**CS673 Software Engineering** 

**Team 3 - MyMedic**

**Software Design Document**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| Adriel Domingo | Requirements Lead | *Adriel Domingo* | 6/15/2025 |
| John Gutierrez | All | *John Gutierrez* | 6/16/2025 |
| Mengliang Tan | QA Lead | *Mengliang Tan* | 05/25/2025 |
| Tyler Gonsalves | Configuration Lead | *Tyler Gonsalves* | 05/25/2025 |
| Uzay Isin Alici | Security Lead | *Uzay Isin Alici* | 05/25/2025 |
| Indra Sigicharla | Team / Configration | *Indra Sigicharla* | 6/9/2025 |
|  |  |  |  |
|  |  |  |  |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
| 1.0 | Team 3 | 5/25/2025 | N/A |
| 1.1 | Team 3 | 6/09/2025 | Updated class diagrams, software architecture, ui design, AI usage, and design patterns.+ |
| 1.2 | Team 3 |  | Updated AI usage |

[Introduction](#_87t9hln2vjz0)

[Software Architecture](#_3ipvmjgn6clp)

[Class Diagram](#_ky60nv8suxxm)

[UI Design (if applicable)](#_7ucksmkf6rzx)

[Database Design (if applicable)](#_tcmuor4nl1kz)

[Security Design](#_x18fj36s1121)

[Business Logic and/or Key Algorithms](#_mtfbusfb0eq3)

[Design Patterns](#_f4irtjyz36a8)

T[hree Tier Architecture](#_cr97iltbklku)

REST [REST API](#_9a2dvlqtk0j2)

[Any Additional Topics you would like to include.](#_15tmymhipvdv)

[References](#_50ojo9i46ytq)

[Glossary](#_8n34lvocupub)

# Introduction - Uzay Isin Alici

In this section, we provide an overview of the MyMedic Software Design Document and outline the design goals of the system. This document describes the structural and behavioral design of the MyMedic application, a healthcare-focused platform that helps users manage appointments, access medical records and communicate with healthcare assistants and family members.

The document includes key aspects of the system design such as software architecture, class diagrams, UI layout, database schema, security mechanisms, business logic and design patterns. Each section is contributed by different team members and reflects collaborative planning and decision-making throughout the development process.

The main design goals of the MyMedic system are:

* To ensure modularity and scalability using a microservices-based architecture.
* To maintain security and privacy, in compliance with HIPAA regulations.
* To provide a user-friendly interface with clearly defined user roles.
* To enable flexibility and maintainability for future expansion.

This design aims to support a robust, secure and efficient healthcare application that delivers value to both users and administrators.

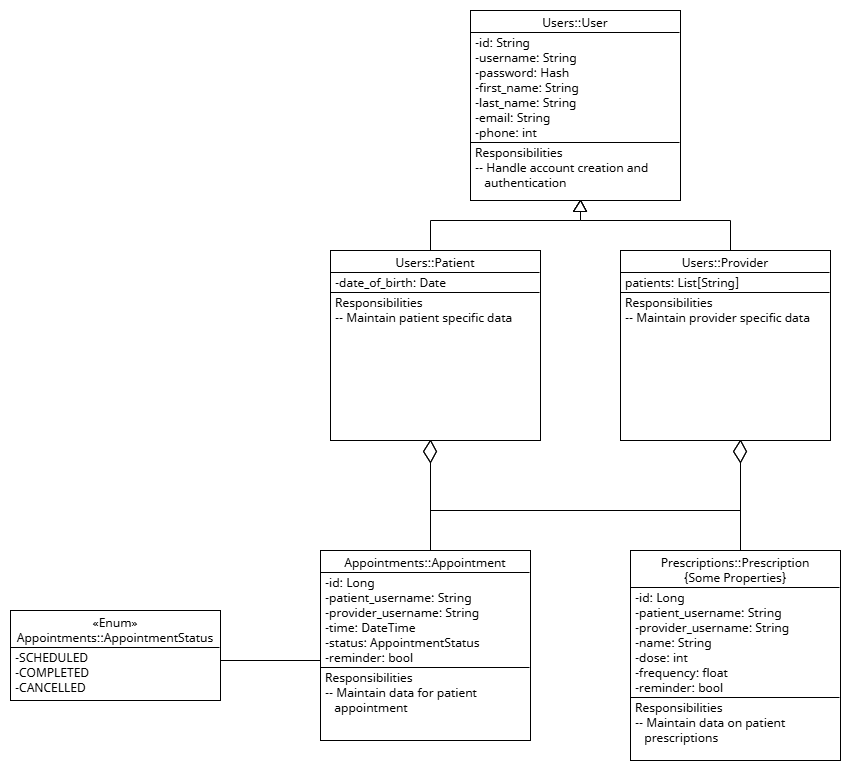
# Software Architecture - Tyler & [John Gutierrez](mailto:johng13@bu.edu)



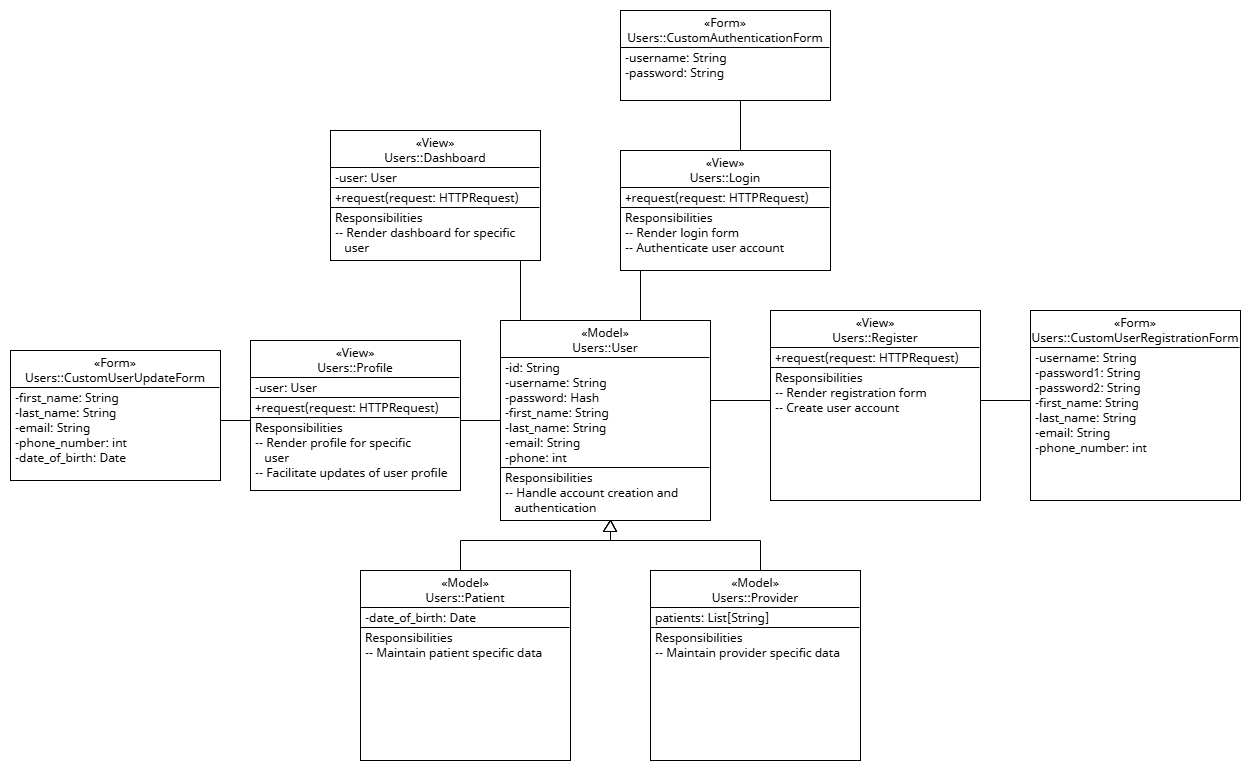
The **My Medic app** is built with a clean and organized three tier Django structure. It has three main parts working together: **scheduling appointments**, **managing users like doctors**, **patients**, and **pharmacists**, and **handling all medical records** such as **lab results** and **visit notes**. Even though everything runs inside one Django application, each part is clearly separated and handles its own responsibilities. All the data is stored in a single, centralized SQLite database. On the frontend, a Node.js interface lets users interact smoothly with the system whether they are booking appointments or reviewing records. The frontend and backend talk to each other through APIs, ensuring a real-time experience that is secure and role-based. It’s a practical, scalable design that keeps things simple but flexible for future growth.

# Class Diagram - Tyler

## System Model Relationships

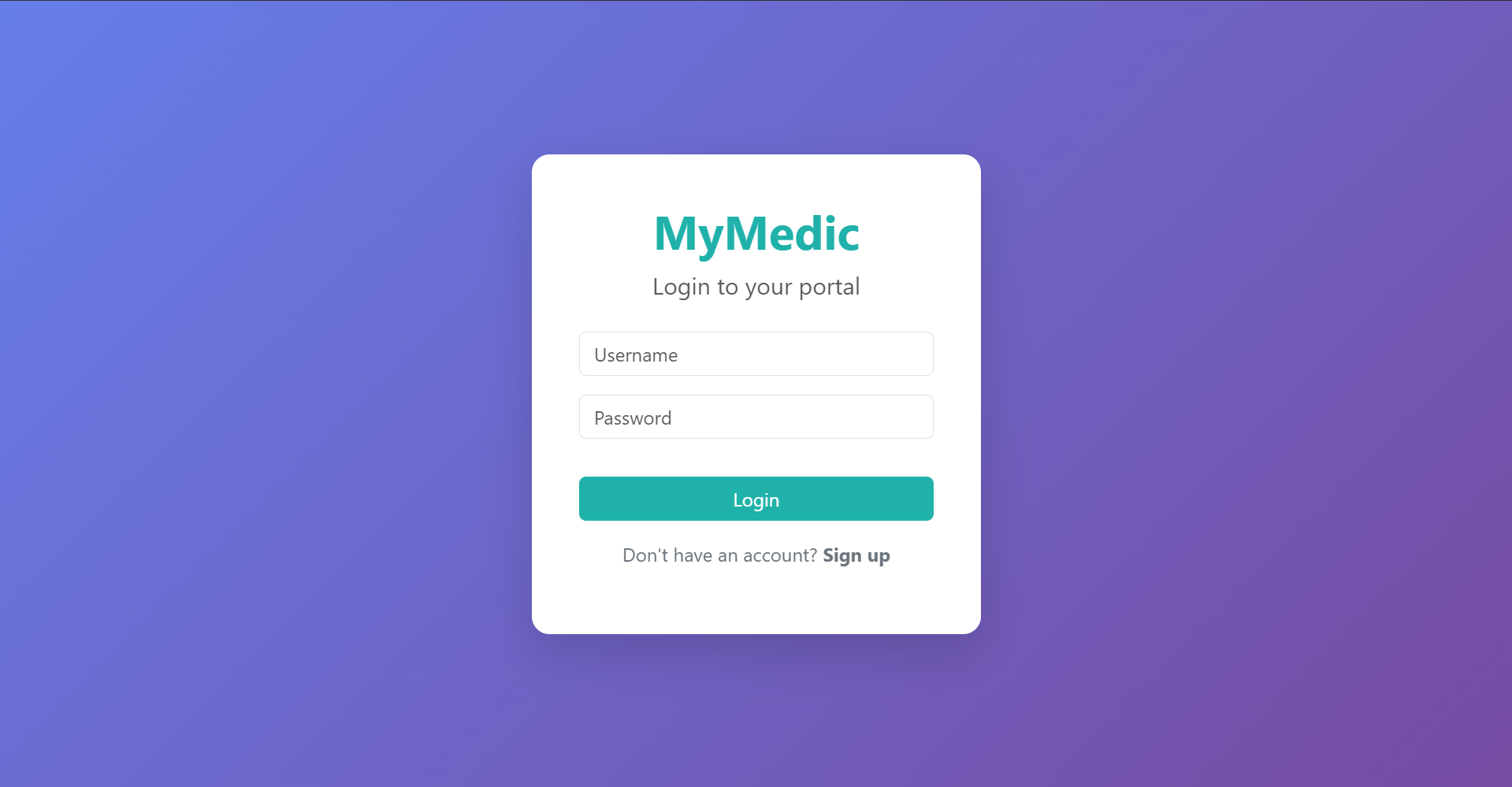


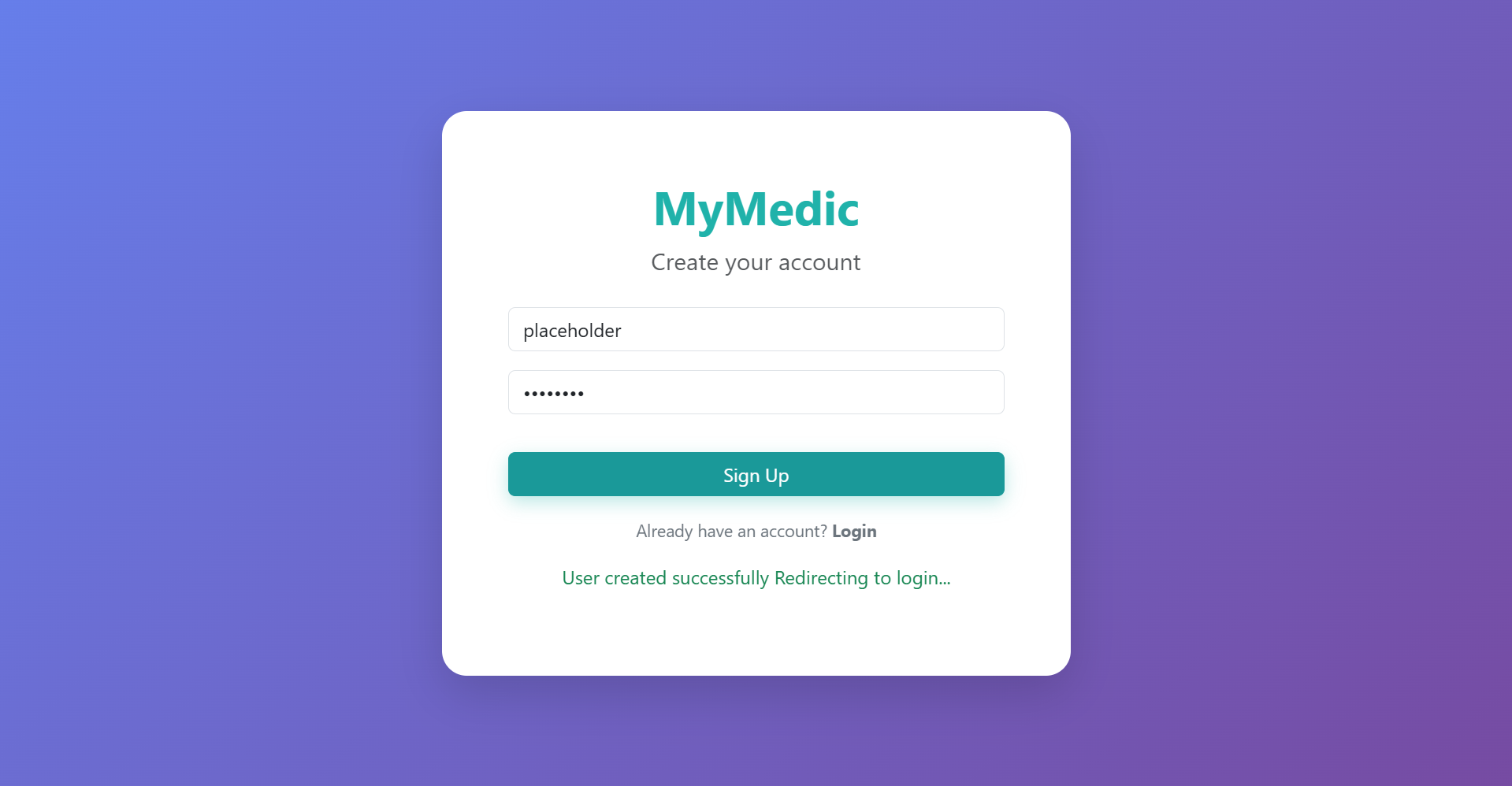
## Model/View Relationships



# UI Design - Indra Sigicharla

## User Login and Sign Up Screen

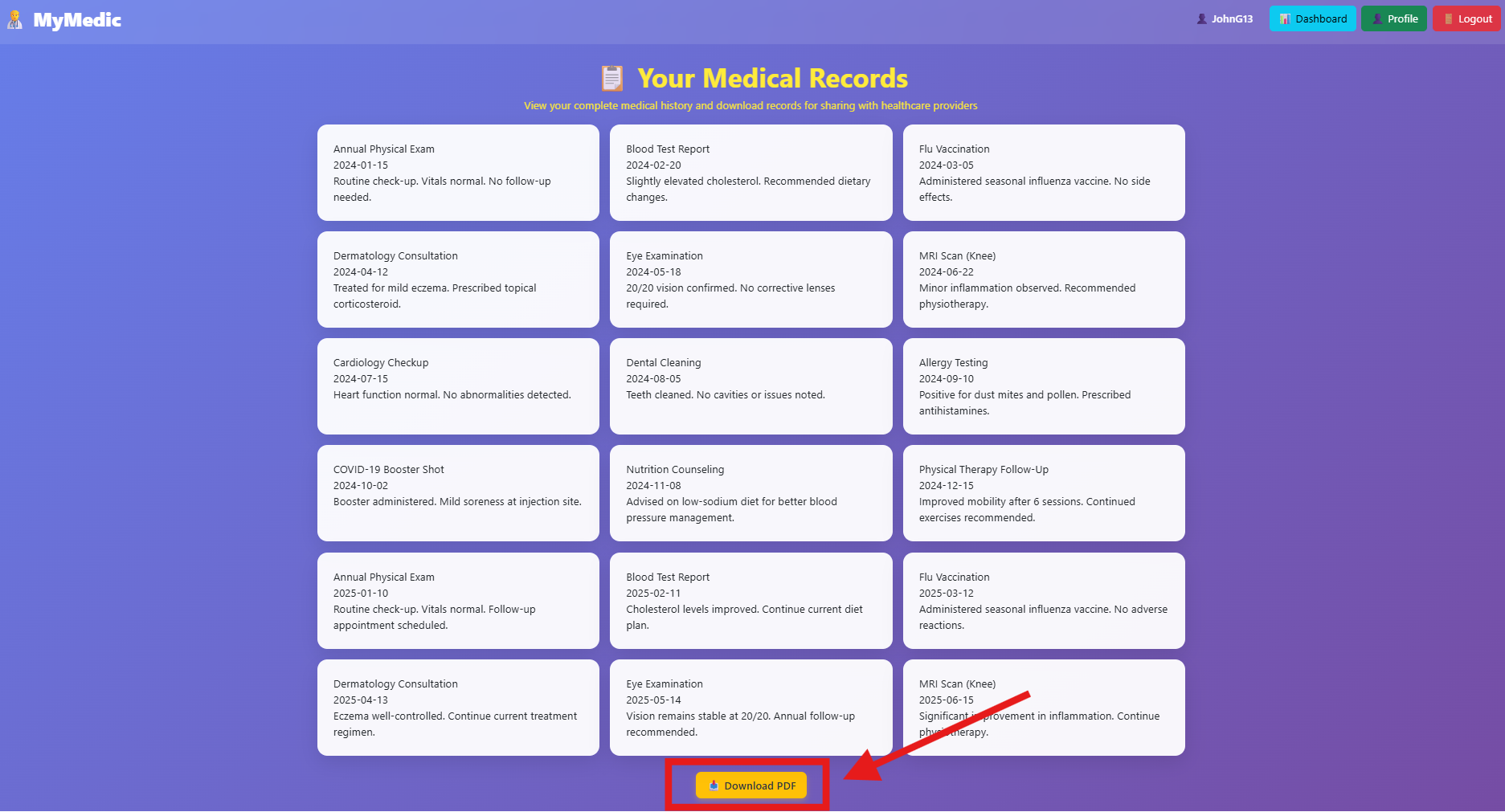
****

****

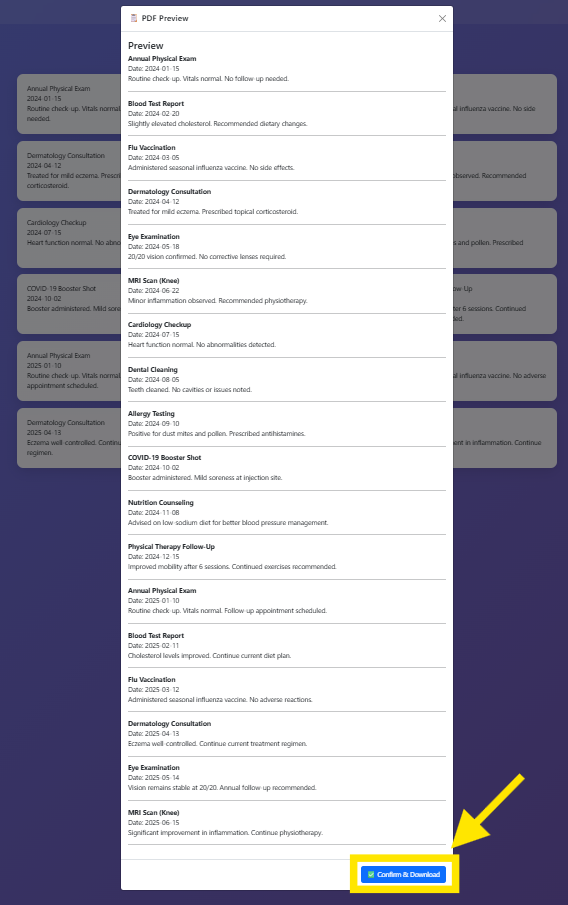
## User Dashboard Screen

## 

**View Records & Download PDF Screens -** [**John Gutierrez**](mailto:johng13@bu.edu)

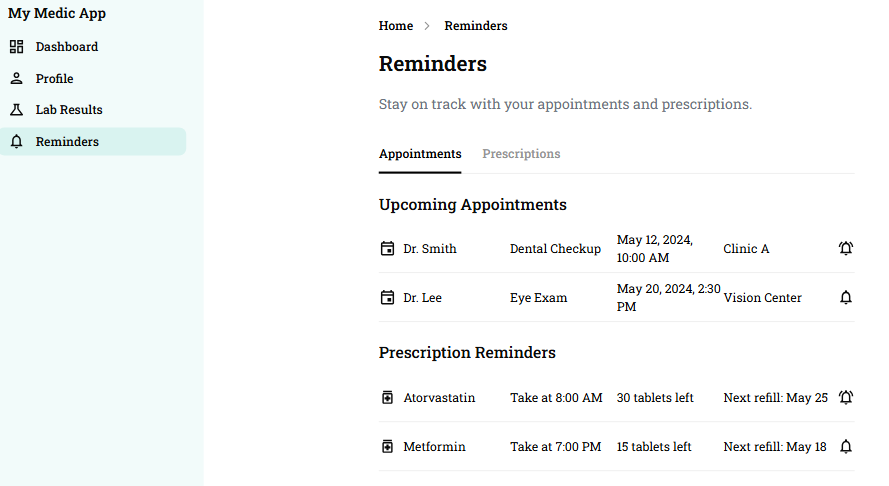
**Medical Records Screen**

**PDF Preview/Confirm & Download View**

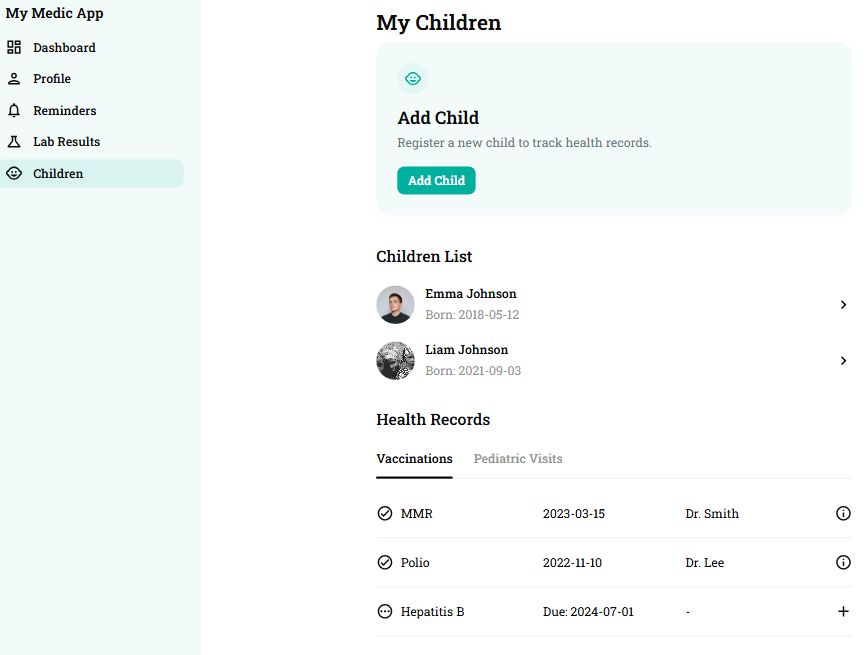


## 

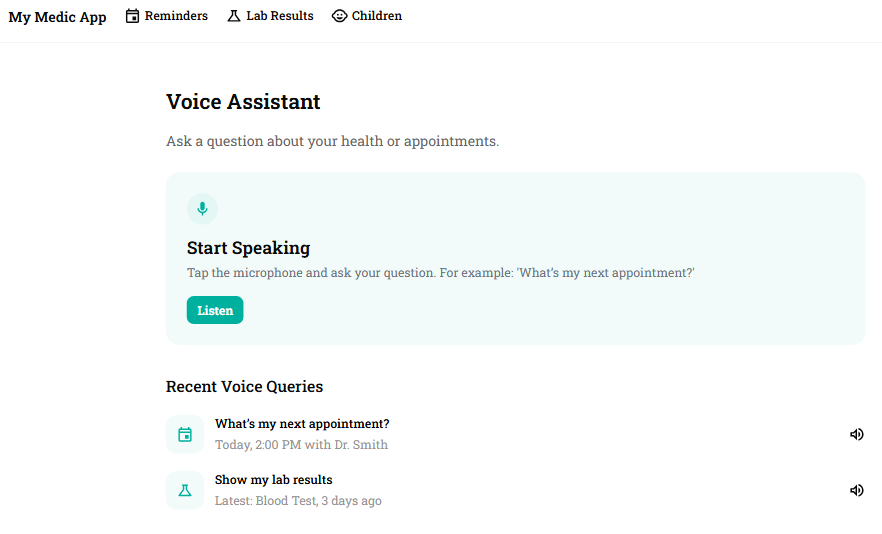
## User Calendar and Reminders Screen

****

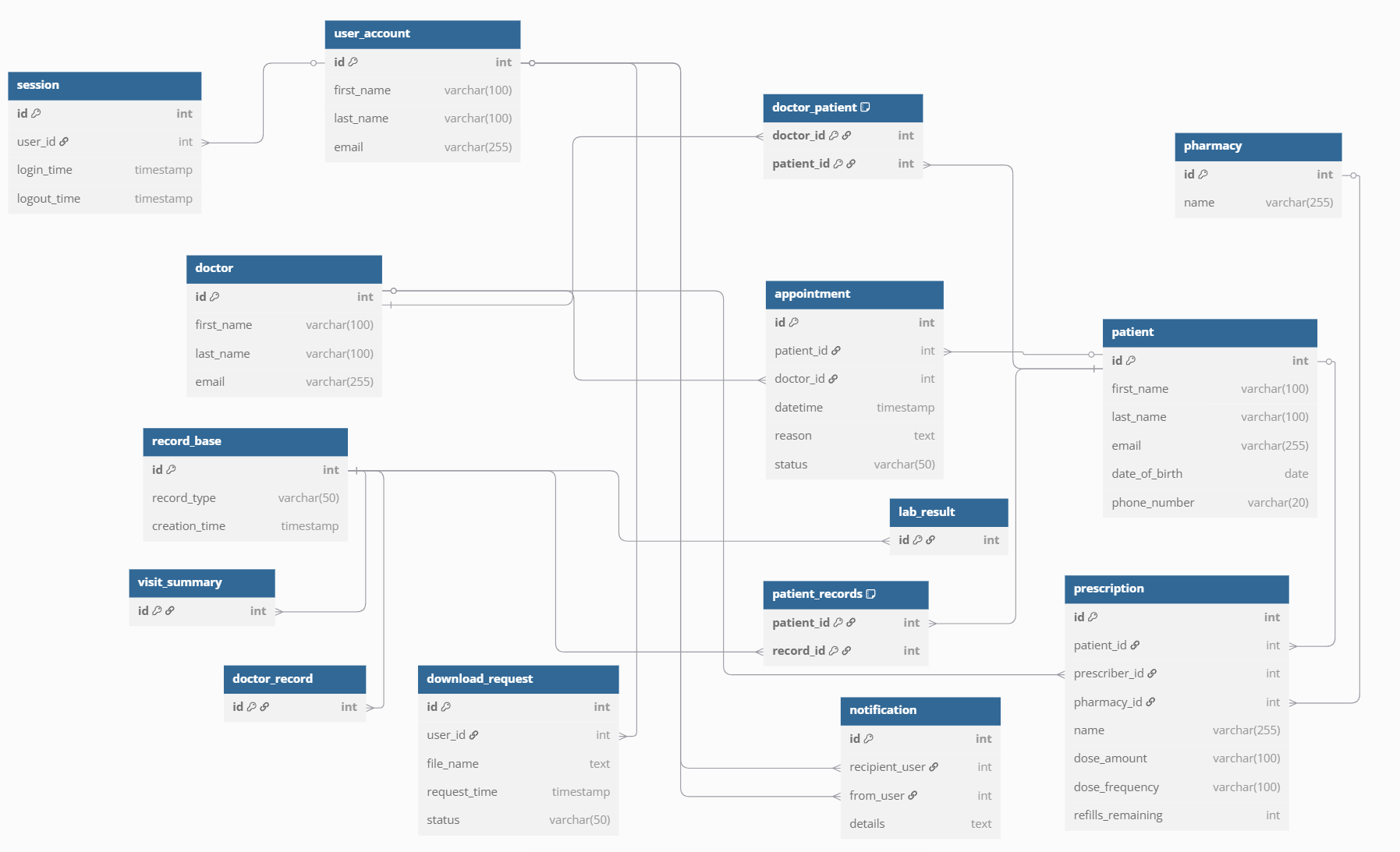
## User Family Account Screen

****

## Voice Assistant Screen

****

# Database Design (if applicable) — Mengliang Tan



# Security Design - Uzay Isin Alici

In this section, we describe the security design decisions implemented in the MyMedic application to protect sensitive health-related user data. Our approach ensures confidentiality, integrity, availability (CIA), and aligns with HIPAA regulatory requirements.

### 1. Security Principles

We follow widely accepted security design principles including:

* **Least Privilege**: Users can only access data relevant to their role (patient, family member, assistant).
* **Fail-safe Defaults**: Access is denied by default and granted only when explicitly allowed.
* **Complete Mediation**: Every request is verified using JWT tokens.
* **Defense in Depth**: Multiple layers of protection exist across frontend, backend, and infrastructure.
* **Separation of Duties**: Admin and user roles are strictly separated in both UI and logic.
* **Auditability**: All sensitive actions and login events are logged for future review.

### 2. Architecture Risk Analysis (STRIDE)

We identified potential security threats using the STRIDE framework and applied relevant mitigation techniques:

| **Threat** | **Example** | **Mitigation** |
| --- | --- | --- |
| Spoofing | Fake login attempts | JWT, bcrypt password hashing |
| Tampering | Modified data requests | HTTPS, input validation |
| Repudiation | Denied login activity | Secure logging |
| Information Disclosure | Exposed medical records | TLS, AES-256 encryption |
| Denial of Service | Brute-force attack | Rate limiting, lockout |
| Elevation of Privilege | Unauthorized admin access | RBAC, strict route  protection |

### 

### 3. Authentication and Authorization

Authentication is based on JWT (JSON Web Token). On login, the backend generates:

* a short-lived access token (15 minutes)
* a refresh token (valid for 7 days)

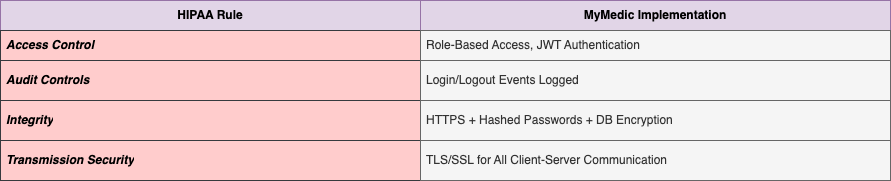
Passwords are securely hashed using **bcrypt**, and role-based access control (RBAC) ensures users access only appropriate resources.

### 4. Data Protection

* All communications are encrypted using HTTPS/TLS.
* Sensitive data such as medical records is encrypted at rest using SHA-256.

### 5. HIPAA Compliance

We applied HIPAA principles in the system as follows:



### 6. Application and Infrastructure Security

* The backend is containerized using Docker and exposed only via secure interfaces.
* The database is located in a private network and is not directly accessible.
* All secrets are stored using environment variables or secure vaults.
* Docker images are scanned for vulnerabilities regularly.

# Business Logic and/or Key Algorithms - John Gutierrez

This section provides the business logic and key algorithms supporting the user authentication and profile management features in the **My Medic application**. These features comply with **HIPAA regulations** and use secure technologies including **OAuth2** and **JWT** to protect sensitive patient data.

### User Registration (Sign-Up) Key Functional Modules and Algorithms

### 

### Pseudocode for validating and creating a new user:

### 

### *FUNCTION registerUser(name, email, password, role, confirmPassword):*

### *IF email is NOT valid THEN*

### *RETURN ERROR "Invalid email format"*

### *IF password != confirmPassword THEN*

### *RETURN ERROR "Passwords do not match"*

### *IF password length < 8 OR lacks numbers/symbols THEN*

### *RETURN ERROR "Password must be 8+ chars with letters, numbers, and symbols"*

### *IF role NOT in ["patient", "doctor", "nurse"] THEN*

### *RETURN ERROR "Invalid role"*

### 

### *IF database.userExists(email) THEN*

### *RETURN ERROR "Email already registered"*

### 

### *hashedPassword = bcrypt.hash(password, saltRounds=12)*

### *user = CREATE\_USER(name, email, hashedPassword, role)*

### *database.save(user)*

### 

### *LOG "New user registered: " + email*

### 

### *RETURN SUCCESS "Account created. Please log in."*

### 

### Visual Workflow and Sequence:

### 

### 

### 

### User Authentication (Sign-In with JWT) Key Functional Modules and Algorithms

### Pseudocode for authenticating a user and issuing JWT tokens:

### *FUNCTION loginUser(email, password):*

### *IF email is NOT valid THEN*

### *RETURN ERROR "Invalid email format"*

### 

### *user = database.getUserByEmail(email)*

### *IF user == NULL THEN*

### *RETURN ERROR "User not found"*

### 

### *IF NOT bcrypt.compare(password, user.hashedPassword) THEN*

### *LOG "Failed login attempt: " + email // HIPAA audit*

### *RETURN ERROR "Invalid credentials"*

### 

### *token = jwt.generate({*

### *userId: user.id,*

### *role: user.role,*

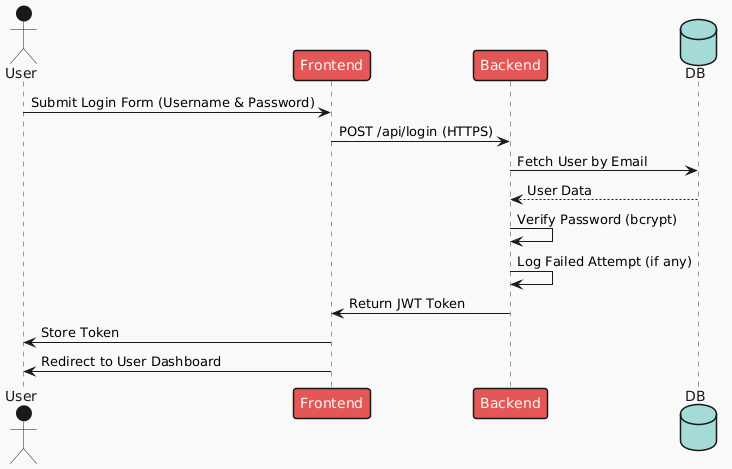
### *exp: currentTime + 15 minutes*

### *})*

### *LOG "User logged in: " + email*

### *RETURN SUCCESS token*

### Visual Workflow and Sequence:



### Profile Management (HIPAA Data Encryption) Key Functional Modules and Algorithms

### Pseudocode for updating a user's profile:

### *FUNCTION updateProfile(userId, profileData):*

### *IF !validatePhone(profileData.phone) THEN*

### *RETURN ERROR("Invalid phone number")*

### *END IF*

### *encryptedData = aes256.encrypt(profileData, key=SECRET\_KEY)*

### *database.updateProfile(userId, encryptedData)*

### 

### *LOG\_AUDIT("Profile updated by user: " + userId)*

### 

### *RETURN SUCCESS("Profile updated")*

### 

### *FUNCTION getProfile(userId):*

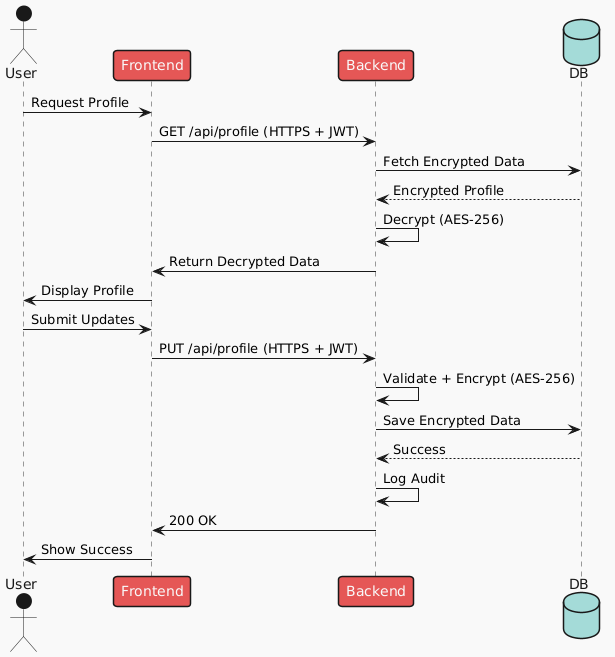
### *encryptedData = database.fetchProfile(userId)*

### *decryptedData = aes256.decrypt(encryptedData, key=SECRET\_KEY)*

### *RETURN decryptedData*

### Visual Workflow and Sequence:

### 



### My Medic enforces HIPAA practices through the following mechanisms:

### Passwords are hashed using Django’s PBKDF2 with SHA-256.

### JWT tokens are used for short-lived, stateless sessions.

### OAuth2 is used for authorization flow when integrating with external systems.

### Validation and sanitation protect against injection and CSRF attacks.

# Access is role-based and logged to support audit trails.

# Design Patterns - Tyler and Adriel

## Composite

The user class will use a composite design pattern to describe the relationship between patients in the same family, or a healthcare dependent of another user. For example, a mother might want easy access to their children's records. This is identified in the class diagram for general user relationships with the aggregation of patients as a field within the Patient class.

## Singletons

Singleton classes will be used for the manager type classes described in the class diagram to avoid concurrency issues writing data to the database.

## Decorators

Decorators will be used to format different types of data belonging to the same superclass on the UI. This will help mitigate duplicate code and make the code base easier to maintain.

## Facades

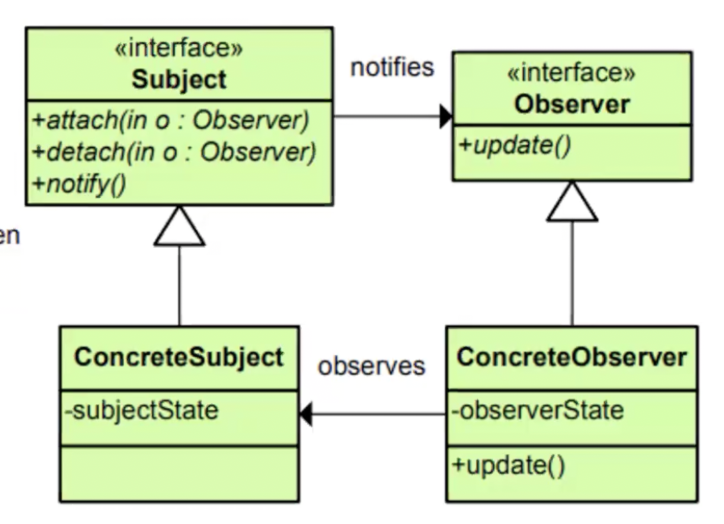
The manager classes will also function as facades to their corresponding components. This will simplify data flow between components and preserve low class coupling.

**Factory Method**

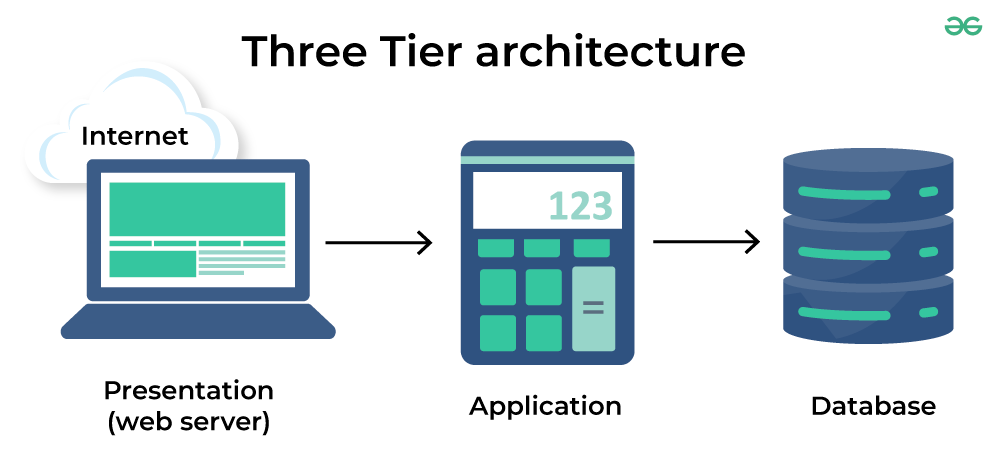
The Factory design pattern will be used to create patient, doctor and pharmacy objects. A factory for each object will be created. This will help simplify the backend programming and will also help with scalability as more and more users and medical establishments use our app.

**Observer Pattern**

The observer design pattern will be used to help implement the notification system. The design pattern will define a dependency between one to many objects so that when one object changes stat, all of its dependents are notified and updated automatically.



# Three Tier Architecture - Adriel



1. **Web Server**

The presentation tier is the user interface and communication layer of the application, where the end user interacts with the application. Its main purpose is to display information to and collect information from the user. For the MyMedic App, this will be the website that will be implemented using [Node.js](http://node.js). This is how the user will interact with the application. For example, this is where the user can input their user data or setup appointments. This is also where doctors can view their appointments or patient data.

1. **Application**

The application layer is the main data processing tier and is the center of the architecture and is where user inputs are processed. Additionally, this is also where the collected data from the web server tier is processed. Furthermore, the application tier can also add, delete, or modify data in the database. The application tier will be developed using Python and Django and will communicate with the web server tier using the REST API, and the database tier using the SQLite API.

1. **Database**

The database tier is where the information that is processed by the application is stored and managed. The MyMedic application will use Django built-in SQLITE3 for its database. This is where application data will be stored such as user id, patient records, addresses, appointment information, etc.

1. **Benefits**

**Encapsulation:** There is a logical and physical separation of function. Each tier can run on a separate operating system and server platform. Each can be customized and optimized without affecting the others.

**Faster development**: Because each tier can be developed simultaneously in parallel by different members of the team. We are able to develop and release our application faster. Furthermore, each tier can use the latest and best languages and tools since they are independent from each other.  
**Improved scalability**: Any tier can be scaled independently of the others as needed.  
**Improved reliability**: An outage in one tier is less likely to impact the availability or performance of the other tiers.  
**Improved security**: Because the web server tier and database tier can't communicate directly, a well-designed application tier can function as an internal firewall, preventing SQL injections and other malicious exploits.

# REST API - Hongcheng Ding and Adriel

REST APIs use web services to communicate between client applications and server-side software.

Web Services consist of three main components:

* URL addressable resources
* Uniform Resource Identifiers (URI)
* Hypertext Transfer Protocol (HTTPS)

1. **REST Principles**

**Client-Server Architecture**: The REST API will be used as the main bidirectional communication between the client (front-end) and server (back-end).

**Uniform Interface:** MyMedic is a web application, but the REST API can be used for any platform including mobile apps. This will help with scalability if MyMedic were to be expanded to other platforms. All devices use internet protocols based on TCP/IP

**Statelessness:** The server does not have to remember the client’s state. All the client’s requests must be “stateless,” so each request must include all necessary information (such as the client’s authentication details).

**Cacheability:** REST servers can cache data and reuse it for other requests in the future.

**Layered System:** REST APIs may have multiple intermediary layers between the client and the server. However, the client does not have to know these implementation details.

1. **Compatibility**

Rest is compatible with several file types such as JSON, YAML and XML

1. **Rest API Calls**

**GET Request :** Returns requested resource in a specified format

**PUT Request:** Updates specified resource

**POST Request:** Adds or updates information on the web service endpoints

**DELETE Request:** Removes data from the server

An API key will be used for authentication and to control access of resources

1. **Django Rest Framework**

The Django Rest Framework will be the framework used for the MyMedic application.

[Django Rest Framework Website](https://www.django-rest-framework.org)

1. **REST API Endpoint Design**

**5.1 User Registration**

Method: POST

Endpoint: /api/register/

Description:

This endpoint allows new users to register by providing a name, email, password, and role (e.g., patient).

**Request Body Example**:

{

"name": "Alice",

"email": "alice@example.com",

"password": "MySecurePwd123!",

"role": "patient"

}

**Success Response**:

{

"status": "success",

"message": "User registered",

"data\_received": {

"name": "Alice",

"email": "alice@example.com",

"password": "MySecurePwd123!",

"role": "patient"

}

}

# Any Additional Topics you would like to include.

# AI usage Log

You are allowed and even encouraged to use AI tools to help you generate the project idea, plan it and build it, but you need to clearly describe 1) What tools were used? 2) for what specific tasks and 3) Is it helpful? 4) how did you evaluate or modify AI-generated content? Additionally, you should submit the exported AI chat history as an appendix or share that with the instructor and facilitators.

| Tools | Who | Tasks | helpful | Evaluation/modification | links |
| --- | --- | --- | --- | --- | --- |
| Github Copilot | Tyler | Generate django code | yes | Modified syntax, code format, added documentation of functions and methods |  |
| GitHub CoPilot | Adriel | Appointment DB Testing | yes | Edited code for PEP8 compliance |  |
| GitHub CoPilot | Adriel | Create HTML and CSS files for viewing appointments | yes | Edited code to work with other existing code |  |
| GitHub CoPilot | Adriel | Create [models.py](http://models.py) for appointment app | yes | changed datetime and status to text |  |
| GitHub CoPilot | Adriel | Create pytest for viewing appointments feature | yes | Datetime spelling and appointment\_api url inserted |  |

# 

# References

**REST API**

<https://www.integrate.io/blog/an-introduction-to-rest-api-with-python/>

<https://www.integrate.io/blog/why-you-need-a-rest-api/>

**Three Tier Architecture**

<https://www.ibm.com/think/topics/three-tier-architecture>

# Glossary