10Cent

LOGiT
Software Architecture Document

Version <1.7>

Revision History

Date	Version	Description	Author
24/07/2024	1.1	Fill out section *1. Introduction* and *2. Architectural Goals and Constraints.*	Nguyễn Tấn Hoàng
24/07/2024	1.2	Import use-case model diagram	Võ Minh Khôi
25/07/2024	1.3	Fill out summary for section 4. Logical View	Võ Minh Khôi
25/07/2024	1.4	Fill out section 4.2 ViewModel components	Lê Nguyễn Minh Châu
27/07/2024	1.5	Fill out the View components	Lê Ngọc Thảo
27/07/2024	1.6	Fill out the Model components	Nhâm Đức Huy
28/07/2024	1.7	Overall document format	Lê Ngọc Thảo, Võ Minh Khôi

Table of Contents

Software Architecture Document

1. Introduction

1.1. Purpose

The purpose of this document is to provide a comprehensive overview of the software architecture for LOGiT. It is intended to capture and convey the significant architectural decisions which have been made on the system.

1.2. Scope

This document provides a comprehensive view of the LOGiT medical application as well as its own external and integral functionalities. Additionally, several architectural views were used specifically to describe different aspects of the structure. Nonetheless, a list of goals and final requirements are depicted detailedly.

1.3 Definitions, Acronyms and Abbreviations

No.	Definitions/Acronyms/Abbreviations	Explanation
1	MVVM	Model-View-ViewModel

1.4 References

- Software Architecture Document template
- LOGiT's Vision Document
- LOGiT's Use Case Specification Document

1.5 Overview

- Introduction: the document provides a brief entry to LOGiT's architect document
- Architectural Goals and Constraints: list out the high-level goals and requirements that affect the structure of LOGiT's application
- Use-case Model: states the use case and scenarios from different stakeholder's views
- Logical View: presents the detailed arrangement of LOGiT's system and illustrating the connection of the components and their shared responsibility for achieving goals
- Deployment: showcases the actual implementation of software components onto physical hardware
- Implementation View:

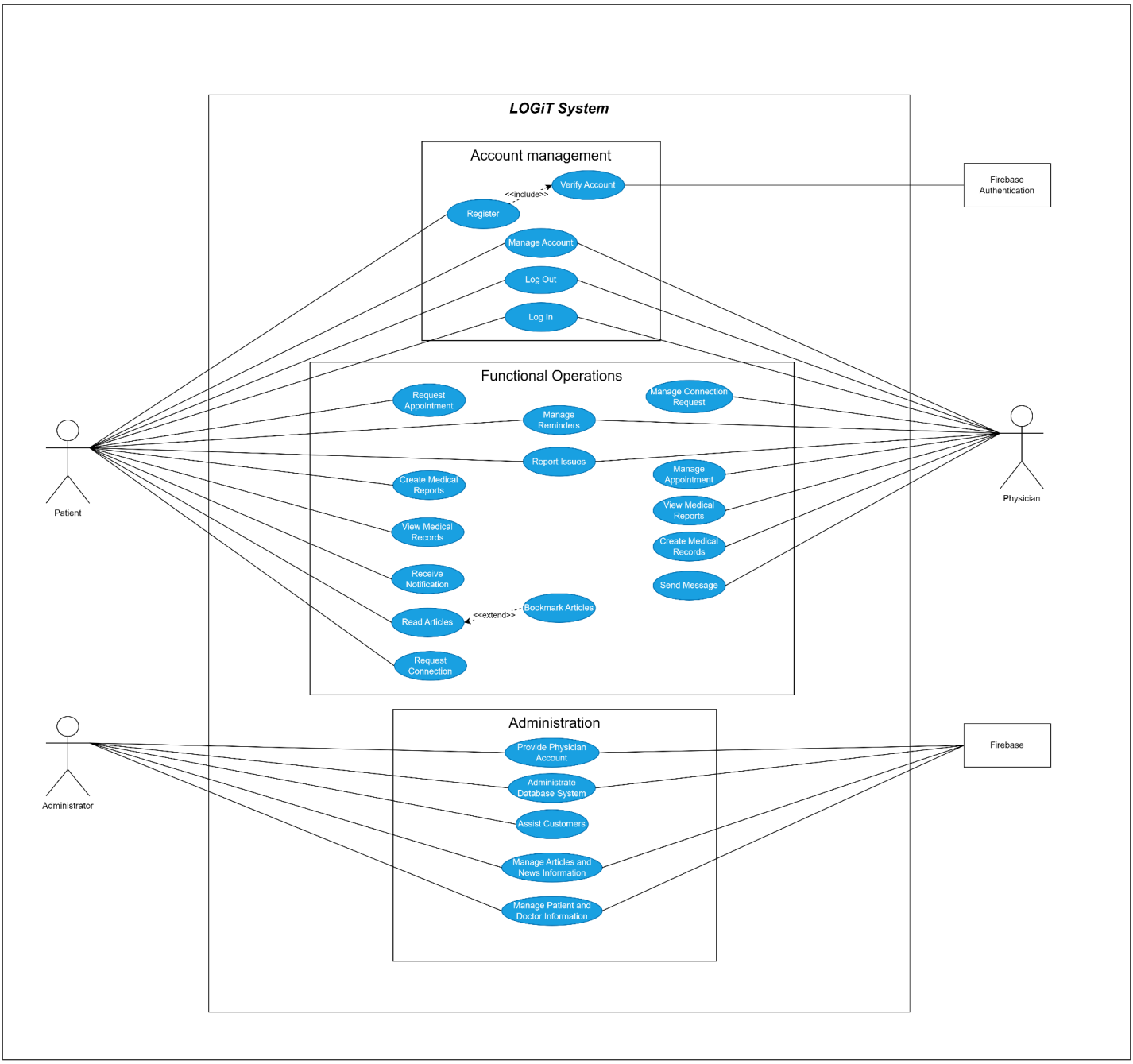
2. Architectural Goals and Constraints

- Programming language: The application must be built using Dart.
- For framework:
    - Front end: Flutter
    - Back end: Firebase
- For database: Firestore Database and Firebase Authentication
- Platform: Ensure the app runs seamlessly on both iOS and Android platforms to reach a broad user base.
- Performance:
    - Goals:
        - Ensure that user interactions (e.g., screen transitions, form submissions) occur with minimal delay, ideally under 200ms.
        - Optimize data retrieval and processing to provide real-time feedback where necessary, such as in the real-time health monitoring feature, instant messaging feature, etc.

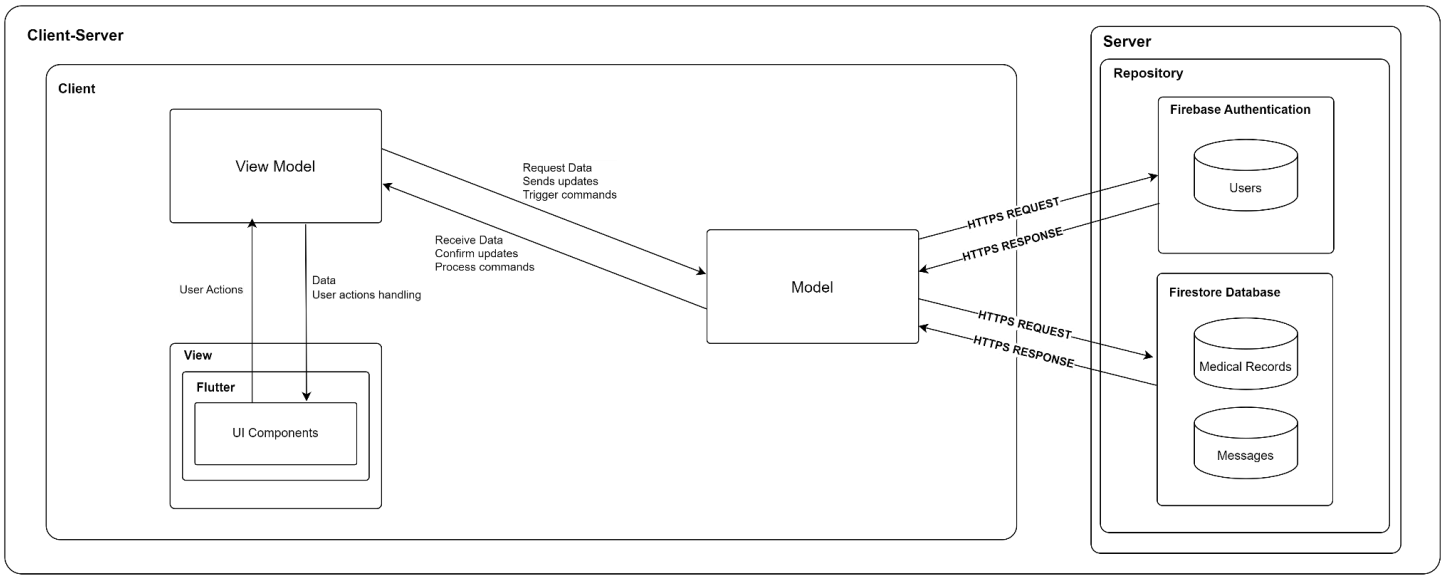
    - Constraints:
        - Design the app to handle varying network conditions, including slow or unstable connections.
        - Ensure the app can handle large volumes of medical data efficiently. Optimize data handling for scenarios involving high throughput.
- Security:
    - Goals:
        - Implement strong authentication mechanisms to protect user accounts. This should include support for multi-factor authentication (MFA) to add an extra layer of security.
    - Constraints:
        - Ensure compliance with relevant regulations and standards. Regularly review and update security practices to remain compliant with evolving regulations.

-

3. Use-Case Model



4. Logical View



The **MVVM** (Model-View-ViewModel) architecture is applied for the overall implementation of the LOGiT application. Extracting from the Logical View package diagram, the architecture consists of the following components:

Client-side:

- **View Package: UI Components**
  - Responsible for managing the user interface and its related components and screens for different user roles (patients, physicians, technicians).
  - Communicates with the ViewModel package to display relevant data and respond to user inputs.
- **ViewModel Package**:
  - Manages the application's data and logic.
  - Interacts with the Model to fetch and update data.
  - Receives and handles users' actions and sends data to the View for display.
  - Maintains the state of UI components, ensuring that the interface reflects the current data and state of the application.
- **Model Package**:
  - Houses the data entities and the business logic of the application, including patient records, appointments, user authentication, etc.
  - Interacts with the Repository to perform CRUD (Create-Read-Update-Delete) operations and handle user authentication.
  - Retrieves, manipulates, and validates data, ensuring data consistency and integrity.

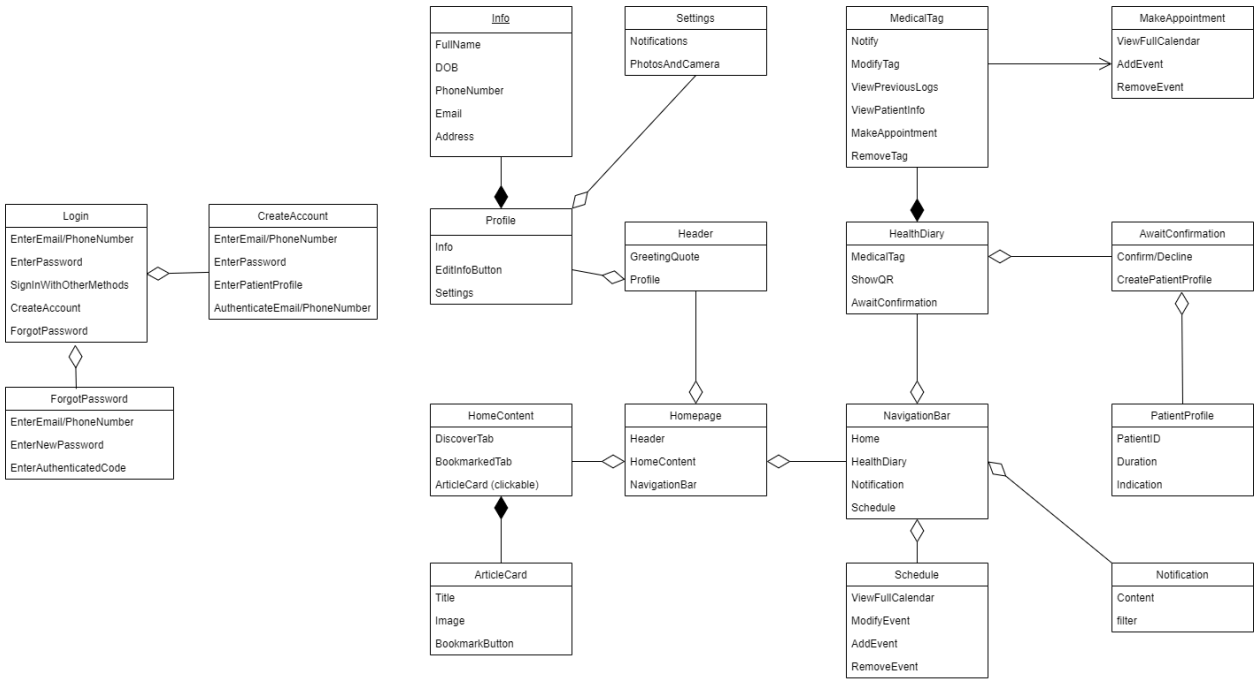
Server-side:

- **Firestore Database**:
  - Stores and synchronizes application data in real-time, providing a robust backend for data storage.
  - Handles data storage for patient records, appointment details, user profiles, and more.
- **Firebase Authentication**:
  - Manages user authentication, registration, and account management.

- Ensures secure login and registration processes for users.

4.1 Component: View

The provided component diagram offers a visual representation of the component structure within the application, depicting various components and their hierarchical relationships.



Key Components and Their Functions:

User Authentication Components:

1. Login Screen:
   - Components:
     - Username TextField: An input field for the user to enter their username.
     - Password TextField: An input field for the user to enter their password, with obfuscation for security.
     - Login Button: A button to submit the login form and trigger the authentication process.
     - Error Message Display: A text area to show error messages if authentication fails.
   - Functionality: Allows users to authenticate themselves by entering their credentials.
2. CreateAccount Screen:
   - Components:
     - Name TextField: An input field for the user to enter their name.
     - Email TextField: An input field for the user to enter their email address.
     - Password TextField: An input field for the user to create a password, with obfuscation for security.
     - Confirm Password TextField: An input field to confirm the password, ensuring they match.
     - Register Button: A button to submit the registration form.
     - Validation Message Display: Text areas to show validation messages for input errors.
   - Functionality: Enables new users to create an account by entering necessary details.
3. ForgotPassword Screen:
   - Components:

    - Email TextField: An input field for the user to enter their email address to receive password reset instructions.
    - Submit Button: A button to submit the email address for password reset.
- Functionality: Provides a way for users to reset their password if they forget it.

User Profile Components:

1. Profile Screen:
    - Components:
        - Avatar: An image view displaying the user's profile picture.
        - Name TextField: An input field for the user to edit their name.
        - Email TextField: An input field for the user to edit their email address.
        - EditInfoButton: A button to save changes made to the profile.
    - Functionality: Allows users to view and edit their profile information.
2. Info Component:
    - Functionality: Contains the basic profile details such as name and email, which are displayed and editable in the Profile screen.

Home Screen Components:

1. Homepage:
    - Components:
        - NavigationBar: A side menu for accessing different sections of the app.
        - Header: A header containing the title and navigation actions.
        - HomeContent: Displays a list or grid of items (e.g., recent activities, notifications).
    - Functionality: Serves as the main dashboard after successful login.

Health Diary Components:

1. MedicalTag: This component likely serves as a container for various tags or labels that can be applied to health-related data entries. These tags could represent categories like medications, allergies, medical conditions, or lifestyle factors.
2. ShowQR: This component potentially generates a QR code that encapsulates a user's health information. This could be a convenient way to share critical medical details in emergency situations.
3. HealthDiary: This core component appears to be where users input and track their health data. It might include fields for recording measurements (e.g., blood pressure, blood sugar), symptoms, medication usage, and other relevant health metrics.
4. ViewPreviousLogs: This likely enables users to review their past health data entries, potentially with filtering or search options for specific tags or date ranges.

Appointment Management Components:

1. MakeAppointment: This component presents the interface for users to schedule new appointments. It would likely include options for selecting appointment types, available time slots, and potentially providers or locations.
2. ViewFullCalendar: This component provides a visual calendar view, allowing users to see scheduled appointments within a broader timeframe. It might offer features like filtering by appointment type or adding reminders.
3. AddEvent, RemoveEvent, ModifyEvent: These components handle the actions of adding, removing, or editing appointments within the calendar.
4. AwaitConfirmation: This likely indicates a state where an appointment has been requested but is pending confirmation. It might display relevant details about the appointment and provide options for confirmation or cancellation.

Settings Components:

1. Settings Screen:
    - Components:
        - Toggle Switches: Switches for enabling or disabling settings.
        - DropDown Menus: Menus for selecting preferences (e.g., theme selection, notification preferences).
        - Save Button: A button to save the chosen settings.
    - Functionality: Allows users to customize their app experience by changing settings.
2. Notifications Component:
    - Functionality: Manages and displays notifications for various events and updates within the app.

Additional Components:

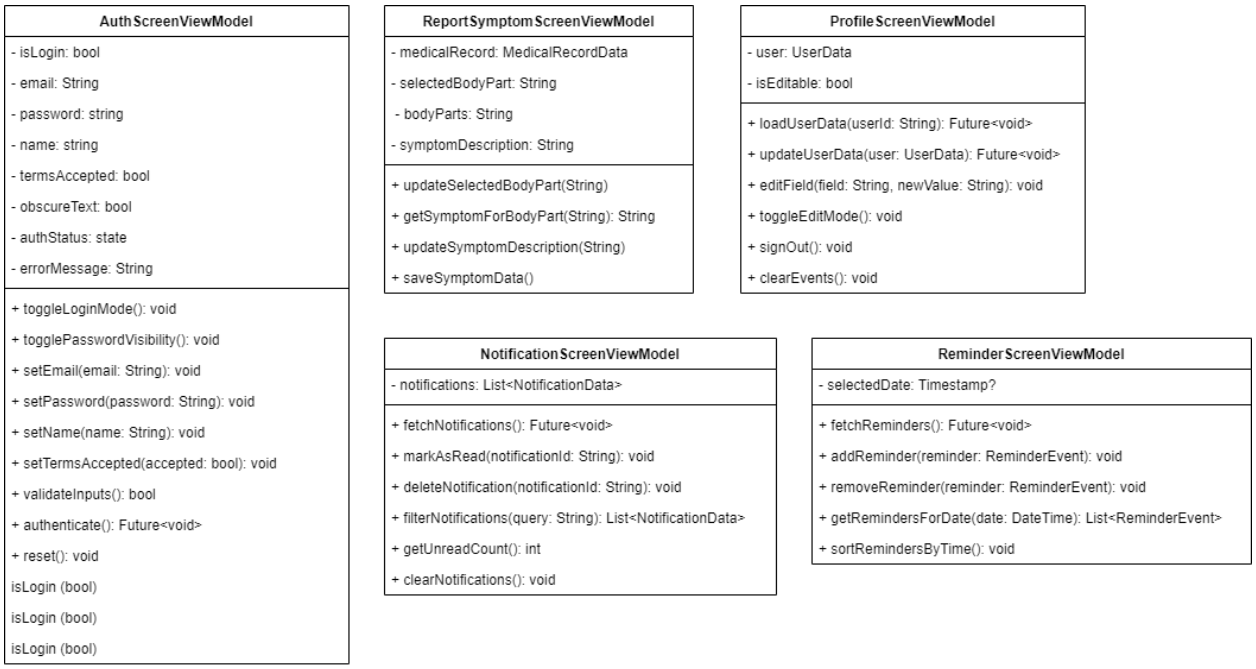
1. AwaitConfirmation Screen:
    - Functionality: Displays a confirmation screen for actions that require user verification.
2. PatientProfile Screen:
    - Functionality: Allows users to view and manage patient profiles.
3. DiscoverTab Component:
    - Functionality: Enables users to discover new content and features within the application.
4. BookmarkedTab Component:
    - Functionality: Displays content that has been bookmarked by the user for easy access.
5. ArticleCard Component:
    - Functionality: Displays individual articles in a card format for easy reading and interaction.

4.2 Component: View Model

View Model Classes responsibility:

- Encapsulates the business logic required for authentication, separating it from the UI logic, which allows the ViewModel to process inputs, perform validation, and execute authentication flows without UI dependencies.
- It acts as a bridge between the View and the Model, enabling data binding by exposing observable properties and methods that reflect the current state and user actions, ensuring that the View is automatically updated when data changes.

Classes Diagram:



Key Classes Explanation:

AuthScreenViewModel:

- The **"AuthScreenViewModel"** class in LOGiT is an important part located in the ViewModel section.
- It's in charge of key user authentication actions such as logging in and signing up.
- Maintains and manages the authentication state, including login/signup modes and any transient state related to the authentication process, such as loading or error states, to facilitate smooth transitions and user interactions.
- Validates user input to ensure compliance with authentication requirements, such as checking the format of email addresses and the strength of passwords, thus preventing invalid data from being sent to the Model or backend services.

ReportSymptomScreenViewModel:

- The **"ReportSymptomScreenViewModel"** class is a crucial component within the ViewModel section, designed for managing the symptom reporting process in a medical application.
- It handles the state and logic related to reporting symptoms, including tracking selected body parts

and their associated symptoms.
- Manages the symptom reporting data by providing methods to update and retrieve information about the body parts and symptoms, ensuring that the symptom descriptions are accurate and up-to-date.

NotificationScreenViewModel:

- The **"NotificationScreenViewModel"** is a key component of the ViewModel section, responsible for managing user notifications within the app.
- Manages a list of notifications, including fetching and storing them from a database, to ensure users are updated with the latest information.
- Processes notifications by marking them as read, deleting, and filtering based on search queries to provide users with relevant and updated content.

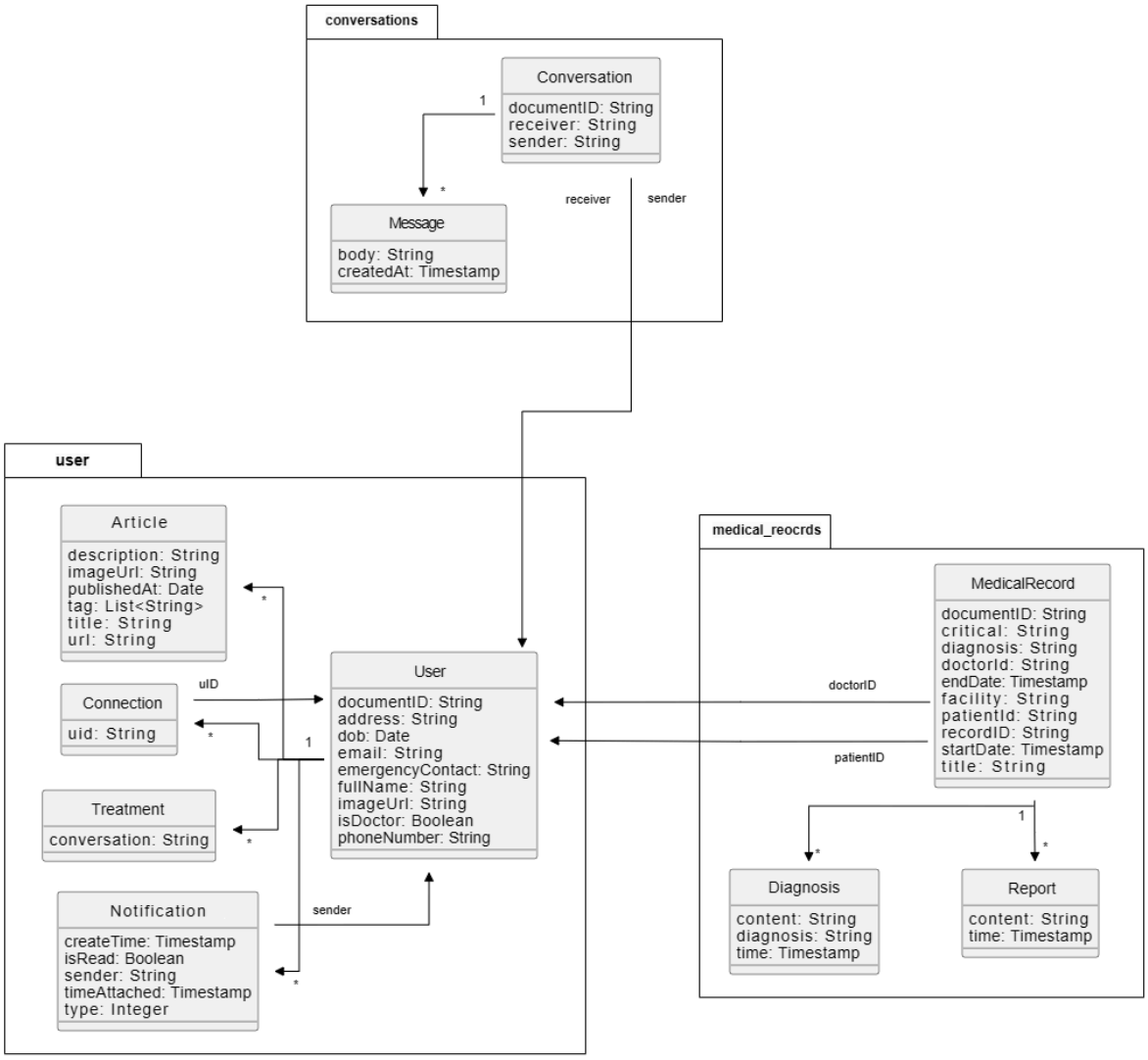
NotificationScreenViewModel:

- The **"ProfileScreenViewModel"** class is a key component of the ViewModel section, responsible for managing the user profile data within the app.
- It manages the user state, including loading user data, updating user information, and handling edit modes, to ensure a seamless user profile management experience.
- The class processes user input for profile updates, ensuring that changes are valid and updating the user data in the underlying data store.
- The class also handles user session management, including sign-out operations and clearing related session data, ensuring a secure user experience.

ReminderScreenViewModel:

- The **"ReminderScreenViewModel"** class in LOGiT is an important part located in the ViewModel section. It's in charge of managing reminder events for specific dates.
- Maintains and manages the reminder state, including selected date and the list of reminders, to facilitate smooth interactions and updates.
- Provides methods to fetch, add, and remove reminders, ensuring that the reminder list is kept up-to-date and accurate.

4.3 Component: Model



Key Classes Explanation:

User:

- Represents the users of the system.
- Manages personal information such as address, date of birth, email, emergency contact, full name, image URL, and phone number.
- Determines if the user is a doctor (`isDoctor` attribute).
- Collaborates with `Article`, `Connection`, `Notification`, `MedicalRecord`, `Conversation`, and `Message` to provide comprehensive user functionalities.
- Handles user authentication and profile management.
- Can interact with other users, manage connections, and receive notifications.

Article:

- Represents articles associated with users.
- Stores details such as description, image URL, published date, tags, title, and URL.
- Each article is linked to a specific user (`uID` attribute).

Connection:

- Represents connections between users.
- Stores user ID (`uid`) to indicate connections between users.
- Allows users to manage their network and interactions.

Notification:

- Represents notifications sent to users.
- Stores information such as creation time, read status, sender, attached time, and type.
- Allows users to stay informed about important updates and messages.

MedicalRecord:

- Represents medical records linked to users.
- Stores critical information, diagnosis, doctor ID, end date, facility, patient ID, record ID, start date, and title.
- Each medical record can have multiple diagnoses and reports.
- Links to users both as patients (`patientID`) and doctors (`doctorID`).

Diagnosis:

- Represents diagnoses within a medical record.
- Stores diagnosis content and time.
- Linked to a specific medical record.

Report:

- Represents reports within a medical record.
- Stores report content and time.
- Linked to a specific medical record.

Conversation:

- Represents conversations between users.
- Stores sender and receiver information.
- Each conversation can have multiple messages.

Message:

- Represents individual messages within a conversation.
- Stores message body and creation time.
- Linked to a specific conversation.

Relationships:

- **DocumentID** is a unique id every time a document of a model is created in firebase.
- **User and Article**: A user can have multiple articles.
- **User and Connection**: A user can have multiple connections.
- **User and Notification**: A user can receive multiple notifications.
- **User and MedicalRecord**: A user can be linked to multiple medical records, either as a patient or

doctor.

- **MedicalRecord and Diagnosis**: A medical record can have multiple diagnoses.
- **MedicalRecord and Report**: A medical record can have multiple reports.
- **User and Conversation**: A user can participate in multiple conversations.
- **Conversation and Message**: A conversation can contain multiple messages.

5. Deployment

[Leave this section blank for PA4.

In this section, describe how the system is deployed by mapping the components in Section 4 to machines running them. For example, your mobile app is running on a mobile device (Android, iOS, etc), your server runs all components on the server side including the database]

6. Implementation View

[Leave this section blank for PA4.

In this section, provide folder structures for your code for all components described in Section 4.]