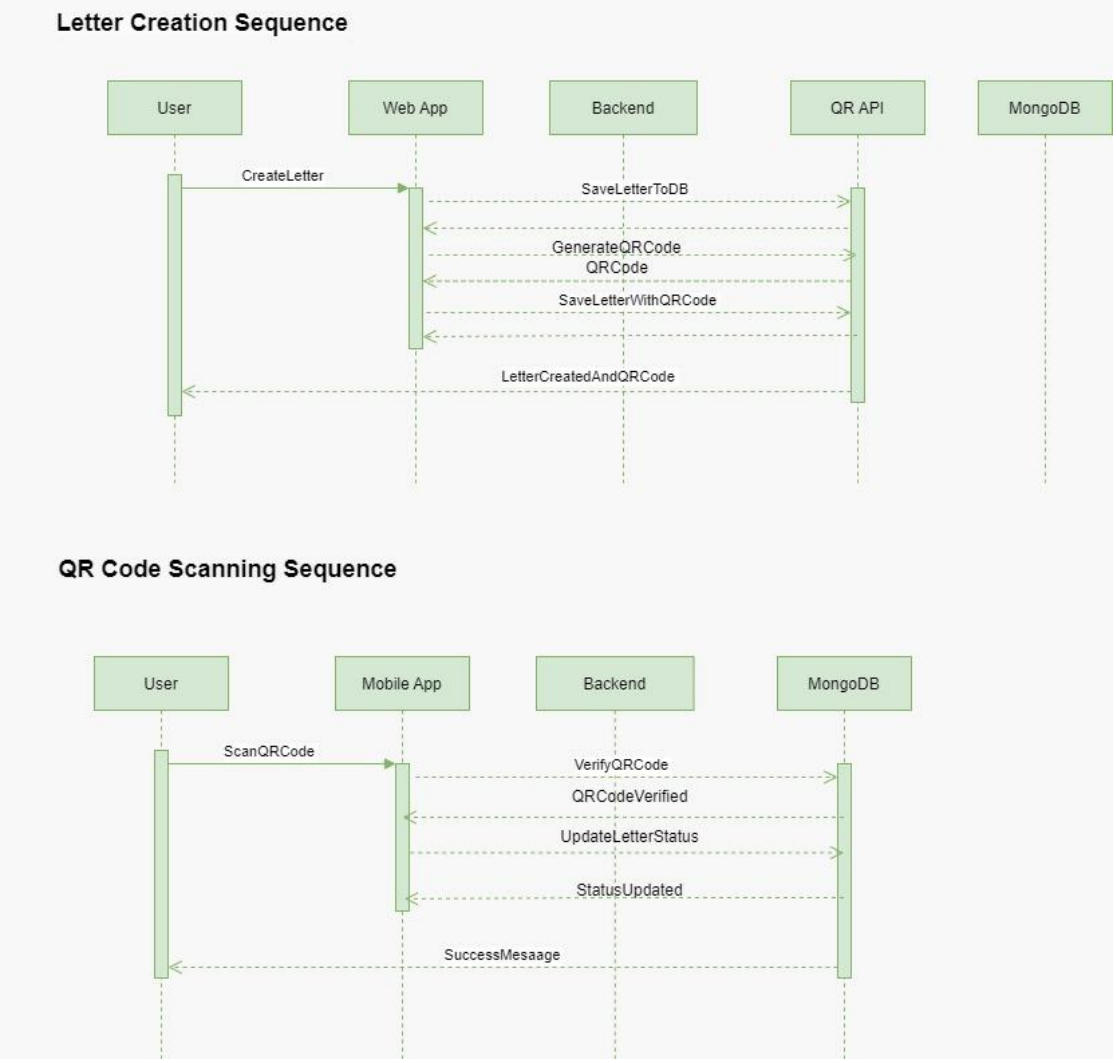
- Users interact with the web and mobile applications to create, manage, and track letters.
- API Requests:
  - Frontend and mobile applications communicate with the backend via RESTful APIs for data retrieval and updates.
- Database Operations:
  - The backend interacts with MongoDB to store and retrieve data.

3.1.1.2 Low-Level Design (LLD)

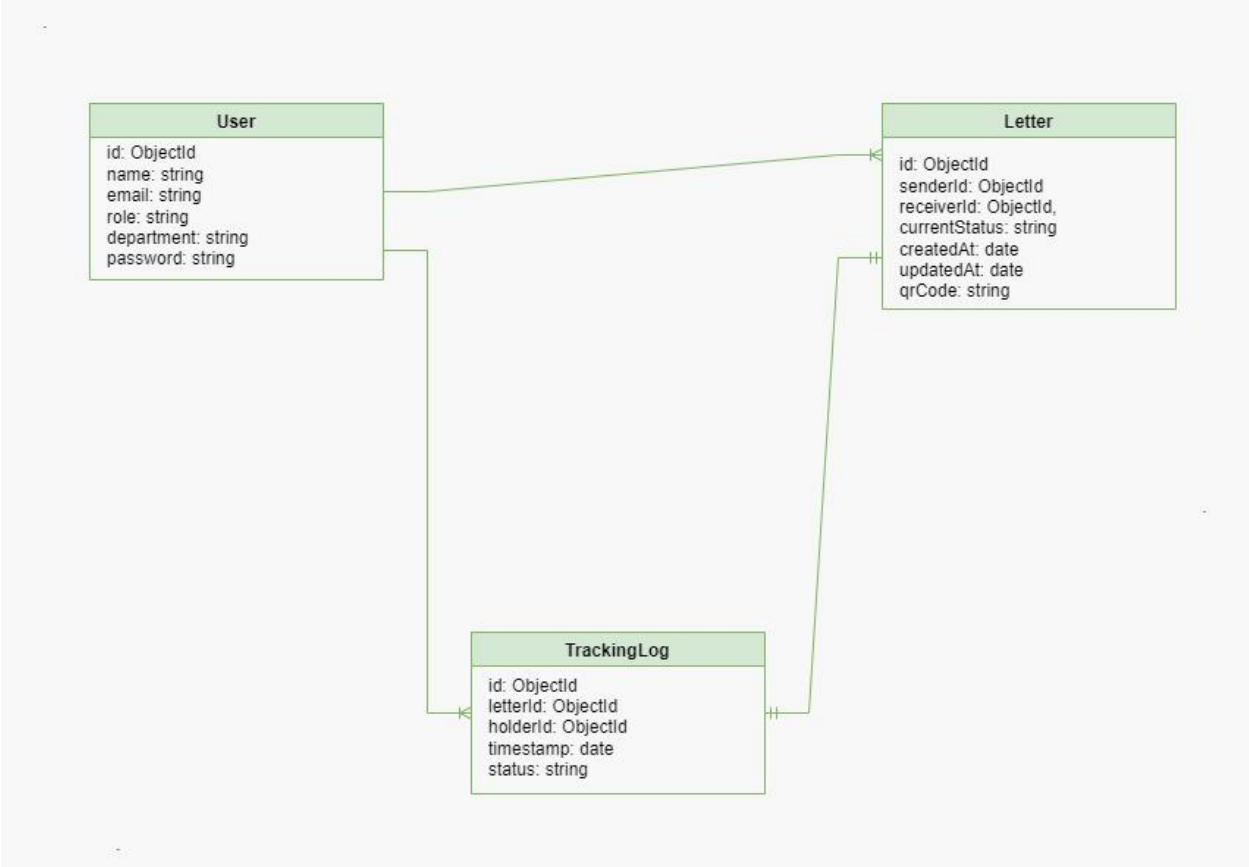
- Class Diagrams



- Sequence Diagrams



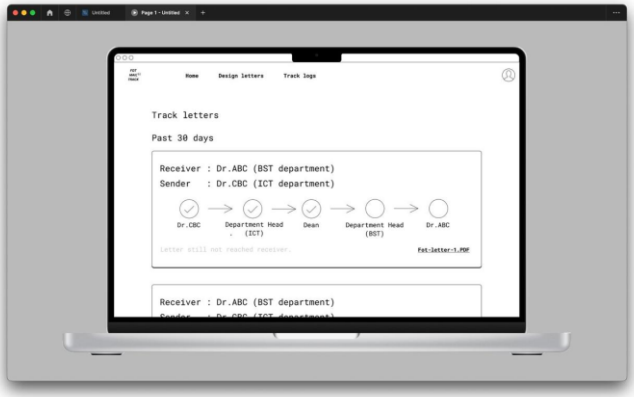
- Database Schema



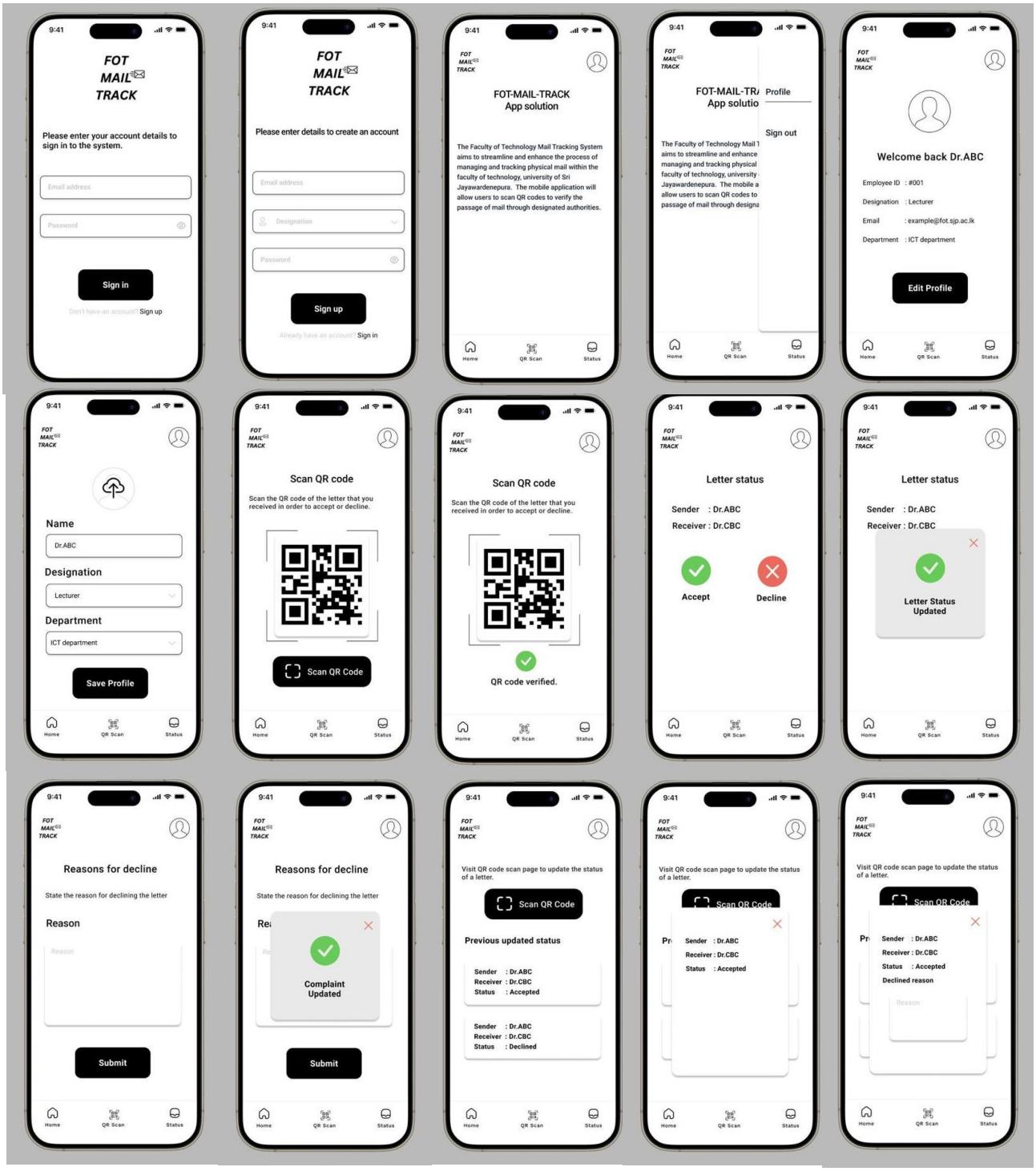
3.1.2 Prototypes and Mockup

- **Low fidelity FMT app**

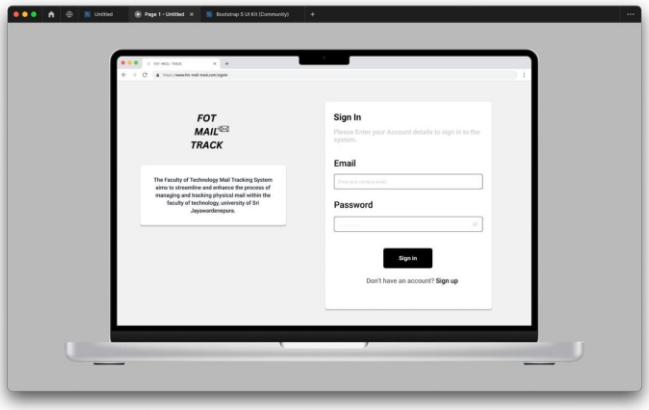
- **Low fidelity webapp**

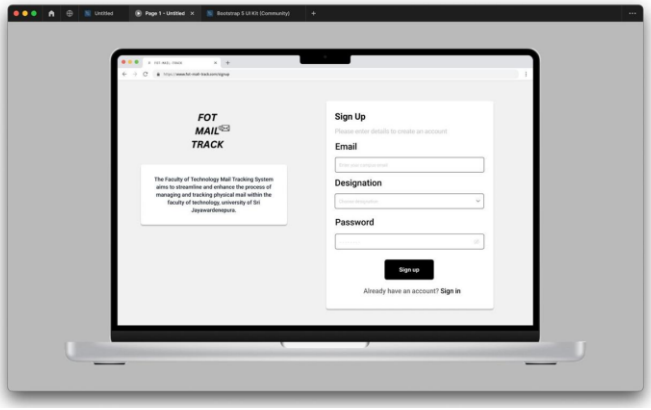


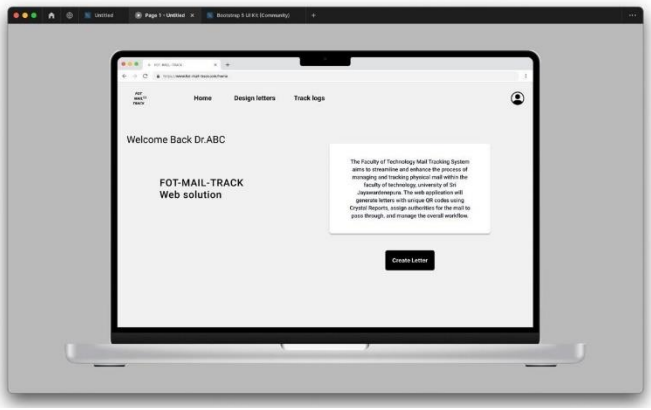
- **High fidelity App**

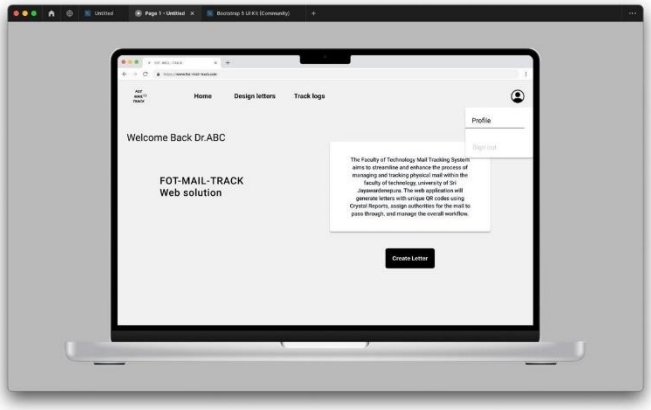


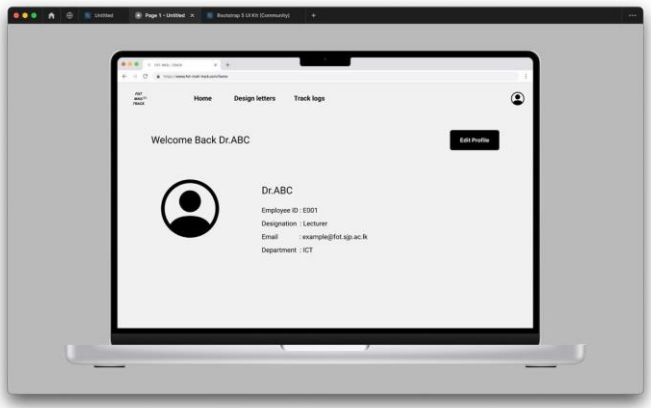
- **High fidelity web app**

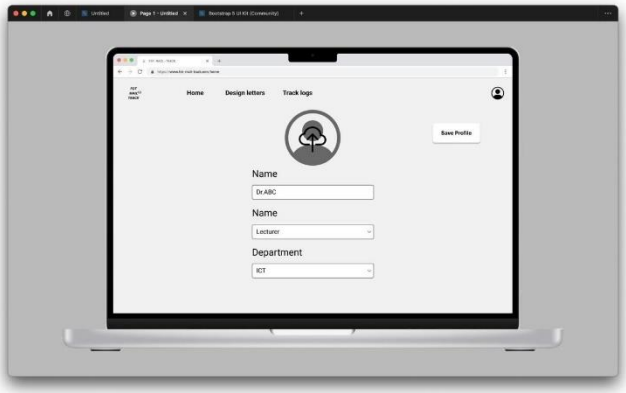


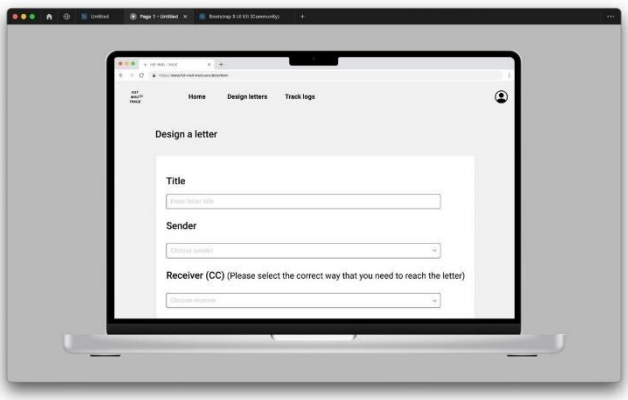








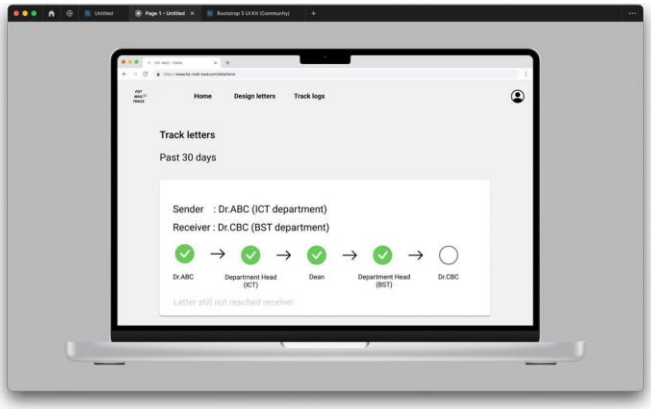


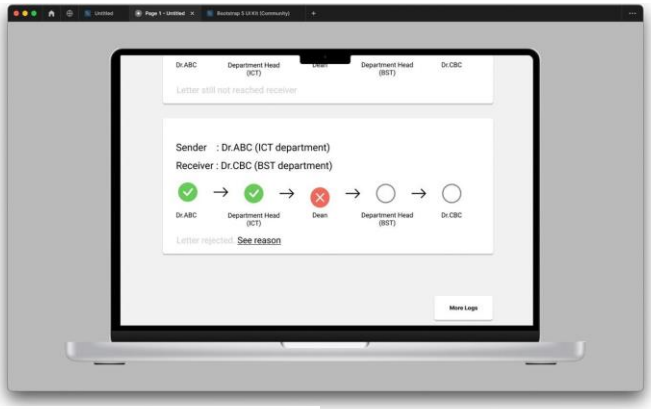


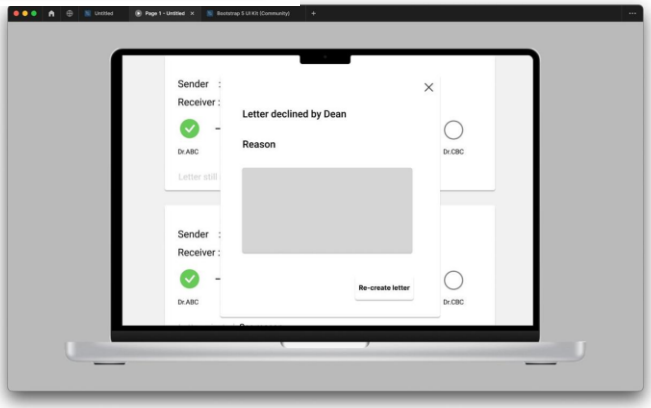
All


Sender
Dr.ABC
Receiver (CC) (Please select the correct way that you need to reach the letter)
Department Head (ICT)
Dean
Department Head (BST)
Dr.CBC
Content
Lorem ipsum dolor sit amet

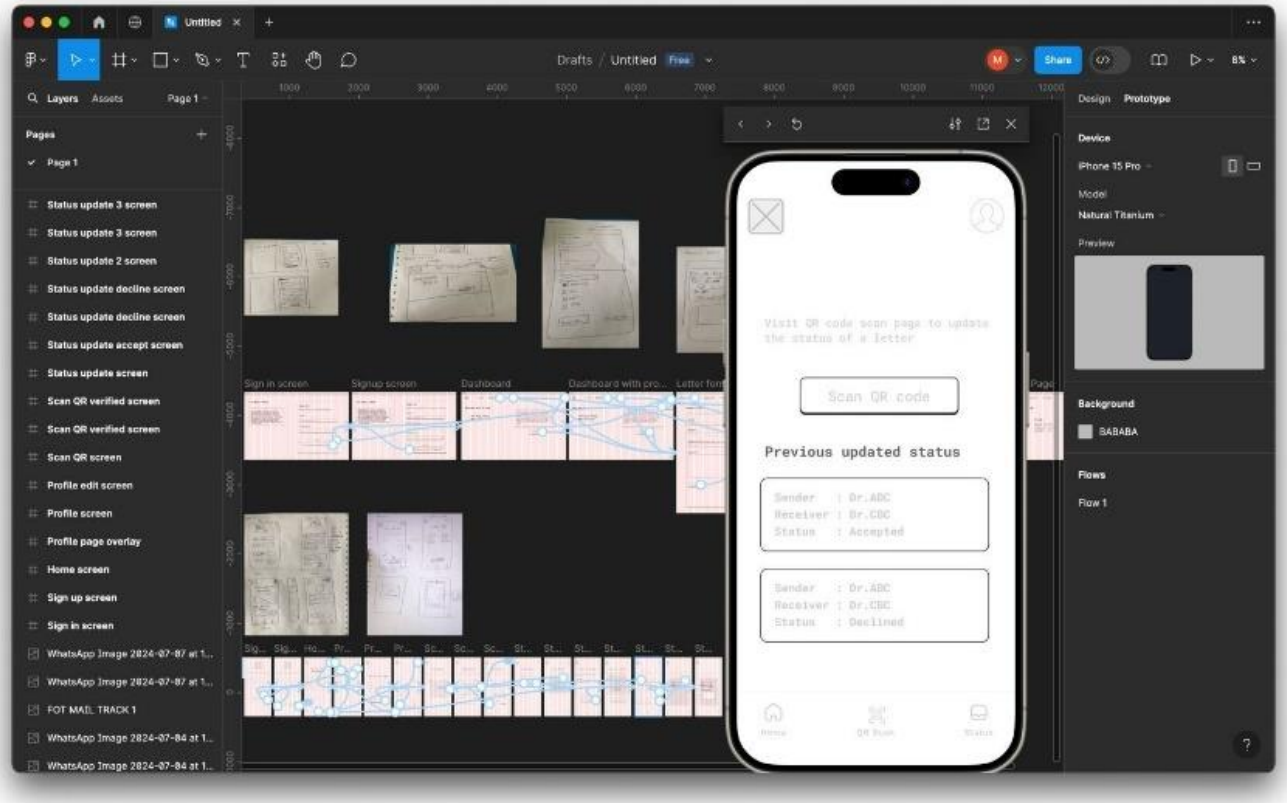

Dr.ABC
Receiver (CC) (Please select the correct way that you need to reach the letter)
Content
Your letter has generated.
View generated letter


Home
Design letters
Track logs
Track letters
Past 30 days
Sender : Dr.ABC (ICT department)
Receiver : Dr.CBC (BST department)
Dr.ABC
Department Head (ICT)
Dean
Department Head (BST)
Dr.CBC
Letter still not reached receiver


Sender : Dr.ABC (ICT department)
Receiver : Dr.CBC (BST department)
Letter rejected. See reason
More Logs


Letter declined by Dean
Reason
Re-create letter

- **Prototype**



Prototype Link

https://www.figma.com/design/0gJPV4JUSSgLXEnHwgQa1y/Fot-Mail-Track?node-id=0-1&t=F7IARXTehIWP2x0B-1

3.1.3 Security Plan

- **Overview**

This security plan outlines the detailed security measures, including authentication, authorization, and encryption strategies, to protect the Faculty of Technology Mail Tracking System. The goal is to ensure the confidentiality, integrity, and availability of the system and its data.

- **Authentication**

Objective: Verify the identity of users to ensure that only authorized individuals can access the system.

Strategies:

- Username and Password:
  - Users must provide a unique username and password to access the system.
  - Implement strong password policies (e.g., minimum length, complexity requirements).
  - Use bcrypt or a similar library to hash passwords before storing them in the database.

- **Authorization**

Objective: Control access to resources and actions based on the user's role and permissions.

Strategies:

- Role-Based Access Control (RBAC):
  - Define user roles (e.g., Lecturer, Head of Department, Dean).
  - Assign permissions to each role (e.g., create letter, approve letter, scan QR code).
  - Implement middleware in the backend to check user roles and permissions before allowing access to protected routes.
- Access Control Lists (ACLs):
  - Define specific permissions for users or groups for fine-grained control.
  - Store ACLs in the database and enforce them at the application level.

- **Encryption**

  Objective: Protect data at rest and in transit to prevent unauthorized access and tampering.

  Strategies:

  - Data Encryption:
    - Use MongoDB's built-in encryption features to encrypt data at rest.
    - Encrypt sensitive fields (e.g., passwords, personal information) before storing them in the database.
  - Transport Layer Security (TLS):
    - Use HTTPS to encrypt data in transit between clients (web and mobile applications) and the backend server.
    - Obtain and install SSL/TLS certificates from a trusted certificate authority (CA).
  - API Key and Secret Management:
    - Use environment variables or a secrets management service (e.g., AWS Secrets Manager) to securely store API keys and other sensitive credentials.
    - Avoid hardcoding sensitive information in the source code.

- **Data Protection**

  Objective: Ensure the confidentiality, integrity, and availability of data.

  Strategies:

  - Database Security:
    - Use MongoDB Atlas IP whitelist to restrict access to specific IP addresses.
    - Regularly update and patch the database system to protect against vulnerabilities.
    - Implement database backups and disaster recovery plans.
  - Data Integrity:
    - Use checksums or hashes to verify the integrity of data during transmission and storage.
    - Implement version control for critical data to track changes and recover previous versions if necessary.
  - Data Availability:
    - Use cloud infrastructure (e.g., AWS, Google Cloud) to ensure high availability and scalability.
    - Implement load balancing and failover mechanisms to handle traffic spikes and server failures.

3.1.4 Strategies for Performance Optimization and Scalability

- **Performance Optimization**

  Objective: Enhance the performance of the Faculty of Technology Mail Tracking System to ensure fast response times and efficient resource usage.

  Strategies:

  - Efficient Database Queries:
    - Optimize MongoDB queries by creating appropriate indexes.
    - Use MongoDB's aggregation framework for complex queries to reduce the number of database calls.
    - Avoid using expensive operations (e.g., $regex, $nin) in queries where possible.
  - Caching:
    - Implement caching at various levels (e.g., application-level, database-level) to reduce the load on the database and improve response times.
    - Use in-memory data stores like Redis or Memcached to cache frequently accessed data.
  - Code Optimization:
    - Profile and identify bottlenecks in the code using tools like Node.js Profiler or Chrome DevTools.
    - Optimize the code by refactoring inefficient algorithms and removing unnecessary computations.
    - Use asynchronous programming (e.g., Promises, async/await) to handle I/O operations efficiently.
  - Content Delivery Network (CDN):
    - Use a CDN to deliver static assets (e.g., images, CSS, JavaScript) to reduce the load on the server and improve content delivery speed.
    - Ensure assets are properly versioned and cached by the CDN.
  - Lazy Loading:
    - Implement lazy loading for non-essential resources to improve initial load times.
    - Use techniques like code splitting and dynamic imports to load JavaScript modules only when needed.

- **Scalability**

  Objective: Ensure the system can handle increased loads and accommodate growth in users and data without degradation in performance.

  Strategies:

  - Horizontal Scaling:
    - Design the system to support horizontal scaling by adding more instances of servers or services.
    - Use load balancers (e.g., Nginx, HAProxy) to distribute incoming traffic across multiple servers evenly.
  - Microservices Architecture:
    - Break down the monolithic application into smaller, independent microservices.
    - Each microservice should handle a specific business function and can be scaled independently.
    - Use containers (e.g., Docker) and orchestration tools (e.g., Kubernetes) to manage and deploy microservices efficiently.
  - Database Sharding:
    - Implement sharding in MongoDB to distribute data across multiple servers or clusters.
    - This helps in handling large datasets and high throughput by distributing the load.
  - Auto-scaling:
    - Use auto-scaling features provided by cloud providers (e.g., AWS Auto Scaling, Google Cloud AutoScaler) to automatically adjust the number of instances based on demand.
    - Set up appropriate scaling policies to ensure resources are scaled up during peak times and scaled down during off-peak times.
  - Message Queues:
    - Use message queues (e.g., RabbitMQ, Apache Kafka) to decouple services and manage high volumes of asynchronous tasks.
    - This helps in smoothing out spikes in traffic and ensuring tasks are processed reliably.
  - Database Connection Pooling:
    - Implement database connection pooling to reuse existing connections and reduce the overhead of creating new connections.
    - Use connection pool libraries (e.g., mongoose, pg-pool for PostgreSQL) to manage database connections efficiently.
  - Monitoring and Alerts:
    - Implement comprehensive monitoring and alerting using tools like Prometheus, Grafana, or ELK Stack (Elasticsearch, Logstash, Kibana).
    - Monitor key performance metrics (e.g., response times, CPU usage, memory usage) and set up alerts for anomalies.

- Continuous Integration/Continuous Deployment (CI/CD):
  - Set up a CI/CD pipeline to automate the process of testing, building, and deploying the application.
  - Use tools like Jenkins, GitLab CI, or CircleCI to ensure that code changes are deployed quickly and reliably.
  - By implementing these strategies, the Faculty of Technology Mail Tracking System will be optimized for performance and scalability, ensuring a responsive and reliable user experience even as the system grows.

4. System Development

4.1 Project management Approach

- **Testing and Deployment Strategies (Tools and Technologies)**

Objective: To ensure that the Faculty of Technology Mail Tracking System is thoroughly tested for functionality, performance, and security, and that it is deployed efficiently and reliably.

Testing Strategies:

- Unit Testing:
  - Tools: Jest, Mocha, Chai
  - Description: Test individual components or functions to ensure they work as expected.
- Integration Testing:
  - Tools: Postman , Insomnia
  - Description: Test the integration between different components of the system to ensure they work together seamlessly.
- End-to-End (E2E) Testing:
  - Tools: Cypress, Selenium, Appium (for Flutter app)
  - Description: Test the complete workflow of the application from start to finish to ensure the system behaves as expected in a real-world scenario
- Performance Testing:
  - Tools: Apache JMeter, Gatling
  - Description: Test the system's performance under various load conditions to ensure it can handle high traffic and data volumes.
- Security Testing:
  - Tools: OWASP ZAP, Burp Suite
  - Description: Identify and fix security vulnerabilities in the system to ensure it is secure from attacks.

Deployment Strategies:

- Containerization:
  - Tools: Docker
  - Description: Package the application and its dependencies into containers for consistent and reliable deployment across different environments.
- Orchestration:
  - Tools: Kubernetes
  - Description: Manage and deploy containerized applications at scale, ensuring high availability and automatic scaling.
- Continuous Integration/Continuous Deployment (CI/CD):
  - Tools: Jenkins, GitHub action
  - Description: Automate the process of testing, building, and deploying the application to ensure quick and reliable updates.
- Cloud Deployment:
  - Platforms: AWS
  - Description: Deploy the application to a cloud platform to ensure it is accessible, scalable, and reliable.
- Monitoring and Logging:
  - Tools: Prometheus, Grafana
  - Description: Monitor the application's performance and logs to identify and fix issues proactively.

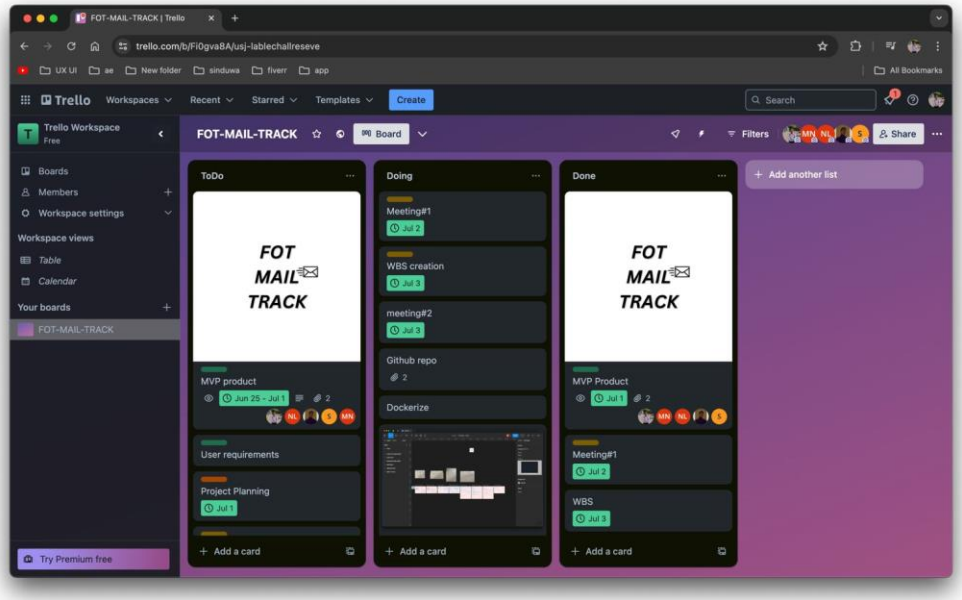
Project Management Approach Methodology

- Agile Scrum:

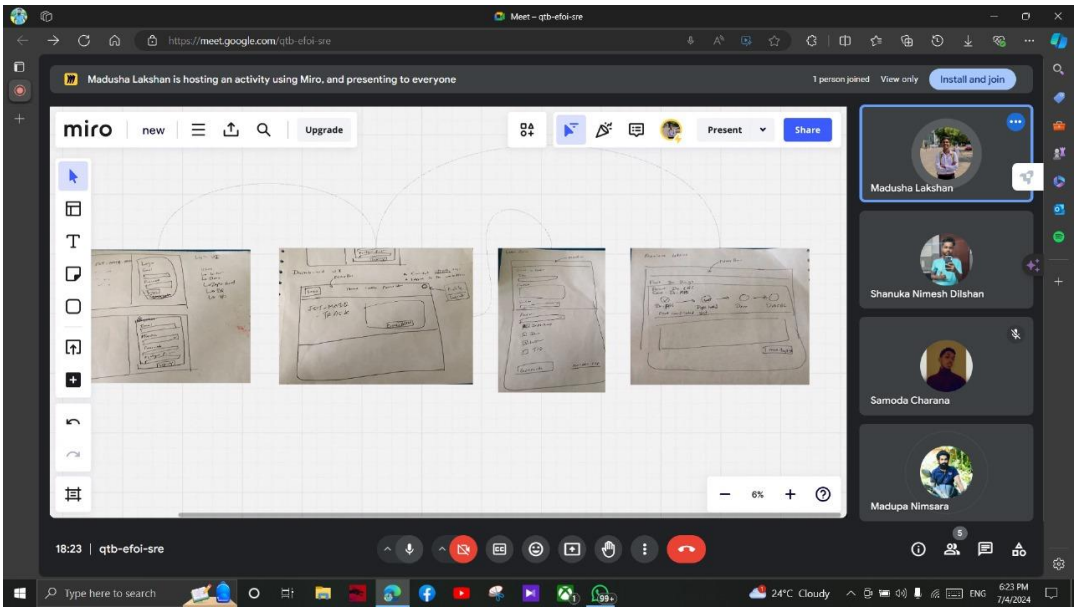
  Description: Agile Scrum is an iterative and incremental project management methodology that promotes adaptive planning, evolutionary development, early delivery, and continuous improvement. It encourages flexible responses to change.

- Key Components:
  - Sprints: Short, time-boxed iterations (usually 2-4 weeks) during which specific work is completed and made ready for review.
- Scrum Meetings:
  - Daily Stand-up: Daily team meeting to discuss progress, upcoming work, and any impediments.
  - Sprint Planning: Meeting to plan the work for the upcoming sprint.
  - Sprint Review: Meeting to review the work completed in the sprint and gather feedback.
  - Sprint Retrospective: Meeting to reflect on the sprint and identify ways to improve.

- Advantages:
    - Increased flexibility and adaptability to changes.
    - Continuous feedback and improvement.
    - Enhanced collaboration and communication within the team.
    - Project Management Tools and Techniques
- Project Management Tools:
    - Trello: A visual project management tool that uses boards, lists, and cards to organize tasks and projects.



- Collaboration Tools:
    - Google Meet: Conduct daily stand-ups, sprint reviews, and retrospectives.



- Version Control Systems:
  - Git: A distributed version control system for tracking changes in source code.
  - GitHub: A web-based platform for version control and collaboration using Git.
- Documentation Tools:
  - Google Docs: An online word processor for creating and sharing documents in real-time.
- Time Management Tools:
  - Toggl: A time tracking tool to monitor the time spent on different tasks and projects.

Techniques:

- Kanban:

  Description: A visual project management technique that uses boards and cards to represent work items and their status.

  Application: Helps to visualize work, limit work in progress, and optimize flow.

- Gantt Charts:

  Description: A bar chart that represents a project schedule, showing the start and end dates of tasks.

  Application: Useful for planning and tracking project timelines and dependencies.

- Burndown Charts:

  Description: A graphical representation of work left to do versus time.

  Application: Helps to track progress in a sprint and predict project completion.

- Backlog Grooming:

  Description: Regular review and prioritization of the project backlog to ensure it is up-to-date and reflects the current project priorities.

  Application: Ensures that the team is always working on the most important tasks.

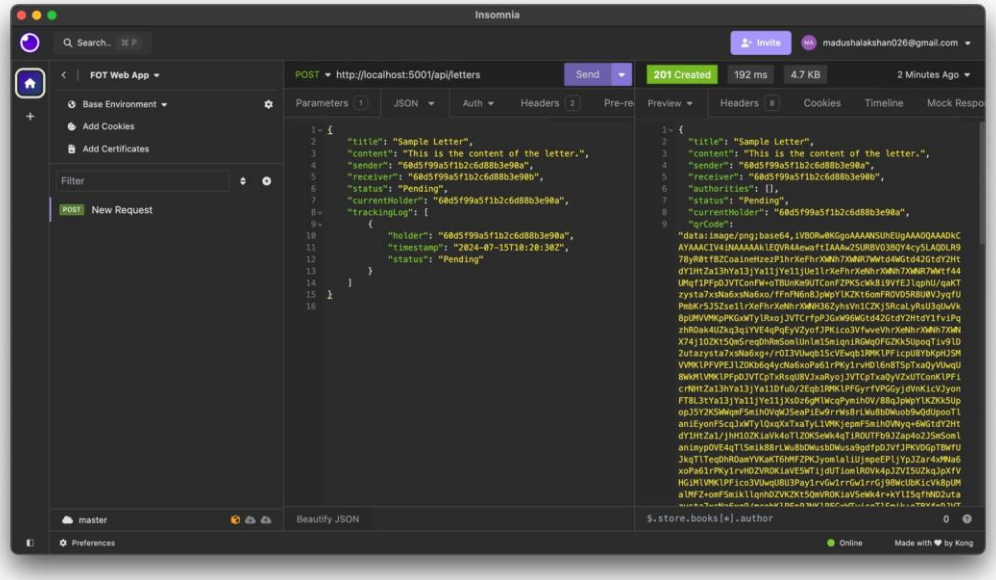
By employing these testing and deployment strategies, project management methodologies, and tools, the development and management of the Faculty of Technology Mail Tracking System will be efficient, collaborative, and aligned with best practices.

4.2 Quality Assurance

- **Quality Control Measures**
  - Code Reviews:
    - Peer Reviews: Conduct regular peer reviews to ensure code quality, adherence to coding standards, and best practices.
    - Automated Code Analysis: Use tools like ESLint for JavaScript and Dart Analysis for Flutter to automatically check for code quality issues.
  - Continuous Integration (CI):
    - Automated Builds and Tests: Integrate automated builds and testing into the CI pipeline using tools like GitHub Actions or Jenkins.
    - Immediate Feedback: Provide immediate feedback to developers on code changes to quickly identify and resolve issues.
  - Static Code Analysis:
    - Tools: Use SonarQube for static code analysis to detect code smells, bugs, and security vulnerabilities early in the development cycle.
    - Reporting: Generate reports to track code quality metrics and improvements over time.
  - Coding Standards and Guidelines:
    - Documentation: Maintain a comprehensive set of coding standards and guidelines.
    - Training: Conduct regular training sessions for team members to ensure compliance with these standards.
  - Test-Driven Development (TDD):
    - Practice: Adopt TDD practices to write tests before writing the actual code.
    - Benefits: This ensures that the codebase is always covered by tests and helps in identifying issues early.
  - Version Control:
    - Branching Strategy: Implement a branching strategy like Git Flow to manage feature development, bug fixes, and releases.
    - Pull Requests: Use pull requests to facilitate code reviews and discussions before merging changes into the main branch.

Testing Strategies

- Unit Testing:
  - Scope: Test individual components and functions to ensure they work as expected.
  - Tools: Jest for React (frontend), Mocha/Chai for Node.js (backend), and Flutter Test for Flutter (mobile).
- Integration Testing:
  - Scope: Test the interactions between different modules and components to ensure they work together correctly.
  - Tools: Insomnia for API testing, Selenium for end-to-end testing in the web application.



- System Testing:
  - Scope: Validate the complete and integrated software system to ensure it meets the specified requirements.
  - Tools: Cypress for comprehensive end-to-end testing of the web application.
- User Acceptance Testing (UAT):
  - Process: Involve end-users to validate the system's functionality, usability, and performance.
  - Goal: Ensure the system meets user expectations and business requirements.
  - Tools: Use platforms like TestRail to manage UAT processes and gather feedback.

- Performance Testing:
  - Scope: Assess the system's performance under various conditions to ensure it can handle expected loads.
  - Tools: JMeter and Apache Bench to simulate load and measure performance metrics like response time, throughput, and resource utilization.
- Security Testing:
  - Scope: Identify and mitigate security vulnerabilities within the system.
  - Tools: OWASP ZAP for web application security testing, dependency-check tools to identify vulnerabilities in third-party libraries.
- Regression Testing:
  - Scope: Ensure that new changes do not introduce bugs or break existing functionality.
  - Tools: Automated test suites running in CI pipelines to quickly identify regressions.
- Beta Testing:
  - Scope: Release a beta version of the application to a limited audience to gather real-world feedback and identify any issues before the final release.
  - Tools: Beta testing platforms like TestFlight (iOS) and Google Play Beta Testing (Android).

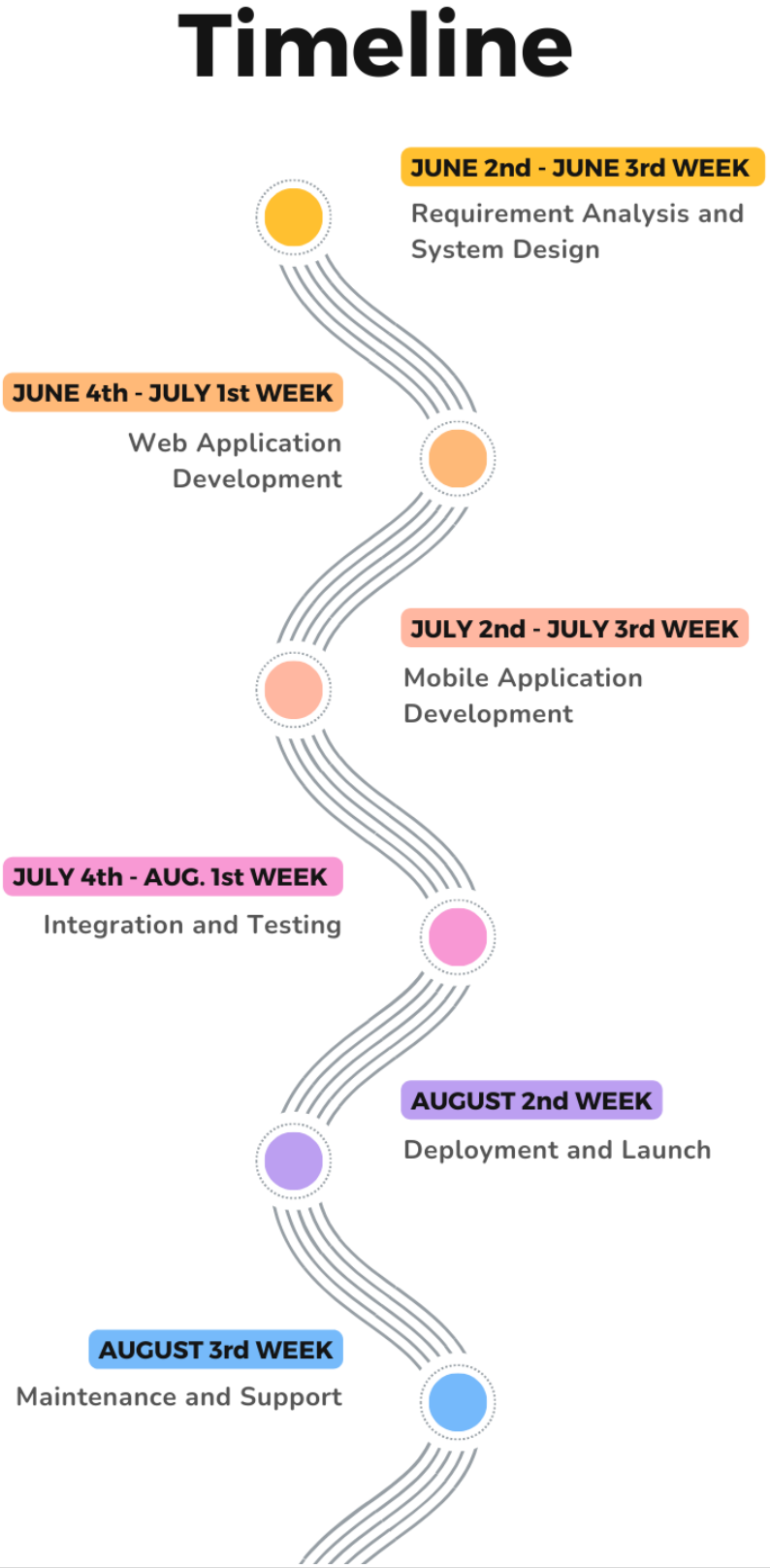
Conclusion

Implementing robust quality control measures and comprehensive testing strategies ensures the delivery of high-quality software. Regular code reviews, automated testing, and adherence to coding standards help maintain code quality. Using a mix of unit, integration, system, and user acceptance testing ensures that the system meets all functional and non-functional requirements. Performance and security testing further ensure that the system is reliable and secure. By following these practices, the Faculty of Technology Mail Tracking System will be built with a focus on quality and reliability.

4.3 Sustainability and Maintenance

- **Sustainability**
  - Modular Architecture:
    - Design for Modularity: Ensure the system is built with a modular architecture to allow for easy updates and maintenance.
    - Microservices: If applicable, consider using microservices architecture for better scalability and maintainability.
  - Documentation:
    - Comprehensive Documentation: Maintain detailed documentation for code, APIs, and system architecture.
    - Update Regularly: Ensure documentation is updated regularly to reflect changes in the system.
  - Version Control:
    - Use Git: Implement a robust version control system using Git to track changes and manage different versions of the codebase.
    - Branching Strategy: Use a branching strategy like Git Flow to manage feature development, bug fixes, and releases.
  - Automated Testing:
    - Continuous Testing: Integrate automated testing into the CI/CD pipeline to ensure continuous testing of the system.
    - Test Coverage: Aim for high test coverage to ensure the reliability and stability of the system.
- **Maintenance**
  - Regular Updates:
    - Software Updates: Keep all software components, libraries, and dependencies updated to the latest versions to ensure security and functionality.
    - Security Patches: Apply security patches promptly to protect the system from vulnerabilities.
  - Monitoring and Logging:
    - Real-time Monitoring: Implement real-time monitoring using tools like Prometheus and Grafana to track system performance and identify issues.
    - Logging: Use centralized logging solutions like ELK stack (Elasticsearch, Logstash, Kibana) to collect and analyze logs for troubleshooting.

5. Project Milestones and Timeline



Deliverables

Functional Web Application

A web app with user authentication, letter generation

Functional Mobile Application

A mobile app with login, QR code scanning, and verification features.

Documentation

Comprehensive documentation covering system architecture, installation, and user guides.

Deployment

Dockerized applications deployed on a cloud platform.