# Project Report

Midpoint Deliverable

Riya Danve Riya Deshpande Riya Adsul Mitali Juvekar

Github Repo Link: https://github.com/riyaa74/PatientTrackerSystem

## 1. Requirements

## 1.1. Overview

- Guideline: Start by giving a brief description of the project, its objectives, and its intended audience. Summarize the primary goals and motivations behind the project.

**Example for Patient Tracker**- Briefly explain the importance of a digitalized system for patient tracking in the current healthcare ecosystem. Highlight the challenges of manual paperwork and the need for real-time updates in medical records.

## 1.2. Features

- Example: List out at least five main features of your project. For instance, "User Authentication, Profile Management, Data Visualization, Real-time Notifications, and Customizable Dashboard."

## 1.3. Functional Requirements (Use cases)

- Guideline: Detail out the functional aspects of the project. This would be where you dive into the technical side, explaining what each function does. Describe over 10 user stories or use cases.

Describe the actions that users (doctors and patients) can perform in the system. Example: "As a doctor, I want to view the list of patients scheduled for the day so that I can prepare for the consultations."

## 1.4. Non-Functional Requirements

- Guideline: These are the requirements that do not pertain directly to the functionality of the project but relate to user experience, reliability, performance, etc.

Mention system constraints such as performance, reliability, and security considerations. Example: "The system should ensure data encryption and secure transmission of patient data."

# 2. Design

## 2.1. Architecture Diagram

- Guideline: Illustrate the high-level structure of your project using a diagram. Show how different components of your project interact with each other. Describe the choice of using an Object-Oriented programming language, such as Java or Python, for the implementation.

-\*Example: A flowchart showing how data flows between the user interface, server, and database.

## 2.2. Technology Stack with Justification

- Guideline: List out the technologies used in the project and provide reasoning for the choice. Discuss the advantages of chosen technologies in relation to the project's requirements.

- Example: "We chose Flask for backend development due to its lightweight nature and flexibility in developing web applications."

## 2.3. UI Mockup

- Guideline: Provide visual representations or wireframes of the user interface. This helps in giving a tangible sense of how the user would interact with the product.

- Example: Screenshots or sketches of the main pages/screens of the application. Showcase a few user interface designs and layouts for the system, emphasizing doctor and patient interactions.

## 2.4. Data Model

- Guideline: Illustrate how data is structured and organized in your project. This may include database schema diagrams, entity-relationship diagrams, etc.

- Example: A diagram showing the tables in your database, their columns, and the relationships between tables. Provide an Entity-Relationship (ER) diagram or a class diagram to represent the structure of patient data. Describe the relationships between entities, such as Doctor-Patient, Patient-Medications, and so on.

# 3. Implementation

1. Outline the coding process, methodologies to be followed, and potential challenges.:

**Process:**

* Understanding Requirements and Planning: Our first step is to thoroughly understand the project requirements and create a detailed plan. This includes defining the user roles (admin, doctor, patient), their functionalities, and the data they need to interact with. We should also clarify the data schema for our SQL database.
* Setting up the Development Environment: To begin coding, we need to set up our development environment. This includes installing Flask, a SQL database, and any other necessary libraries. We'll choose a code editor or integrated development environment (IDE) and create a version control repository for our project (e.g., using Git).
* User Authentication and Login Page: Implementing user authentication is a crucial first step. We'll use Flask-Login or a similar extension to manage user sessions and create a login page. It's important to ensure that only authorized users can access their respective interfaces.
* Building the Admin Interface: The admin interface should be capable of adding, editing, and removing doctors, patients, and appointments. We will also need a configuration system to manage user permissions. Flask-WTF can be used to create forms for data input.
* Creating the Doctor Interface:The doctor interface should allow doctors to add, edit, and remove patients and appointments. We should also provide functionality for doctors to add/edit patient records and view health records and graphs. Data visualization libraries such as Matplotlib or Plotly can help with graph generation.
* Developing the Patient Interface: For patients, we'll build a user interface that allows them to add/edit/remove appointments and view their information and graphs. Data visualization will play a key role in presenting their health records.
* User Management: Implement a user management system to handle user profiles, roles, and permissions. Flask-Principal or Flask-User can help manage user roles and access control.
* Database Setup and Interaction: Set up the SQL database and define the tables for doctors, patients, appointments, and health records. Use an Object-Relational Mapping (ORM) library like SQLAlchemy to interact with the database from your Flask application.
* Data Visualization: Implement data visualization using appropriate libraries to generate health graphs and records. Matplotlib or Plotly can be integrated into the Flask application to create these visualizations.
* Notifications/Alerts: Integrate a notification system for appointment reminders and alerts. This could be done using email notifications or SMS services.

**Methodologies:**

* Agile Development: We'll follow an agile development methodology, breaking the project into sprints with specific goals and features for each sprint. Regular meetings and feedback loops will be essential.
* Version Control: We'll use Git and platforms like GitHub to manage and collaborate on the codebase, allowing multiple team members to work on different parts of the project simultaneously.
* Testing: Comprehensive unit and integration testing will be a priority to ensure the application works as expected. We'll consider using testing frameworks like PyTest.
* Documentation: We'll maintain documentation for code, database schemas, and user guides. This will aid in project management and future maintenance.

**Potential Challenges:**

* Security: Ensuring the system is secure and protecting sensitive patient data from unauthorized access.
* Scalability: As the number of users and data grows, the application should remain performant.
* User Experience: Designing a user-friendly interface for doctors and patients to efficiently manage appointments and health records.
* Data Integrity: Ensuring the accuracy and integrity of patient health records and appointment data.
* Regulatory Compliance: Adhering to healthcare regulations and data privacy laws, such as HIPAA if applicable.
* Notifications: Implementing a reliable notification system that delivers alerts and reminders to users on time.
* Data Visualization: Creating informative and visually appealing health graphs and records.
* Database Management: Managing database migrations and backups as the application evolves.

## 3.1. Security and Risks

1. **User Authentication and Authorization:**
   * Threat: Unauthorized access to the system by someone impersonating a legitimate user.
   * Vulnerability: Weak password policies and lack of multi-factor authentication.
   * Prevention: Implement strong password policies, encourage users to use unique and complex passwords, and consider implementing multi-factor authentication to enhance security.
2. **Data Storage and Access Control:**
   * Threat: Unauthorized users gaining access to sensitive patient data.
   * Vulnerability: Insufficient access controls and improper database configurations.
   * Prevention: Use proper role-based access control (RBAC) to limit what each user type (admin, doctor, patient) can access and ensure the database is properly configured with appropriate permissions.
3. **Data Encryption:**
   * Threat: Data interception during transmission between the client and the server.
   * Vulnerability: Unencrypted data transmission.
   * Prevention: Implement SSL/TLS for secure data transmission between clients and the server to encrypt data in transit.
4. **SQL Injection:**
   * Threat: Malicious users exploiting vulnerabilities in SQL queries to access or modify the database.
   * Vulnerability: Insufficient input validation and lack of prepared statements.
   * Prevention: Sanitize user inputs, use prepared statements, and employ ORM (Object-Relational Mapping) tools to prevent SQL injection attacks.
5. **Data Backups:**
   * Threat: Data loss due to system failures, accidental deletion, or other unforeseen events.
   * Vulnerability: Inadequate or non-existent data backup procedures.
   * Prevention: Regularly backup the database and implement backup and recovery procedures to ensure data availability.
6. **Session Management:**
   * Threat: Unauthorized access to another user's session.
   * Vulnerability: Insecure session management.
   * Prevention: Use secure session management practices, such as generating and storing session tokens securely.
7. **Notification and Alert Security:**
   * Threat: Unauthorized users receiving sensitive notifications or alerts.
   * Vulnerability: Lack of access control for notifications.
   * Prevention: Ensure that notifications and alerts are only sent to authorized users based on their roles and permissions.
8. **Cross-Site Scripting (XSS):**
   * Threat: Malicious scripts injected into web pages viewed by other users.
   * Vulnerability: Inadequate input validation and output encoding.
   * Prevention: Implement proper input validation and output encoding to mitigate XSS attacks.
9. **Data Visualization Security:**
   * Threat: Unauthorized access to patient health records and graphs.
   * Vulnerability: Inadequate access controls for data visualization.
   * Prevention: Apply access controls to patient health records and graphs, ensuring only authorized users can access this information.

To prevent these potential threats and vulnerabilities in our patient tracking system, we will follow best practices for security, including proper authentication, access control, encryption, and data validation. Regularly updating and patching the software and conducting security testing will also be essential for maintaining the security of our system.

3.2 HIPAA Compliance (in case of Patient Tracker)

The Health Insurance Portability and Accountability Act (HIPAA) is a crucial federal law in the United States that was enacted to protect patients' sensitive health information. HIPAA sets standards and regulations for the security and privacy of protected health information (PHI). Our project, the patient tracking system, is closely related to healthcare and involves the storage and management of patient data, making HIPAA compliance extremely relevant. The key aspects of HIPAA that pertain to our project are:

1. **Privacy Rule:** HIPAA's Privacy Rule mandates the protection of patients' PHI by restricting access to authorized personnel only. It is vital for ensuring that patients' personal information, medical history, and health records remain confidential and secure.
2. **Security Rule:** HIPAA's Security Rule outlines the requirements for safeguarding electronic PHI (ePHI). Since our system will involve storing patient records in a database, we must adhere to these security standards to prevent data breaches or unauthorized access.

**Steps and protocols to ensure HIPAA compliance in our project:**

1. **Access Control:** Implement robust access control mechanisms to ensure that only authorized users, such as healthcare professionals and administrators, can access the system. Role-based access control (RBAC) can be employed to restrict access to specific functions and data.
2. **Encryption:** Encrypt data both in transit and at rest. Use secure protocols (e.g., HTTPS) to protect data during transmission and encryption mechanisms to secure data stored in the database.
3. **User Authentication and Authorization:** Implement strong user authentication methods, such as two-factor authentication (2FA), to verify the identity of users. Authorize users to access only the data and functions necessary for their roles.
4. **Incident Response Plan:** Develop a comprehensive incident response plan to address potential data breaches. This plan will include protocols for reporting and mitigating breaches while complying with HIPAA breach notification requirements.
5. **Regular Audits and Assessments:** Periodically conduct security risk assessments and audits to identify vulnerabilities and ensure ongoing compliance. Address any issues promptly.
6. **Documentation:** Maintain thorough records of policies, procedures, and security measures to demonstrate compliance in case of audits or investigations.

## 3. Work Plan

- Guideline: This is where you detail the steps to be taken to execute the project. Think of it as a roadmap or high-level timeline. You also specify each group member’s responsibility and planned contribution to the project development.

- Example:

Group Member 1: List of tasks to be done

Group Member 2: List of tasks to be done

Group Member 3: List of tasks to be done

High-level Timeline: This can include the incremental development plan. You can follow the Spring development plan presented in Moodle too.