# Software Requirements and Design Document

**- Increment 1 -**
**Group #2**


## Authors:

*Daivion Brooks*
*Brooks Berry*
*Jason Lee*
*Nadav Peper*
*Nandagopal Nair*

## 1. Overview (5 points)

*The following design implementations have been made as of increment 1 on October 3, 2025.*
*Database*

- *Runs through Docker*
- *Yml file specifies the services such as volumes, environment, and ports*
- *SQL Schema - includes user profile table, authentication table, and medical data tables such as labs, symptoms, diet, and treatments. All tables include fields that we deemed necessary for the initial foundation of the app.*
- *Models implemented in backend to match the schema*
- *Enabled alembic through flask so that schema changes can be handled through migrations instead of raw SQL*

*Backend*

- *Runs through Flask*
- *Written in Python*
- *Flask app initialization*
- *Models - align with database schema*
- *Routes - Modular CRUD implementation for database fields - routes directory*
- *Secure authentication with JWT tokens and hashed password using Werkzeug library*

*Frontend*

- *Runs through React (requires Node.js)*
- *Written in TypeScript*
- *React components configured to Vite with tailwind CSS*
- *Sign up, Home, and Profile pages created with basic skeleton*
- *Authentication page connected to backend routes*

## 2. Functional Requirements (10 points)

*Functional requirements by priority*

- *(high) CRUD implementation for treatments, symptoms, medications, doctors, etc. Displayed via frontend on different pages in the app*
- *(low)Data should be able to be read on an embedded calendar. For example, they should see data such as upcoming treatments/appointments, or their medication schedule*
    - *Embedded Calendar will have a single page in the app, where the user is able to read different fields of data that are paired with a specific date*
    - *Should be able to filter which fields you can see. Freedom to display all together or select any number of types*
- *(medium)Data should also be read on other pages via list view*
- *(low)Filter feature to display only specified data (upcoming/completed treatments, daily symptoms, specific lab entries, appointments, etc.)*
- *(low)Graphs using Numpy library to show trends for measurable data*
- *(medium)Web scrape to find doctors in the area*
- *(medium)Web scrape for research related to user condition/illness*
- *(high)Scheduled Push notifications for treatment reminders*
- *(low)Feature to export specified user data to a pdf file*
- *(high)secure log in with log in screen*
- *(medium) implement RBAC*
    - *user types would include patients and doctors*

        - *(low)Some variation in pages*
    - *(low)Potentially implement messaging between doctors and patients with push notifications*
- *(high) User home page with access to stored data viewership features*
- *(high)Navbar with routing to other pages*
- *(high) logout button on NAVBAR*
- *(high) Pages/Views for all backend features*
- *(high) Button click implementation for all CRUD feature*
- *(high) Symptoms data CRUD: users will log the date, user notes, and user rating (how do you feel out of 10) (for numerical tracking)*
- *(high) Treatments: user will log scheduled treatments as well as completed treatments. They will store the date, notes, and completion flag. Push notifications for approaching date*
- *(high) Lab data (for example, rbc count, blood pressure, etc.):*
- *(high)User data: user can update profile information such as email, age, weight, medications, insurance, picture, etc.*
    - *(high)Read on automatic display on main user homepage*

## 3. Non-functional Requirements (10 points)

***Security:*** *All database queries should be secure and not prone to SQL injections, ensuring that sensitive user information is protected from unauthorized access and data leaks.*

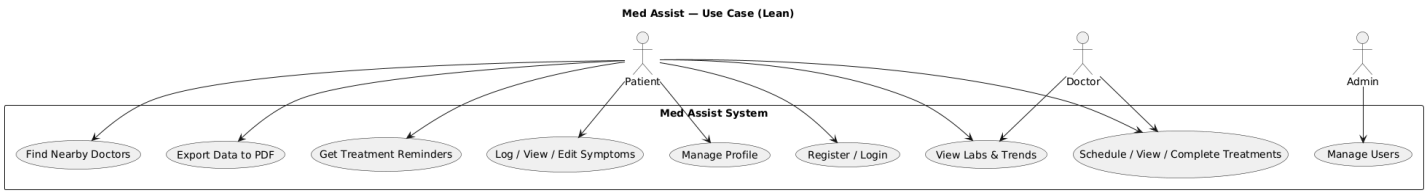
***Maintainability:*** *The system should be designed for ease of maintenance so that all team members can understand, update, and test the codebase with minimal difficulty.*

***Performance:*** *All API endpoints should respond within 1-2 seconds during regular usage. This ensures end-users can use the application efficiently.*

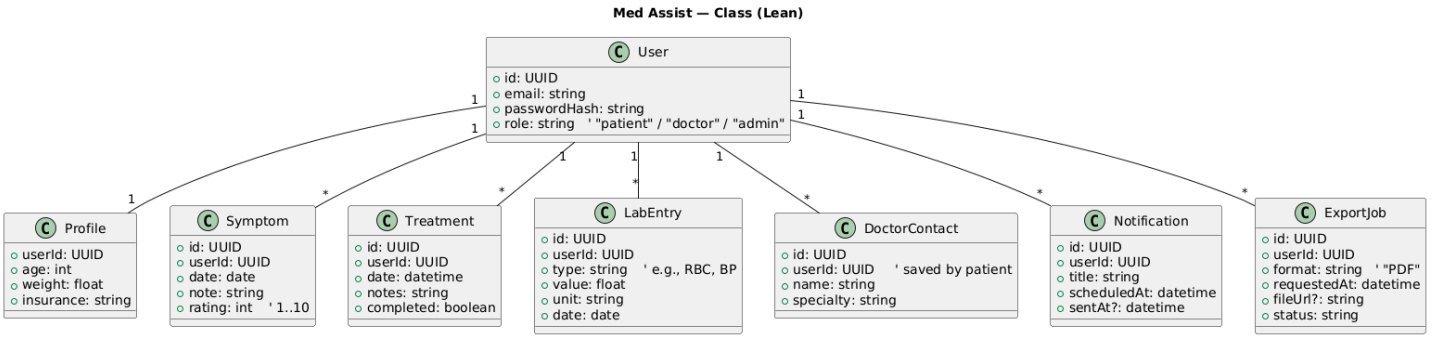
***Usability:*** *The system should provide an accessible and readable interface with clear navigation, so that patients can understand and utilize the application without confusion.*

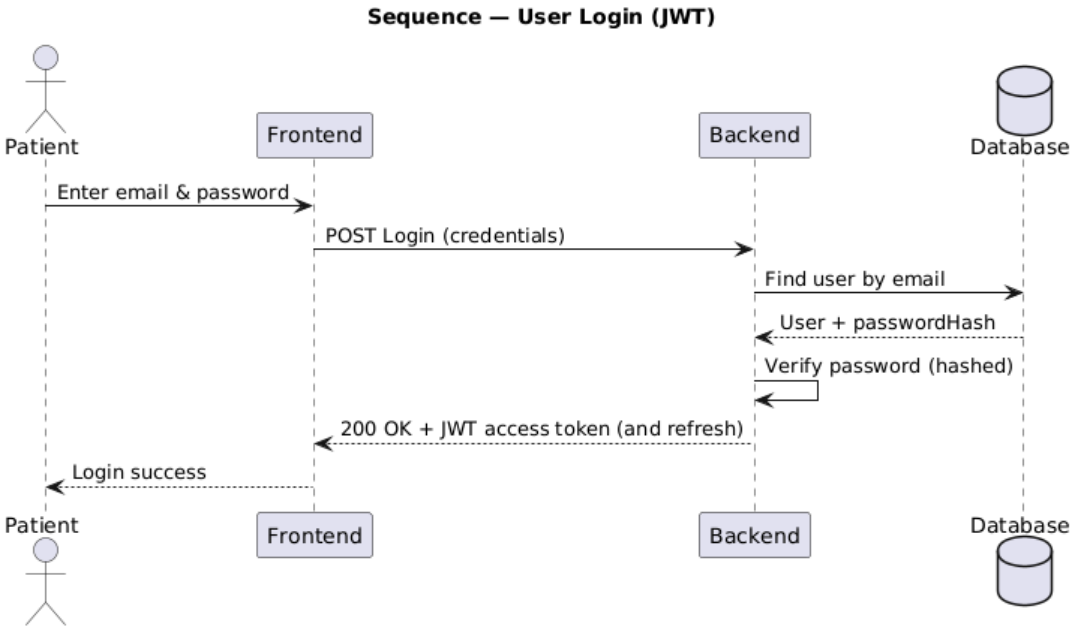
***Reliability:*** *The system must reliably store and retrieve data from the database after migrations to maintain data integrity and ensure that end-users' information remains consistently accessible.*

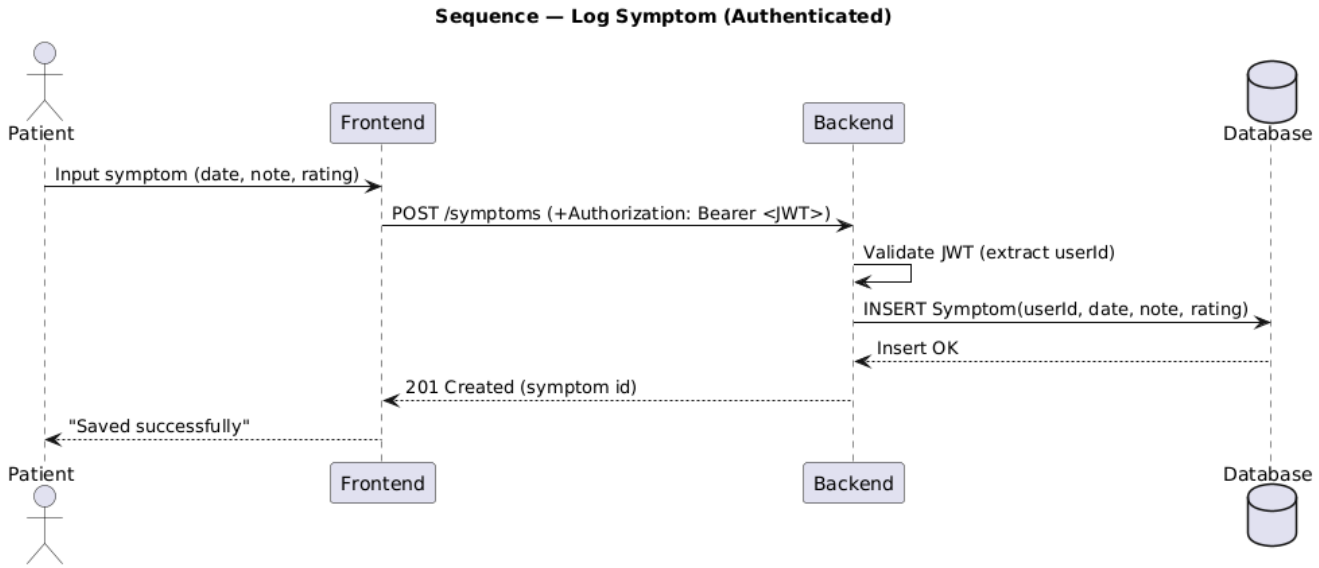
## 4. Use Case Diagram (10 points)

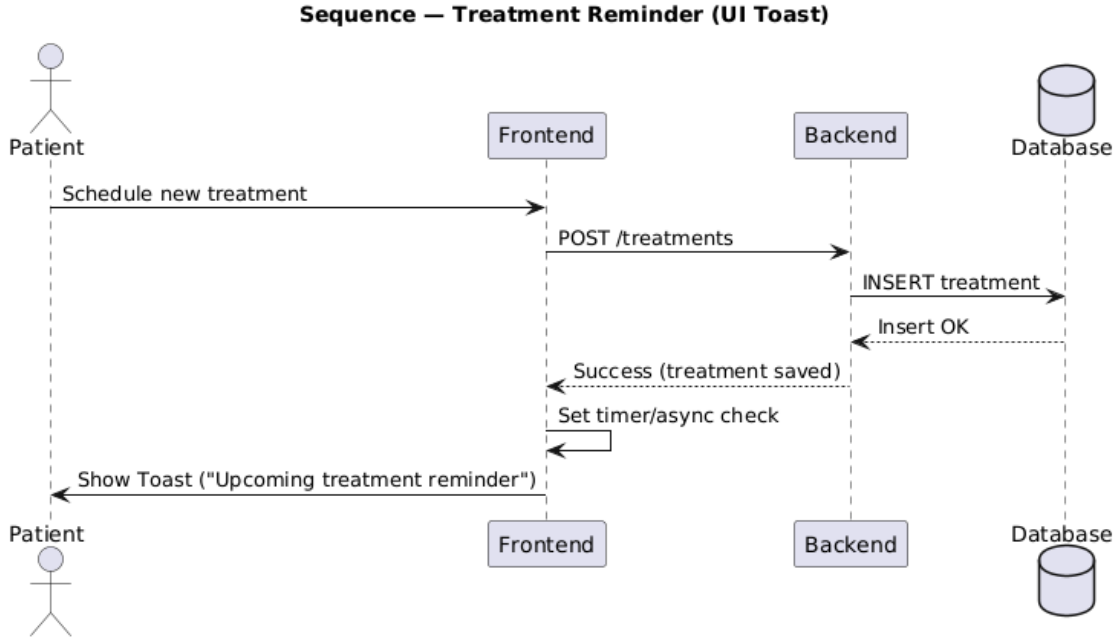


## 5. Class Diagram and/or Sequence Diagrams (15 points)






## 6. Operating Environment (5 points)

***Hardware Platform:*** *The frontend can run on standard personal computers (Windows, macOS, Linux) currently. The backend will operate on a server environment capable of running Docker containers.*

***Operating System:*** *The frontend is platform independent and only requires a supported browser. The backend can be deployed on any OS that supports Docker. Development was performed primarily on Windows and macOS machines.*

***Web Browser Support:*** *The application is compatible with major browsers including Google Chrome, Microsoft Edge, and Safari.*

***Database:*** *The backend uses PostgreSQL for persistent storage of user accounts, symptoms, and treatments.*

## 7. Assumptions and Dependencies (5 points)

***Assumptions***

*Team members will continue using Docker to keep development environments consistent across machines.*

*The database schema will not undergo major changes beyond small adjustments (such as adding columns) during this increment.*

*The backend and frontend will communicate through JSON, and all future features will follow that.*

*Manual testing with Postman and browser-based checks is needed.*

*Users are expected to input their own data (symptoms, treatments) directly, since integration with external medical systems is not planned for this increment.*

***Dependencies***

*The system depends on external libraries/frameworks including Flask, Marshmallow, JWT, React/TypeScript, and PostgreSQL.*

*The project relies on Docker for containerization and environment consistency.*

*GitHub is required for version control and collaboration.*

*The future "Find a Doctor" feature will rely on a third-party API, which may cause some usage limits or require a lot more configuration.*