2024

PROJECT PROPOSAL

DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

FACULTY OF TECHNOLOGY

Project Title: Campus Utilization Monitor

This project, "Campus Utilization Monitor," addresses the critical need for efficient use of university resources by developing a community-driven system. It aims to provide real-time occupancy status updates for campus facilities such as the library, medical center, and canteens. By enhancing accessibility and resource management, the system empowers students and staff to make informed decisions and optimize their time effectively.

Project Personnel

Academic Supervisor

Name	Institution	Contact Details	
		Mobile No.	Email
Ms.Nirasha Kulasooriya	University of Sri Jayewardenepura	071-7102240	nirashakulasooriya@sjp.ac.lk

Co-Supervisor/s

Name	Institution	Contact Details	
		Mobile No.	Email
Ms.Hasani Gamage	University of Sri Jayewardenepura	076-6773128	-

Team Members

Name	Index Number	Specialized Area	Mobile No.	Email
D. N. K. Pathmasiri	ICT/21/892	Multimedia Technology	+94 77 1079 750	ict21892@fot.sjp.ac.lk
K. D. T. Peiris	ICT/21/895	Network Technology	+94 78 7537 325	ict21895@fot.sjp.ac.lk
J. A. D. L. Jayasinghe	ICT/21/859	Network Technology	+94 77 9430 344	ict21859@fot.sjp.ac.lk
K. A. T. R. Karunarathna	ICT/21/868	Network Technology	+94 78 7059 350	ict21868@fot.sjp.ac.lk
R. M. N. P. SUBHASINGHA	ICT/21/929	Software Technology	+94 77 4966 943	ict21929@fot.sjp.ac.lk
N. M. N. L. N. Bandara	ICT/21/813	Software Technology	+94 76 6122 495	ict21813@fot.sjp.ac.lk

Table of Contents

1. Introduction and Background

In a dynamic university environment, efficient use of campus resources is vital for maximizing the benefits they offer to students and staff. Our project, "Community-Driven Solution for Checking Occupy Status of Library, Medical Center, and Canteens" aims to develop an intuitive system that provides real-time information on the occupancy status of these critical campus facilities. By integrating this system, we seek to enhance the accessibility and utilization of campus resources, ensuring that students and staff can make informed decisions and manage their time more effectively.

The resources on the campus have not been properly utilized for maximum utilization. Often, students and staff face challenges such as overcrowded canteens, unavailable library seating, and long waits at the medical center. These issues stem from the lack of real-time information on the occupancy status of these facilities. Our project addresses this gap by developing a community-driven solution that provides up-to-date occupancy statuses for the library, medical center, and canteens.

The primary purpose of this project is to create a user-friendly system that allows students and staff to check the current occupancy status of the library, medical center, and canteens. By implementing a system where the library and medical center utilize Uni ID for tracking, and workers can scan these IDs to update occupancy statuses, we aim to provide accurate and real-time data. This system will help in reducing overcrowding, improving resource management, and enhancing the overall campus experience.

Our solution is highly relevant in the context of modern educational institutions where time management and resource efficiency are crucial. By offering real-time updates on the occupancy of essential campus facilities, we not only enhance the convenience for users but also promote the optimal use of resources. This project stands to benefit a wide range of users, including students who need to plan their study schedules, staff who require quick access to the medical center, and everyone looking for a less crowded dining experience. By making the most out of campus resources, our project aligns with the broader goal of improving campus life and operational efficiency.

2. System Requirements

2.1. Objectives

The objective of the "Campus Utilization Monitor" project is to develop a comprehensive system that provides real-time information on the occupancy status of key campus facilities: the library, medical center, and canteens. This system aims to enhance resource management and improve user experience by enabling students and staff to make informed decisions about when and where to access these facilities based on current occupancy data.

2.2. Scope

- Develop access control mechanisms for the library and medical center using student IDs or record books.
- Implement real-time tracking and reporting of occupancy status for the library, medical center, and canteens.
- Create a web application accessible to students for viewing current and historical traffic data of campus facilities.
- Design admin interfaces for monitoring and managing access logs, traffic data, and status reports.
- Ensure the system is user-friendly, performs well under multiple concurrent users, maintains high security standards, operates reliably during peak hours, and is compatible across various devices and browsers.

2.3. Deliverables

- A fully functional web application for students to check occupancy status in real-time and view historical data.
- Access control mechanisms integrated with student IDs or record books for the library and medical center.
- Admin interfaces with capabilities to monitor and manage access logs, traffic data, and status reports.
- Comprehensive documentation covering system architecture, user manuals, and maintenance guides.
- Training materials for administrators and support staff to effectively operate and maintain the system.

2.4. Budget and Cost Estimates

2.4.1. Detailed budget breakdown

Project Team and Roles

- **Project Managers:** D. N. K. Pathmasiri, K.D.T. Peiris
- **Developers**:
    - **Front-end Developers:** K.D.T. Peiris, J.A.D.L. Jayasinghe, K.A.T.R. Karunarathna
    - **Back-end Developers:** R. M. N. P. Subhasingha, N.M.N.L.N. Bandara
    - **Full-stack Developer:** N.M.N.L.N. Bandara
- **UX/UI Designers**: D. N. K. Pathmasiri
- **QA Engineers/Testers**: R. M. N. P. Subhasingha, J. A. D. L. Jayasinghe
- **Business Analysts**: All Members
- **DevOps Engineers**: N.M.N.L.N. Bandara, K.D.T. Peiris

Development Costs

Software Licenses/Tools and Utilities:

- Jira: Free version
- Figma: Free version
- Visual Studio Code: Free version
- MongoDB/Firebase: Free trials

By leveraging free versions and trials of essential tools, we ensure minimal initial software costs while still utilizing powerful development and project management tools.

Infrastructure Costs

Hardware/Cloud Services:

- Microservices with Docker containers: Utilize VMs under pay-as-you-go plans
- Jenkins, SonarQube: Utilize VMs under pay-as-you-go plans
- Cloud Services (AWS and Azure): Utilize free credits provided by university email for Azure

Our infrastructure strategy involves using microservices architecture with Docker containers deployed on virtual machines (VMs) for scalability and flexibility. We will use VMs for CI/CD tools like Jenkins and SonarQube under pay-as-you-go plans to manage costs effectively. Leveraging Azure's free credits available through university email will significantly reduce cloud service expenses.

2.4.1. Cost-Benefit Analysis

As students, we have unique advantages that reduce the costs associated with the "Campus Utilization Monitor" project. These include:

- **Azure Credits:** By utilizing free credits provided by Azure through our university email, we can significantly reduce the expenses related to cloud services, such as virtual machines for hosting microservices, Jenkins, and SonarQube.
- **Free Trials of Software and Services:** We are leveraging free trials and versions of essential development and project management tools, including Jira, Figma, Visual Studio Code, and MongoDB/Firebase. This approach minimizes our development costs while still providing access to powerful tools necessary for the project's success.
- **University Email Benefits:** University email provides us with additional benefits, such as access to educational resources, discounts, and free credits for various services. These resources help us manage and execute the project cost-effectively.

2.5. Student Focused Requirements

2.5.1. Student's Functional Requirements

Scenario 1: Library

1. **Access Control:**
    - **Description:** Students must present their student ID or record book to enter the library. The security officer scans the barcode using a mobile device with a camera. The system logs the student's TE number, entry time, and phone number (if not already saved).
    - **Rationale:** Ensures only authorized students can access the library and helps track library usage.

2. **Exit Control:**
    - **Description:** Students must present their ID/record book upon exiting the library. The security officer scans the barcode again to log the exit time.
    - **Rationale:** Accurately tracks the time students spend in the library.

3. **View Traffic Status:**
    - **Description:** Students can use a web application to view the current traffic inside the library and daily traffic statistics.
    - **Rationale:** Helps students plan their visits to the library based on real-time and historical data.

Scenario 2: Medical Center

1. **Access Control:**
    - **Description:** Students must present their student ID or record book to enter the medical center. The nursing officer scans the barcode using a mobile device. The system logs the student's TE number and entry time.
    - **Rationale:** Ensures only authorized students can access the medical center and helps track usage.

2. **Exit Control:**
    - **Description:** Students must present their ID/record book upon exiting the medical center. The nursing officer scans the barcode again to log the exit time.
    - **Rationale:** Accurately tracks the time students spend in the medical center.

3. **View Traffic Status:**
    - **Description:** Students can use a web application to view the current traffic inside the medical center and daily traffic statistics.
    - **Rationale:** Helps students plan their visits to the medical center based on real-time and historical data.

Scenario 3: Canteen

1. **Status Reporting:**
    - **Description:** Students can report the current status of the canteen (busy or not) using a web application. This data is gathered in real-time and displayed as the number of votes for each status.
    - **Rationale:** Allows students to inform others about the current state of the canteen, aiding in decision-making.

2. **View Traffic Status:**
    - **Description:** Students can use the web application to view real-time votes on whether the canteen is busy or not.
    - **Rationale:** Helps students decide the best time to visit the canteen based on real-time feedback from peers.

2.5.2. Student's Non-Functional Requirements

1. **Usability:**
    - **Description:** The application should be user-friendly, with an intuitive interface that is easy to navigate.
    - **Rationale:** Ensures a positive user experience and encourages regular use.
2. **Performance:**
    - **Description:** The system should handle multiple concurrent users without significant delays.
    - **Rationale:** Provides a smooth and responsive experience for all users.
3. **Security:**
    - **Description:** Student data (e.g., TE number, phone number) must be securely stored and accessed only by authorized personnel.
    - **Rationale:** Protects sensitive information and complies with data privacy regulations.
4. **Reliability:**
    - **Description:** The system should always be available and functional, especially during peak hours.
    - **Rationale:** Ensures consistent and dependable service for all users.
5. **Compatibility:**
    - **Description:** The web application should be compatible with various devices and browsers.
    - **Rationale:** Provides accessibility to a broad range of users with different devices.

2.6. Admin Focused Requirements

2.6.1. Admin's Functional Requirements

Scenario 1: Library

1. **Manage Access Logs:**
    - **Description:** Security officers can scan student IDs to log entry and exit times. Data (TE number, in time, out time, phone number) is sent to the database.
    - **Rationale:** Ensures accurate record-keeping and tracking of student library usage.

2. **Monitor Traffic:**
    - **Description:** Admins can view real-time and historical traffic data to manage library occupancy.
    - **Rationale:** Helps in planning and managing library resources effectively.

Scenario 2: Medical Center

1. **Manage Access Logs:**
    - **Description:** Nursing officers can scan student IDs to log entry and exit times. Data (TE number, in time, out time) is sent to the database.
    - **Rationale:** Ensures accurate record-keeping and tracking of student medical center usage.

2. **Monitor Traffic:**
    - **Description:** Admins can view real-time and historical traffic data to manage medical center occupancy.
    - **Rationale:** Helps in planning and managing medical center resources effectively.

Scenario 3: Canteen

1. **Monitor Status Reports:**
    - **Description:** The system aggregates and displays the canteen's current status based on student inputs.
    - **Rationale:** Allows student to monitor the real-time status of the canteen to manage crowd control.

2.6.2. Admin's Non-Functional Requirements

1. **Usability:**
    - **Description:** The admin interface should be intuitive and easy to use for quick operations.
    - **Rationale:** Ensures efficient management and operation of the system.

2. **Performance:**
    - **Description:** The system should process scans and updates promptly without lag.
    - **Rationale:** Provides a smooth and responsive experience for admins.

3. **Security:**
    - **Description:** Only authorized personnel should have access to the admin functions and data.
    - **Rationale:** Protects sensitive information and ensures compliance with data privacy regulations.

4. **Reliability:**
    - **Description:** The system should be dependable with minimal downtime, especially during busy hours.
    - **Rationale:** Ensures consistent and dependable service for all users.

5. **Scalability:**
    - **Description:** The system should be able to handle an increasing number of users and data entries over time.
    - **Rationale:** Accommodates growth and increased usage without compromising performance.

3. System Design

3.1. System Architecture

The system architecture is like a plan for building the plant disease detection web app. It makes sure the app is well-organized, can grow, and can handle users, data, and different parts working together. The design keeps everything in order and easy to manage. In Our Project adopts a client-server architecture. Boths side encompasses the website, providing passengers with an interactive interface. This one is a community-driven app for checking occupancy status and automating the registry of faculty premises.

Purpose:

The purpose of this project is to develop an online population monitoring system for the technology faculty, specifically targeting the library, medical center, and canteen. The system will allow students to view live, real-time population data for these locations, helping them decide the best time to visit based on current occupancy levels.

Technologies:

- Frontend: React.js for the user interface
- Backend: Node.js with Express.js for server-side logic
- Database: MongoDB for data storage
- Real-time Data Handling: Web Sockets (e.g., Socket.io) for real-time updates
- Authentication: JWT (JSON Web Tokens) for secure user authentication
- Hosting: AWS for cloud deployment

Requirements and Constraints:

- The system should efficiently handle concurrent users.
- It must be scalable to accommodate future growth and additional locations.
- Strong security measures should be implemented to protect user data.
- The user interface should be responsive and user-friendly.
- Real-time population data should be accurate and up to date.

Existing Components or Services:

- An existing user authentication service that uses JWT.
- A third-party service for monitoring and updating population data (if available).

THREE-TIER Architecture

Presentation Tier:

The presentation tier encompasses the user interface (UI) and focuses on the visual aspects of the application. The key components of the presentation tier include:

- User Interface (UI): This part shows information to the user and collects their inputs. The UI will use React.js to make it dynamic and responsive, so it works well on different devices and screen sizes.

Logic Tier:

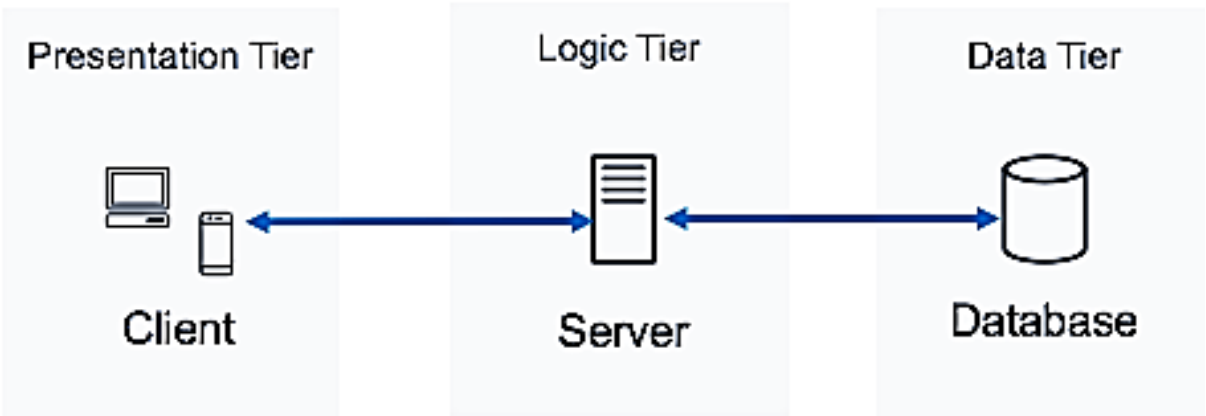
The logic tier handles the application's data interactions and managing Them. Key components in the logic tier include

- Back-End Server
- User Authentication

Data Tier:

The data tier focuses on data storage and management. It ensures and Store that user data, time securely and efficiently. Key components in the data tier include

- Database
- Data Access Layer.

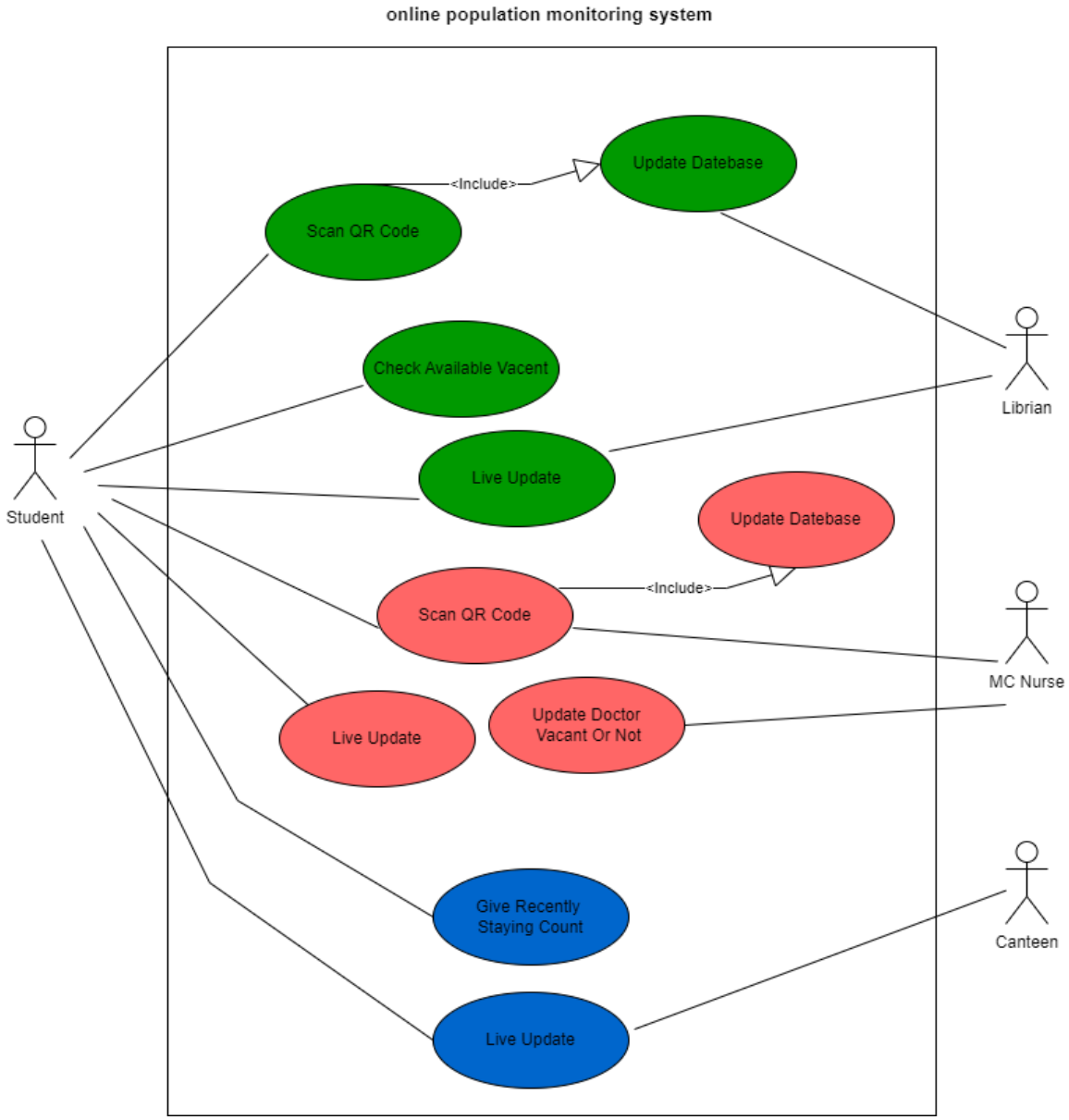


The three-tier architecture keeps things separate, making the application easier to maintain and expand. It allows different parts to be developed and updated independently. For example:

1. Scalability.
2. Modularity.
3. Flexibility.

Use Case

Our online population monitoring system Web Application's Use Case Diagram elegantly illustrates the interactions between different actors and the system. This diagram outlines the key functionalities and user interactions that drive our application's core capabilities.



At the forefront of our application are the users, represented as "User" in the diagram. They engage with the system through essential actions like Scan the QR code, Live Update, and So on These user interactions form the foundation of our user-centric design. The "System" actor embodies the core functionality of our application, orchestrating various processes seamlessly.

Reasons for Using the Use Case Diagram:

1. Functional Flow.
2. Forecast Administrator's and Student Role.
3. Extend and Include Relationships.
4. Clarity of Interactions.

3.2. User Interface

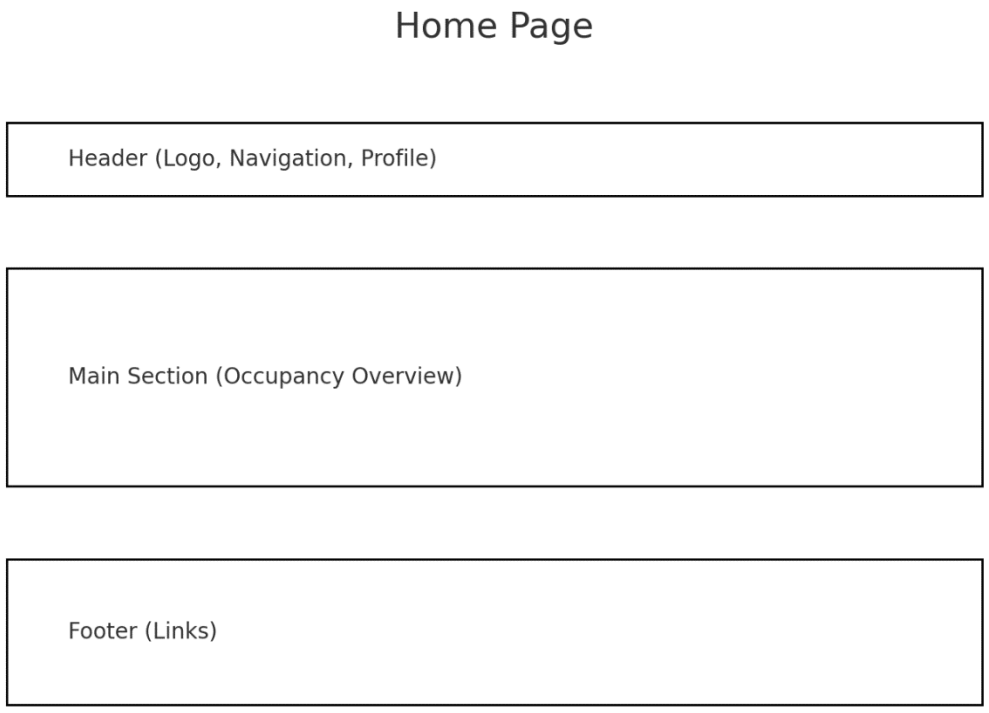
The user interface (UI) of our community-driven web solution is designed with a focus on usability, accessibility, and responsiveness to ensure a seamless experience for all users. The key components of the UI include:

3.2.1. Overview

The UI consists of three main sections corresponding to the three facilities: the library, medical center, and canteens. Each section provides real-time updates on occupancy status and additional relevant information.

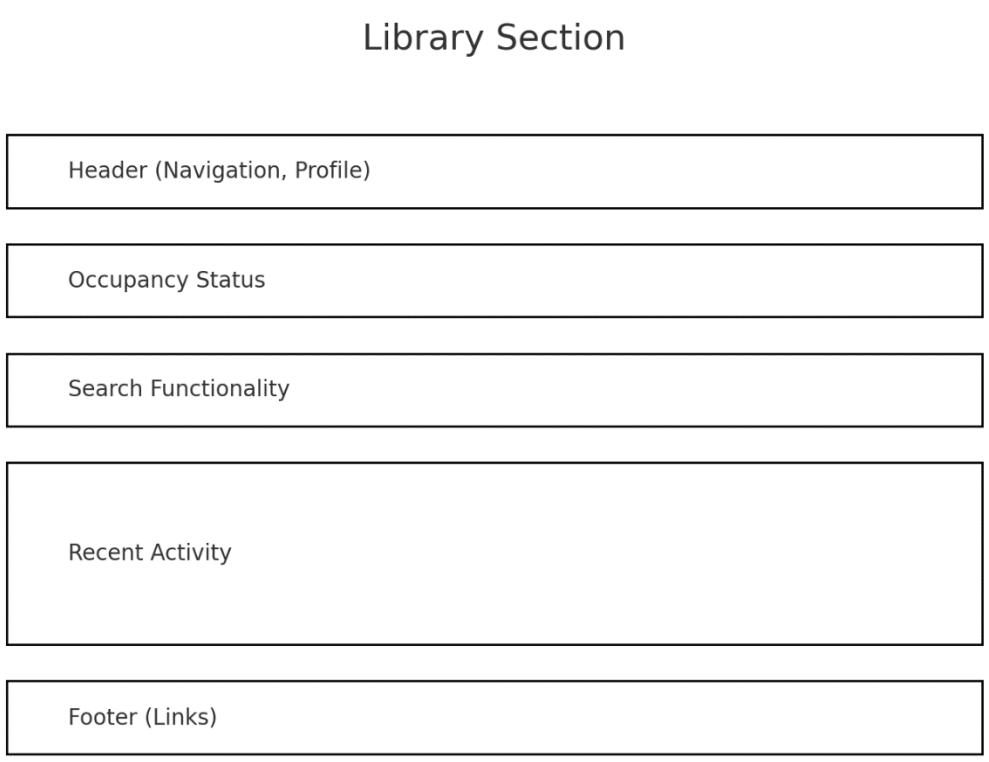
3.2.2. Wireframes

Home Page:



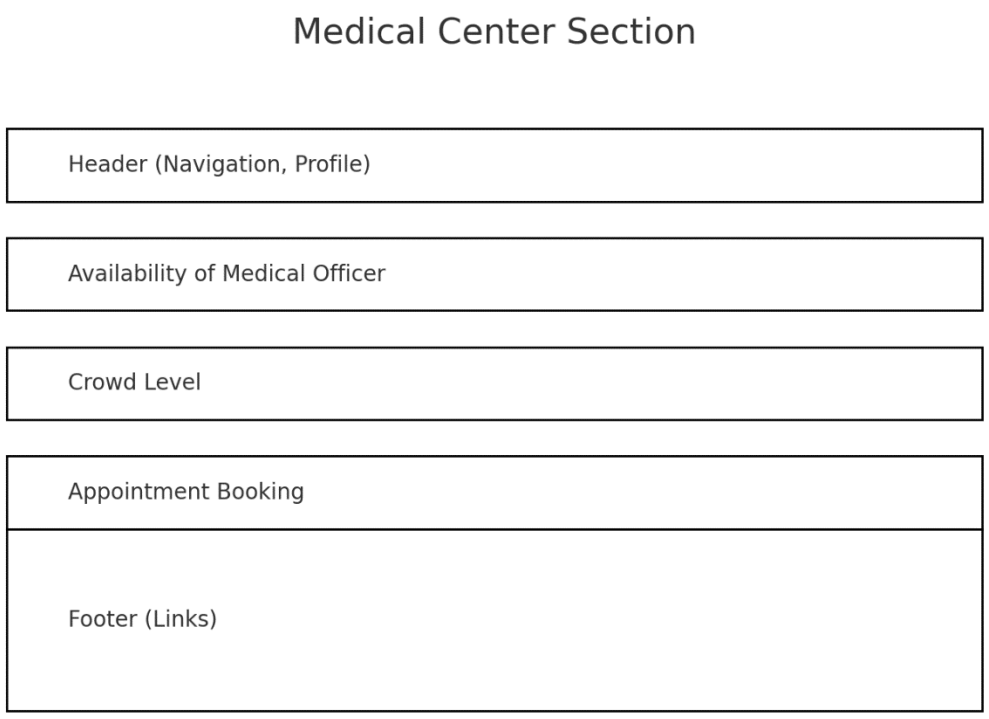
- **Header**: Contains the application logo, navigation links to different sections (Library, Medical Center, Canteens), and a user profile icon.
- **Main Section**: Displays an overview of the current occupancy status of the library, medical center, and canteens, along with quick access buttons to each detailed section.
- **Footer:** Includes links to terms of service, privacy policy, and contact information.

Library Section:



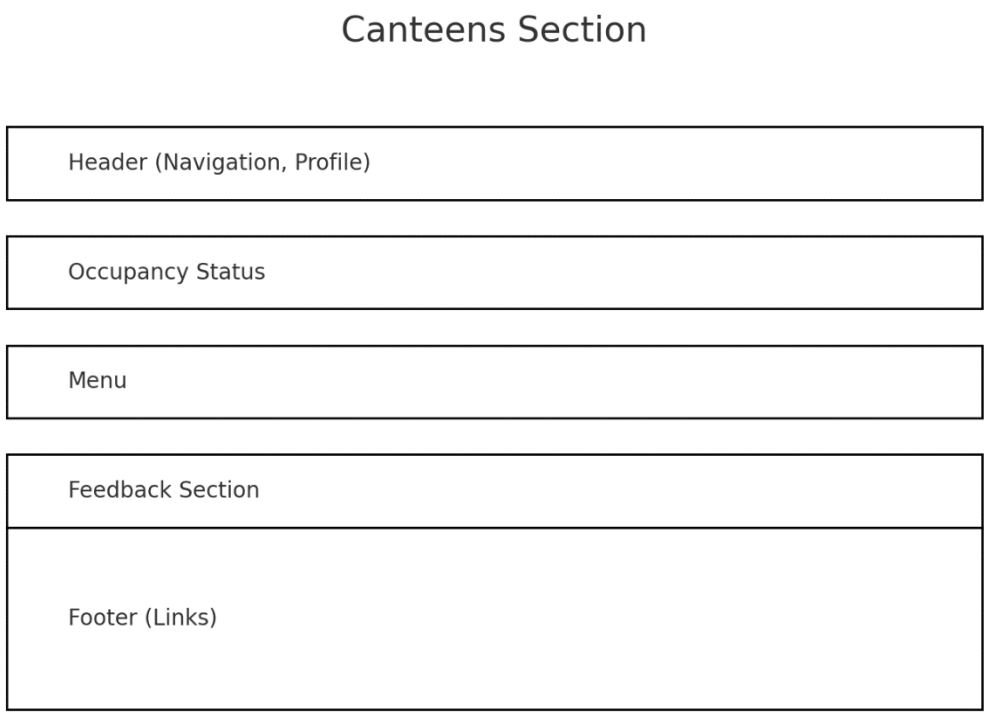
- **Occupancy Status**: A visual indicator (e.g., a bar or pie chart) showing the current occupancy level.
- **Search Functionality**: Allows students to search for specific books or resources and see their availability.
- **Recent Activity**: Displays recent check-ins and check-outs.

Medical Center Section:



- **Availability of Medical Officer**: A status indicator showing whether a medical officer is currently available.
- **Crowd Level**: A visual representation of how crowded the medical center is.
- **Appointment Booking:** Allows students to book appointments with the medical officer.

Canteens Section:



- **Occupancy Status**: Visual indicators for each canteen showing current crowd levels.
- **Menu**: Displays the daily menu for each canteen.
- **Feedback Section:** Allows students to provide feedback on the food and service.

3.2.3. User Interaction Flow

1. **Logging In:**
   - Users log in using their student IDs and passwords.
   - Admins have a separate login page with additional security measures.
2. **Viewing Occupancy Status:**
   - After logging in, users are directed to the home page where they can quickly check the occupancy status of all three facilities.
   - Clicking on any facility gives the user to a detailed view of that facility.
3. **Providing Data:**
   - Students can contribute data by checking in/out at the library and medical center using their student IDs, scanned by the facility worker.
   - Crowd levels for the canteens are updated based on periodic input from students via a simple form.
4. **Admin Functions:**

- Admins log in through a separate interface where they can access additional functionalities.
- They can scan student IDs, update occupancy status, and generate usage reports.

3.2.4. Responsiveness

The UI is designed to be fully responsive, ensuring it works seamlessly across different devices and screen sizes. This is achieved through:

- **Responsive Design Principles:** Using flexible grid layouts, media queries, and scalable vector graphics (SVGs).
- **Mobile-First Approach:** Prioritizing mobile user experience while ensuring desktop compatibility.

3.2.5. Accessibility

We adhere to the Web Content Accessibility Guidelines (WCAG) to ensure our application is accessible to all users, including those with disabilities. Key accessibility features include:

- **Accessible Navigation:** All interactive elements are designed with accessibility in mind.
- **Screen Reader Support:** Ensuring compatibility with screen readers by using proper HTML semantics and ARIA attributes.
- **Contrast and Text Size:** Providing sufficient contrast and adjustable text sizes for better readability.

3.2.6. Usability Testing

We will conduct usability testing sessions with a diverse group of students to gather feedback and identify areas for improvement. This includes:

- **Task-Based Testing:** Observing users as they complete specific tasks to ensure the interface is intuitive and easy to navigate.
- **Surveys and Feedback:** Collecting feedback through surveys and feedback forms to continuously enhance the user experience.

By focusing on these aspects, we aim to create an intuitive, efficient, and user-friendly interface that meets the needs of our students and faculty members.

3.3 Security Plan

Ensuring the security of the "Campus Utilization Monitor" system is a top priority. Our security measures encompass various strategies to protect against common vulnerabilities and ensure secure authentication and authorization. The following detailed security measures will be implemented:

3.3.1. Prevention of Common Attacks:

- **Cross-Site Scripting (XSS):**
  - We will sanitize all user inputs and outputs to prevent the injection of malicious scripts.
  - Implement Content Security Policy (CSP) headers to control which resources can be loaded and executed.
- **NoSQL Injection:**
  - Use parameterized queries and input validation to prevent the injection of malicious code into NoSQL databases.
  - Ensure that database queries are constructed in a safe manner, avoiding direct use of user inputs.
- **HTTP Injection:**
  - Implement strict input validation and sanitization to prevent HTTP header injection.
  - Use secure headers and protocols to protect data transmitted over HTTP.
- **Insecure Direct Object References (IDOR):**
  - Implement proper access controls and authorization checks to ensure that users can only access resources they are authorized to view.
  - Use secure identifiers and avoid exposing direct object references in URLs or APIs.
- **Server-Side Request Forgery (SSRF):**
  - Validate and sanitize all URLs and inputs to prevent SSRF attacks.
  - Implement network-level protections, such as firewalls and access controls, to restrict outbound requests.
- **Cross-Site Request Forgery (CSRF):**
  - Use CSRF tokens to protect against unauthorized actions performed on behalf of authenticated users.
  - Implement same-site cookie attributes and secure headers to mitigate CSRF risks.

3.3.2. Authentication and Authorization:

- **JWT Authentication**:
  - We will use JSON Web Tokens (JWT) for secure authentication, ensuring that user credentials are protected during transmission.
  - JWT tokens will be signed and encrypted to prevent tampering and ensure the integrity of the authentication process.
  - Tokens will include expiration times to minimize the risk of token reuse and mitigate the impact of potential token leaks.

- **Preventing Man-in-the-Middle (MitM) Attacks**:
  - Implement HTTPS across all communications to encrypt data transmitted between clients and servers.
  - Use secure protocols and cipher suites to protect data from interception and tampering.
  - Regularly update and maintain SSL/TLS certificates to ensure secure communication channels.

3.3.3. Encryption Strategies:

- **Data Encryption**:
  - Encrypt sensitive data at rest using strong encryption algorithms (e.g., AES-256).
  - Implement encryption for data in transit to protect against eavesdropping and interception.

- **Database Security**:
  - Use database encryption to protect sensitive data stored in the database.
  - Implement role-based access controls to restrict database access to authorized personnel only.

3.4. Strategies for Performance Optimization and Scalability

To ensure the "Campus Utilization Monitor" system performs efficiently and scales effectively, we will implement the following strategies:

3.4.1. Performance Optimization

- **Caching**:
  - Implement caching mechanisms at various levels (e.g., database query caching, application-level caching) to reduce redundant data retrieval and processing.
  - Use content delivery networks (CDNs) to cache static assets and reduce server load.

- **Asynchronous Processing**:
    - Use asynchronous processing for time-consuming tasks to improve response times and overall system performance.
    - Implement background job queues for handling tasks that do not require immediate processing.
- **Optimization of Database Queries**:
    - Optimize database queries to reduce latency and improve response times.
    - Use indexing and query optimization techniques to enhance database performance.
- **Resource Management**:
    - Monitor and manage server resources (CPU, memory, disk I/O) to prevent resource exhaustion and ensure smooth operation.
    - Implement load balancing to distribute traffic evenly across servers and prevent overloading.

3.4.2. Scalability

- **Microservices Architecture**:
    - Implement a microservices architecture to enable independent scaling of different components based on demand.
    - Use containerization technologies (e.g., Docker) to facilitate easy deployment and scaling of microservices.
- **Auto-Scaling**:
    - Implement auto-scaling policies to dynamically adjust the number of server instances based on traffic and load.
    - Use cloud services (e.g., AWS, Azure) to leverage auto-scaling capabilities and ensure high availability.
- **Database Scalability**:
    - Use database sharding and replication to distribute database load and enhance scalability.
    - Implement read replicas to offload read operations and improve database performance.
- **Monitoring and Analytics**:
    - Implement comprehensive monitoring and analytics to track system performance and identify bottlenecks.

- Use monitoring tools to gather real-time metrics and alerts for proactive performance management.

4. System Development, Testing, and Deployment Strategies

Our project involves developing a community-driven app for checking occupancy status and automating the registry of faculty premises such as the library, medical center, and canteens. We will utilize a structured approach to ensure efficient development, testing, and deployment of both the front-end and backend components of our application.

4.1. Project Management Approach

4.1.1. Scrum Methodology:

- We will use the Scrum methodology for managing the "Campus Utilization Monitor" project. Scrum is an agile framework that supports iterative development, collaboration, and flexibility.

Key Features:

- **Sprint Planning**: Prioritize and plan tasks at the beginning of each sprint.
- **Daily Stand-ups**: Short daily meetings to discuss progress and address obstacles.
- **Sprint Reviews**: Present completed work to stakeholders at the end of each sprint for feedback.
- **Sprint Retrospectives**: Reflect on the sprint to identify successes and areas for improvement.

4.1.2. Jira for Project Management

- Jira will be used to track and manage tasks, sprints, and project progress.

Key Features:

- **Backlog Management**: Create and prioritize the product backlog.
- **Sprint Planning and Tracking**: Plan sprints, assign tasks, and track progress.
- **Task Boards**: Visualize workflow and manage work in progress.
- **Reporting and Analytics**: Monitor progress with burndown charts, sprint reports, and velocity charts.

- **Custom Workflows**: Tailor workflows to match project needs.
- **Collaboration:** Facilitate team communication with comments, mentions, and notifications.

Using Scrum and Jira, we aim to ensure an organized, efficient, and collaborative project environment.

4.2. System Development

4.2.1. Frontend Development:

1. **UI/UX Design and Prototyping:**
   - ✓ **Figma:**
     - Design wireframes for initial layout planning.
     - Develop interactive prototypes to simulate user interactions.
     - Create high-fidelity designs for the final user interface.
     - Conduct usability testing with stakeholders and gather feedback.

2. **Frontend Development:**
   - ✓ **React.js:**
     - Build reusable components for the user interface.
     - Implement state management with Redux.
     - Connect frontend with backend APIs.
     - Ensure responsive design for compatibility across devices.

4.2.2 Backend Development:

1. **Backend Development:**
   - ✓ Node.js:
     - Write server-side logic for handling requests and responses.
     - Manage middleware for authentication and authorization.
     - Express.js:
     - Set up server and define API endpoints.
     - Implement RESTful services for data retrieval and updates.

2. **Database:**
   - ✓ MongoDB:
     - Design database schema to store user data and occupancy status.
     - Implement data models and establish relationships.

3. **Firebase:**
   - ✓ **Authentication**
     - Integrate Firebase Authentication to handle user sign-ups, logins, and secure access.
     - Implement various authentication methods like email/password, Google, Facebook, etc.

   - ✓ **Firestore**
     - Design and implement a NoSQL database schema using Firestore to store user data and occupancy status.
     - Utilize Firestore for real-time data synchronization and offline capabilities.

   - ✓ **Firebase Cloud Functions**
     - Write server-side functions to handle complex business logic and extend backend functionality.
     - Implement background triggers and event-driven functions for tasks like sending notifications or processing data.

   - ✓ **Firebase Storage**
     - Store and manage user-generated content such as images, documents, and other files.
     - Implement file upload, retrieval, and management functionality.

4.2.3. Cross-Component Tools and Technologies:

1. **Agile Project Management:**
   - ✓ **Jira:**
     - Create user stories and tasks.
     - Plan sprints and manage backlog.
     - Track progress with Kanban boards and burndown charts.

2. **Source Code Management:**
   - ✓ **GitHub:**
     - Initialize repository and set up branching strategy (e.g., main, development, feature branches).
     - Commit and push code changes regularly.
     - Create pull requests for code reviews and merge approved changes.

3. **Continuous Integration and Continuous Deployment (CI/CD):**
   - ✓ **Jenkins:**
     - Set up pipelines to automate build and test processes.
     - Integrate with GitHub for triggering builds on code changes.
     - Automated deployment to staging and production environments.

4. **Containerization:**
   - ✓ **Docker:**
     - Create Dockerfiles for backend and frontend services.
     - Build and test Docker images locally.
     - Deploy containers to staging and production environments.

5. **Cloud Deployment:**
   - ✓ **AWS/Azure:**
     - Set up virtual machines and containers for hosting services.
     - Use AWS RDS or Azure SQL Database for managed database services.
     - Implement auto-scaling and load balancing for high availability.
     - Configure monitoring and logging services.

6. **Code Quality Analysis (optional):**
   - ✓ **SonarQube:**
     - Integrate with Jenkins to analyze code quality.
     - Set up rules for code smells, bugs, and security vulnerabilities.
     - Generate reports and act on identified issues.

4.3 Testing

1. **Unit Testing:**
   - ✓ **Frontend:**
     - Jest for testing individual React components.
   - ✓ **Backend:**
     - Mocha/Chai for testing individual functions and server-side logic.

2. **Integration Testing:**
   - ✓ **Tools:**
     - Postman for API testing.

3. **End-to-End (E2E) Testing:**
   - ✓ **Tools:**
     - Cypress for frontend workflow testing.

4. **Performance Testing:**
   - ✓ **Tools:**
     - JMeter and LoadRunner for assessing application performance.

5. **User Acceptance Testing (UAT):**
   - ✓ **Approach:**
     - Involve real users to validate the application meets user requirements.

- Gather feedback and make necessary adjustments.
- Conduct UAT sessions before final deployment.

4.4 Deployment

1. **Continuous Integration:**
   - ✓ Code changes are automatically built and tested using Jenkins.
   - ✓ Ensure all tests pass before merging code into the main branch.

2. **Containerization:**
   - ✓ Use Docker to package the application and its dependencies.
   - ✓ Create Docker Compose files for managing multi-container applications.
   - ✓ Ensure consistency across development, testing, and production environments.

3. **Staging Environment:**
   - ✓ Deploy to a staging environment for final testing.
   - ✓ Use the same configuration as the production environment.
   - ✓ Conduct thorough testing to ensure readiness for production.

4. **Production Deployment:**
   - ✓ Use AWS/Azure for final deployment.
   - ✓ Set up auto-scaling to handle variable loads.
   - ✓ Configure load balancers to distribute traffic evenly.
   - ✓ Implement monitoring and logging for real-time insights.

5. **Monitoring and Maintenance:**
   - ✓ Use cloud platform tools for monitoring application health and performance.
   - ✓ Set up alerts for potential issues and automated responses.
   - ✓ Regularly update and maintain the application to ensure security and performance.

By following these detailed steps for both front-end and backend development, we ensure a structured and efficient approach to developing, testing, and deploying our web application, delivering a robust and reliable solution to our users.

5. Project Milestones and Timeline

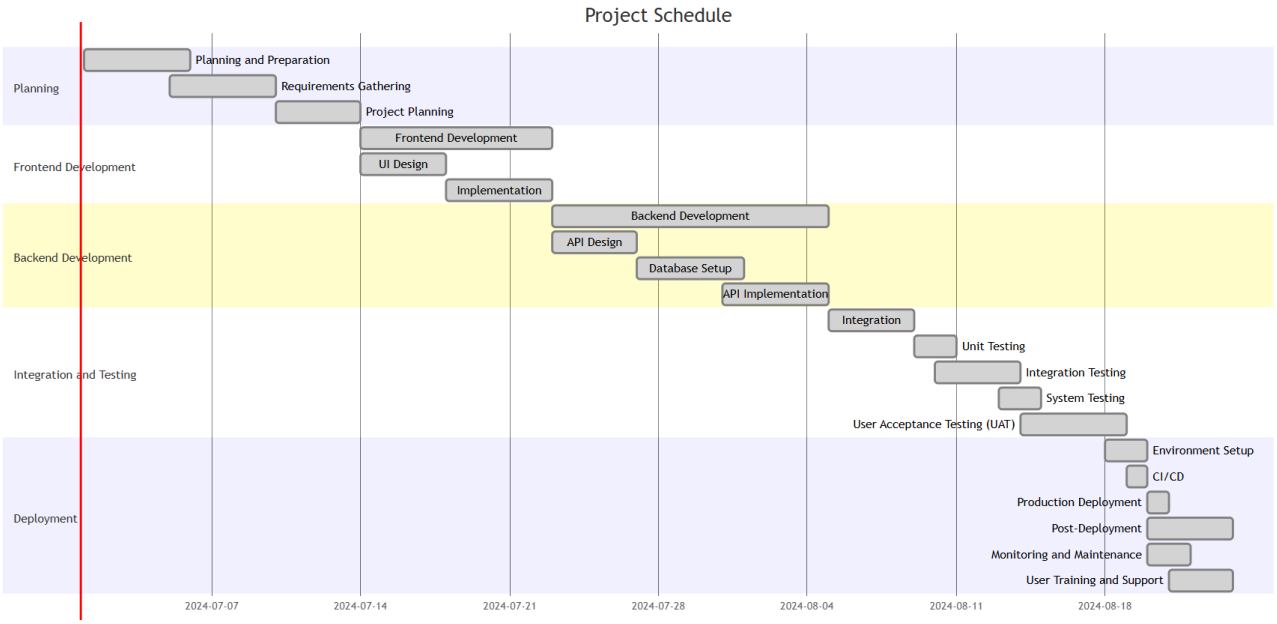
The project is divided into several key phases, each with specific deadlines and evaluation points to ensure timely progress and quality delivery. The timeline below outlines the milestones and the corresponding activities for each phase:

5.1 Milestones

Phase	Milestone	Tasks
Design	Project Planning and Requirements Gathering	▪ Conduct meetings with stakeholders ▪ Document functional and non-functional requirements ▪ Develop a detailed project plan
	UI Design	▪ Create wireframes and mockups for the application ▪ Review and finalize designs with stakeholders
Development	Frontend Development	▪ Implement UI components using React.js ▪ Integrate frontend with backend APIs
	Backend Development	▪ Design and implement RESTful APIs ▪ Set up and configure the database ▪ Develop backend logic using Node.js and Express.js
	Integration	▪ Ensure seamless data flow between client and server ▪ Perform integration testing
Deployment	Testing	▪ Conduct unit, integration, and system testing ▪ Perform User Acceptance Testing (UAT)
	Deployment	▪ Set up development, staging, and production environments ▪ Configure CI/CD pipelines with Jenkins and Docker ▪ Deploy the application to the production environment
Post-Deployment	Post-Deployment Activities	▪ Set up monitoring tools for application performance ▪ Provide user training and support ▪ Plan for regular maintenance and updates

5.2 Timeline

The project timeline is designed to ensure a structured approach to project completion, with regular evaluations to track progress and address any issues promptly. By adhering to this timeline, we aim to deliver a high-quality, fully functional application that meets the needs of the students and faculty.



*** End of the Project Proposal ***