CSC 431

MedSchedule

System Architecture Specification (SAS)

Team #1

Grace (Tianjiao) Gu	Developer
Isaac Attuah	Developer
Jin Curia	Scrum Master

1. Version History

Version	Date	Author(s)	Change Comments
1	3/31/21	Jin Curia, Isaac Attuah, Grace Gu	First draft
2	4/10/21	Jin Curia, Isaac Attuah, Grace Gu	Second draft
3	5/4/2021	Jin Curia, Isaac Attuah, Grace Gu	Added table of figures

2. Table of Contents

CSC 431

MedSchedule

3. Table of Figures

4. System Analysis

4.1. System Overview

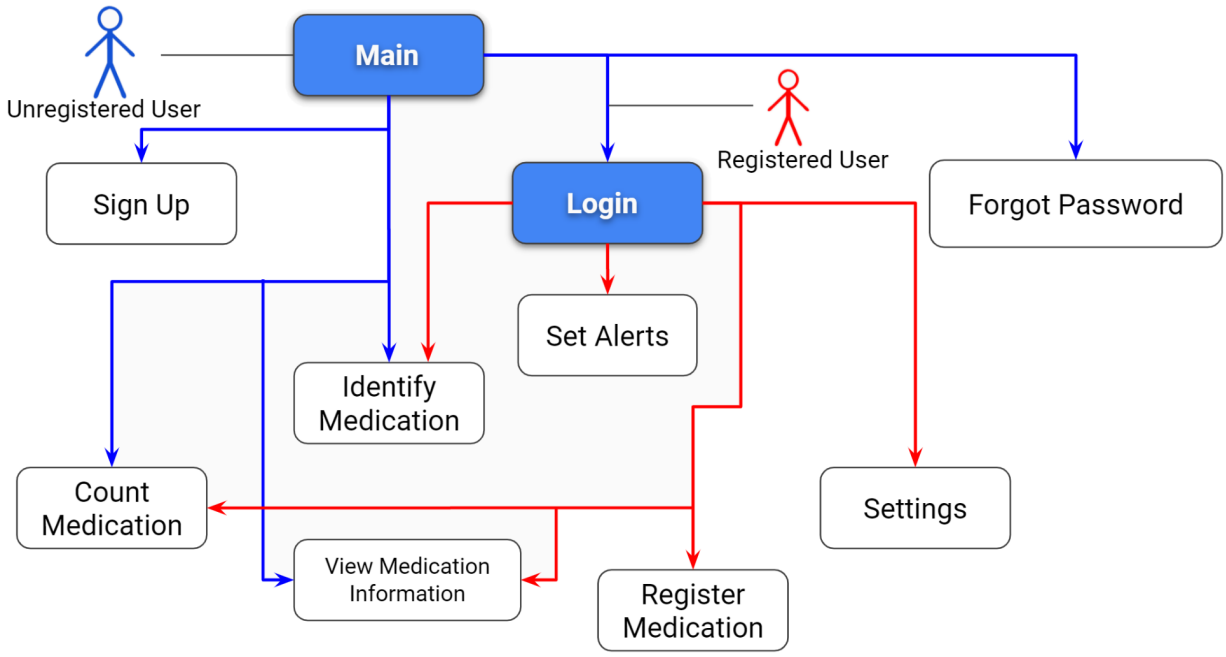
The composition of our system is focused around the following main parts: Setting and sending alerts, Register Medication, Identify Medication, Count Medication, and Settings. Setting alerts will use the user's local system information, since the implementation of this functionality utilizes the existing alarm system contained in all mobile phones.

The Settings information will be stored locally on the user's device in SQL format. The reason for this is so that the application can have faster loading times by using this cached information. Our database will also contain a backup of that information, for users that need to recover their account.

Medication Identifier / Counter / Register will be implemented for offline use, so our neural network will be stored on the user's device for feed-forward purposes only. Our application describes features that relate to services, so the style of our application architecture will follow Service-oriented architecture (SOA).

5. System Diagram(s)

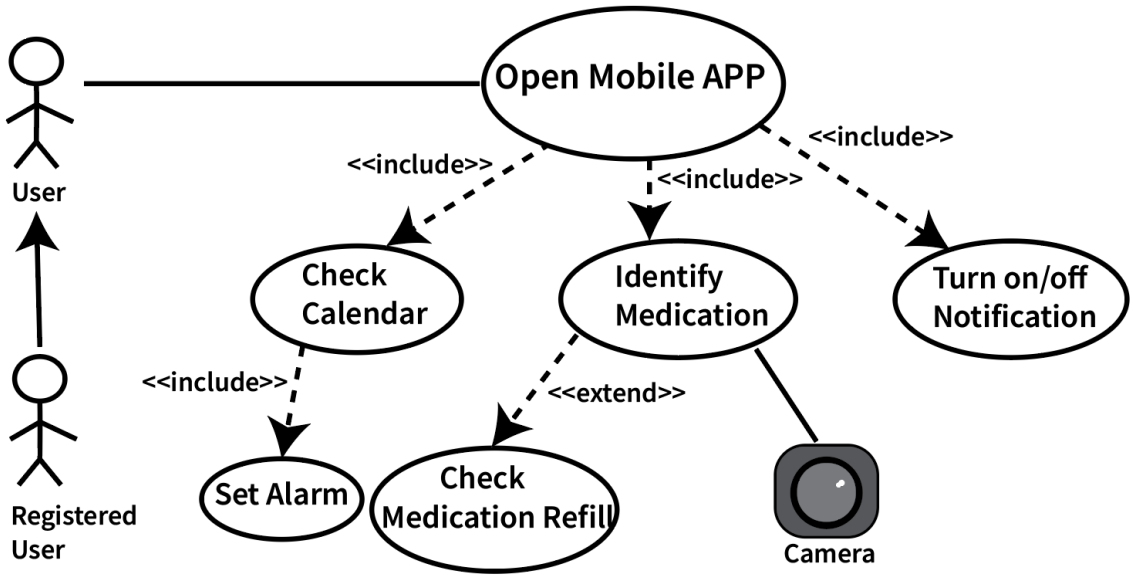
5.1. The following is an overview of the entire system in one diagram, incorporating all functional requirements.



System Diagram of Entire System

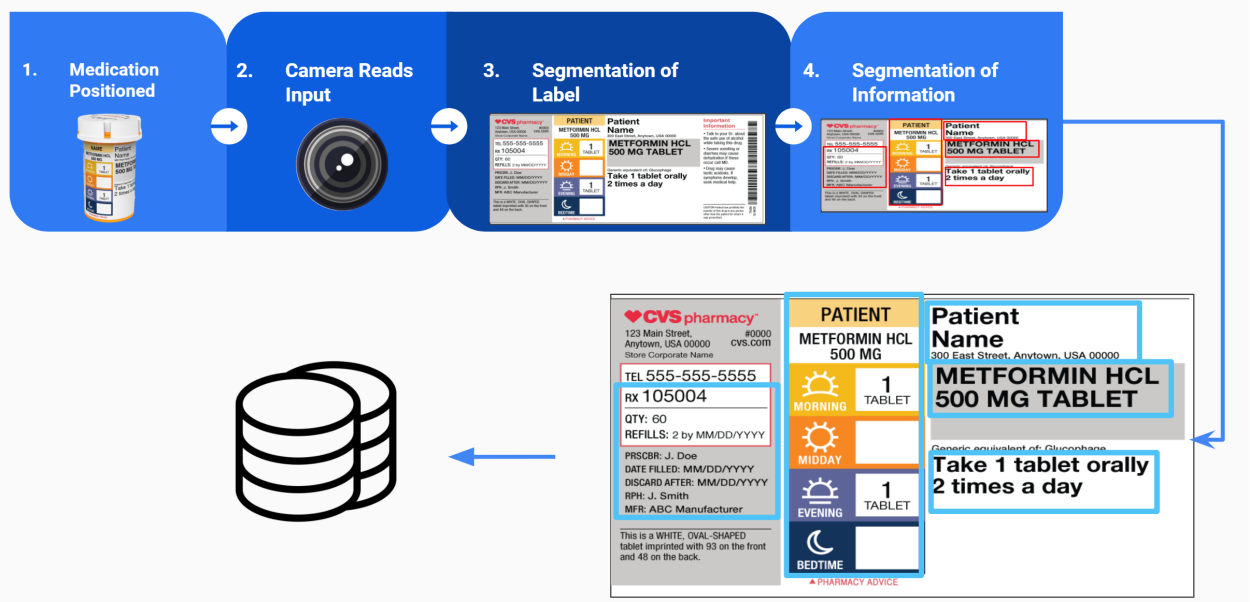
- Each node represents a functional requirement except for Main, which is the root.
- Main has edges to the features that are accessible without an account. Count medication, Identify Medication, and View Medication Information are base features that are accessible to anyone.
- Login has edges to all features (excluding Sign up, forgot password)
- Settings, Set Alerts, and Register Medication involve updating information that will be associated to an account, which is why they are accessible only through Login.

5.2. The following is our use case diagram that incorporates some functional requirements such as Identifying Medication, Check Calendar, Turn on/off alarms and set alarms, etc.



Use-Case Diagram

5.3. The following is our system diagram for Register Medication



Register Medication System Diagram

- The user positions their medication bottle and records it while rotating it
- The panoramic input is processed to segment the prescription label
- After the label is extracted, further segmentation is done to extract other relevant medical information
- The user can exclude some information before approving it to be permanently saved to their settings and to the server's settings (this is what the database stack picture represents in the figure)

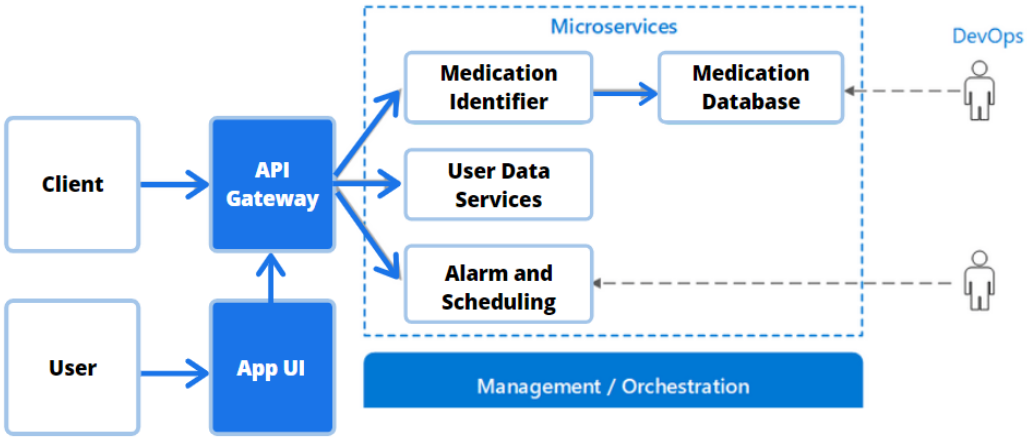
6. Actor Identification

- Patient: It is a user(human) type of actor. A user registers on the application's user interface to add medications to their calendars so that they can track their medications easily. Users will be able to receive alarms and reminders to take medicine and they can also turn off the alarms anytime. The user is our primary actor because he initiates the interaction with the system.
- Database Administrator: It is a system(external system) type of actor. It is a supporting actor(secondary actor) in a use case in an external actor that provides a medication database service to the system under design. It is supported by medication suppliers and pharmacies so that the requested information from users can be retrieved easily through the system.
- Time: It is a system clock. It is the time clock of mobiles which helps track users' medication schedule so that the calendar function can work well.
- Camera: It is an external system type of actor. It also comes with mobiles which helps to identify different medications.

7. Design Rationale

7.1. Architectural Style

Our design will utilize a Microservices architecture to ensure all features are developed independently but with concurrency in mind. This architecture model will enable us to render updates to individual microservices without having to redesign or redeploy the entire application. Additionally, a microservices-oriented approach will ensure scalability and maintainability across the board for all microservices.



Architecture Style Diagram

In our case, the main microservices will be the Medication Identifier, User Data Services and the Alarm and Scheduling microservice. Our developer operations team will be able to work directly with these services and maintain them from the back end on a more consistent basis without any service overlaps. The front-end of the application will interface with the API gateway to utilize various app features which are derived from their respective microservices. Additionally, we will provide permission-based microservice access to internal and external developer clients who would require our API for other applications or online services.

8. Design Pattern(s)

Our application's dependence on microservices will enable us to utilize an Event Sourcing design pattern. This design pattern will ensure that requisite changes are made across microservices when an event is triggered. For example: When it is close to a user's time to take their medication, an event is triggered to the calendar system which sends another event to the notification and alarm system to alert the user to take their medication. After the medication is

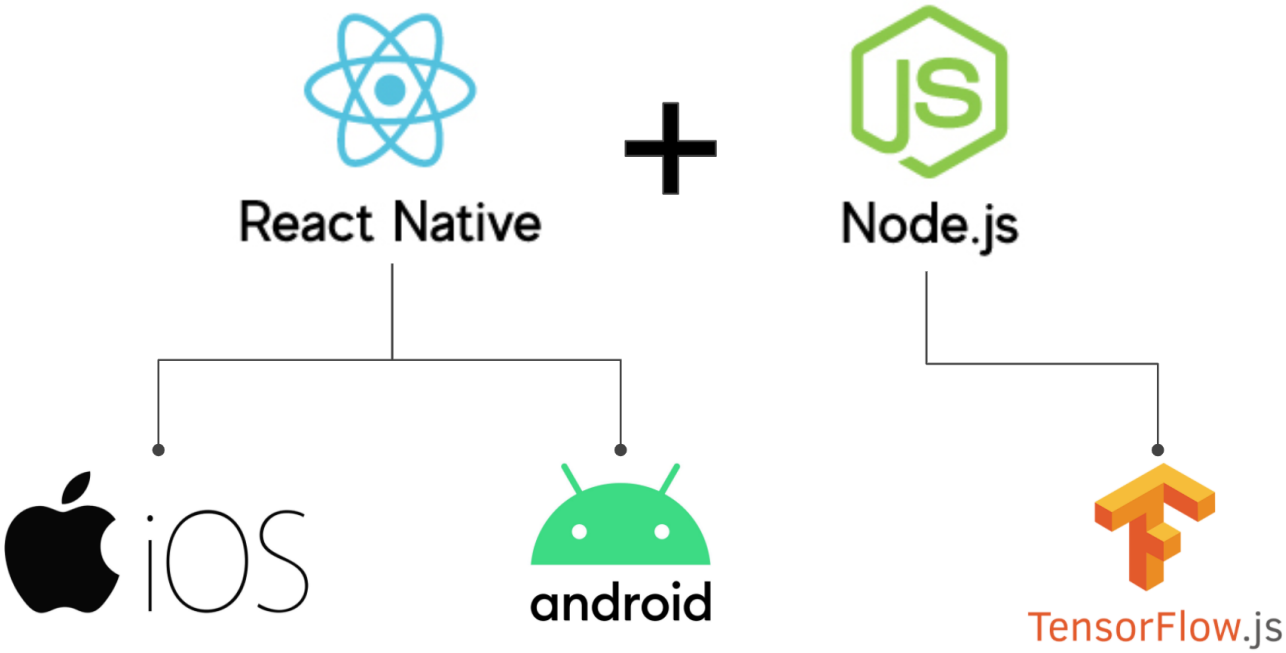
taken, and the user confirms their complete session, an event is sent to the calendar system to mark off the previous medication time.

9. Framework

Our system will utilize the React Native Framework for frontend components and Node.JS for backend. React Native supports cross-platform ability with a single codebase, which is one of our non-functional requirements. We decided to use Node.js since it uses an event-driven, non-blocking I/O model that it is lightweight and efficient. Furthermore, Node.js has a well-developed interface with Tensorflow.js, which will be used for the image classification tasks. Node.js also works efficiently with React Native and has more legacy support compared to other frameworks like Apache Cordova and Flutter.

The Node.js backend will also interface with a SQL database in order to run queries for the various microservices. For example, the Medication identifier microservice will execute the neural network algorithms responsible for image classification (using calls to TensorFlow.js) and then query an SQL database for pill matching; the database contains medications and their information which are hashed by (imprint, shape, color). Additionally, the user data services and alarm scheduling services will query information from their specific SQL databases.

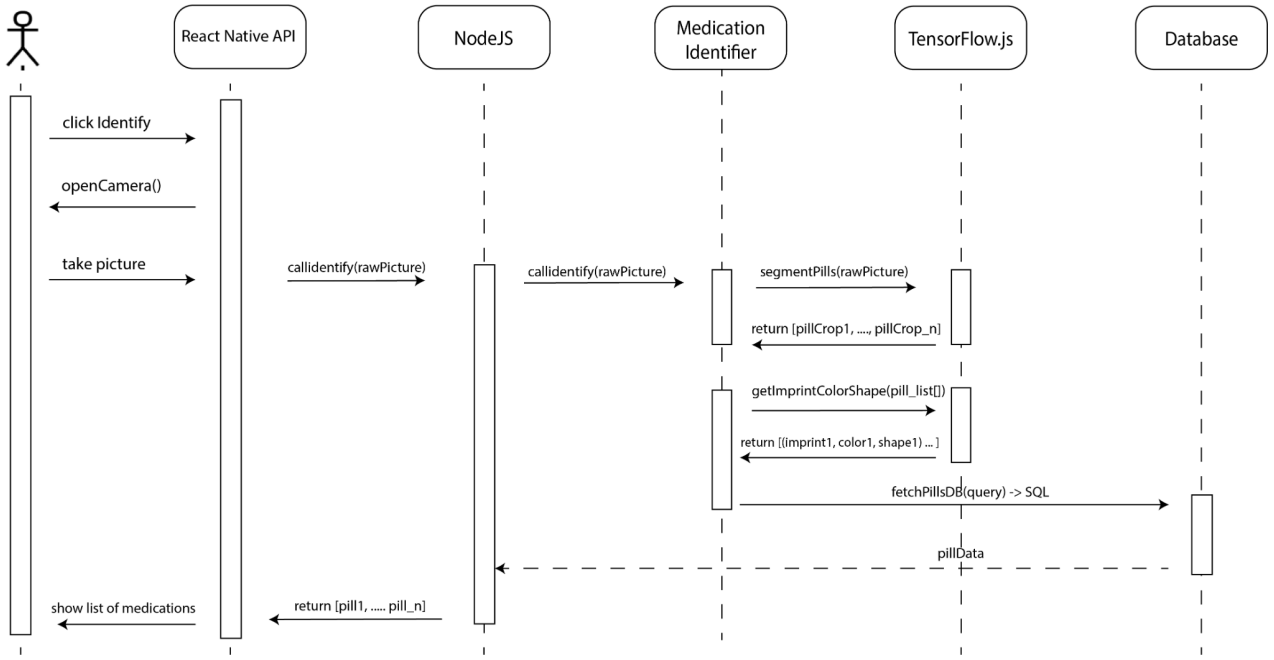
9.1. The following diagram is an overview of our framework.



Framework Diagram

10. Functional Design

10.1. The following is a sequence diagram for Medication Identifier.

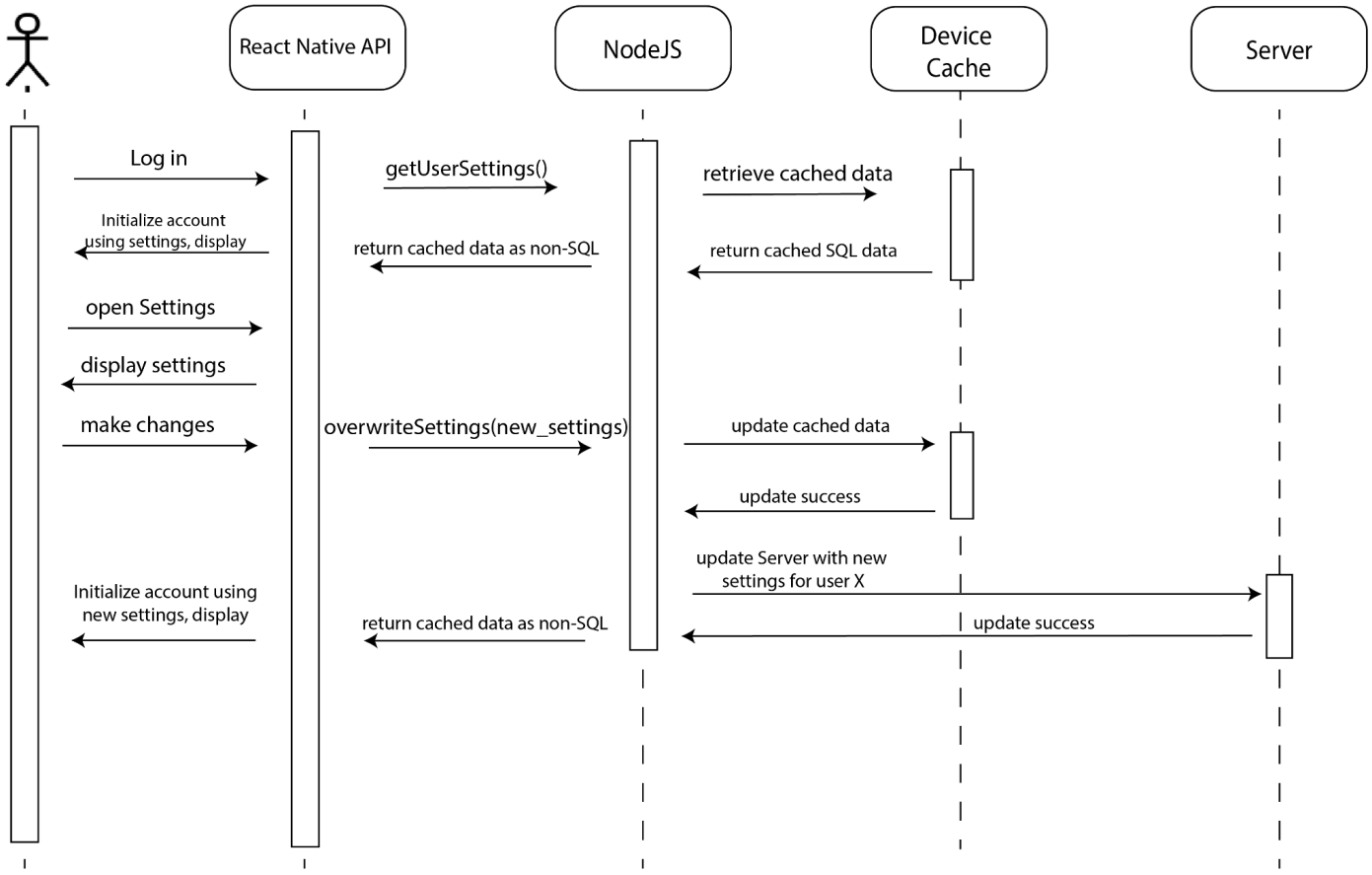


Medication Identifier Sequence Diagram

- When a user clicks Identify, the front end API (React Native) will open the camera on the user's device
- The user then takes a picture of the medication(s)
- The picture, represented as a 2x2 matrix is passed from the front-end to be handled by the back-end (NodeJS)
- Medication Identifier is our abstract representation of the class that handles the processing of picture input to medication information output
- The Medication Identifier will interface with TensorFlow.js and feed the raw image data to be classified.
- Internally, TF.js will utilize convolutional neural networks (CNNs)
- Different CNNs will be used for each image segmentation task because each segmentation task is related but different. It is simpler to use one CNN for extracting pill objects, one for extracting pill imprints, etc.
- Initially, the picture is cropped into individual pills, and returned as an array of cropped pictures.
- Then, each picture is passed to three different networks to classify: pill color, pill shape, pill code imprint.
- Once the results are received, a database that contains a table of medications (and their information) will be queried to find a match (pills are standardized in the US to be matched by shape, color, imprint).

- If matching is successful, the results are returned to the back-end and passed all the way back to the user.

10.2. The following is a sequence diagram for Update Settings

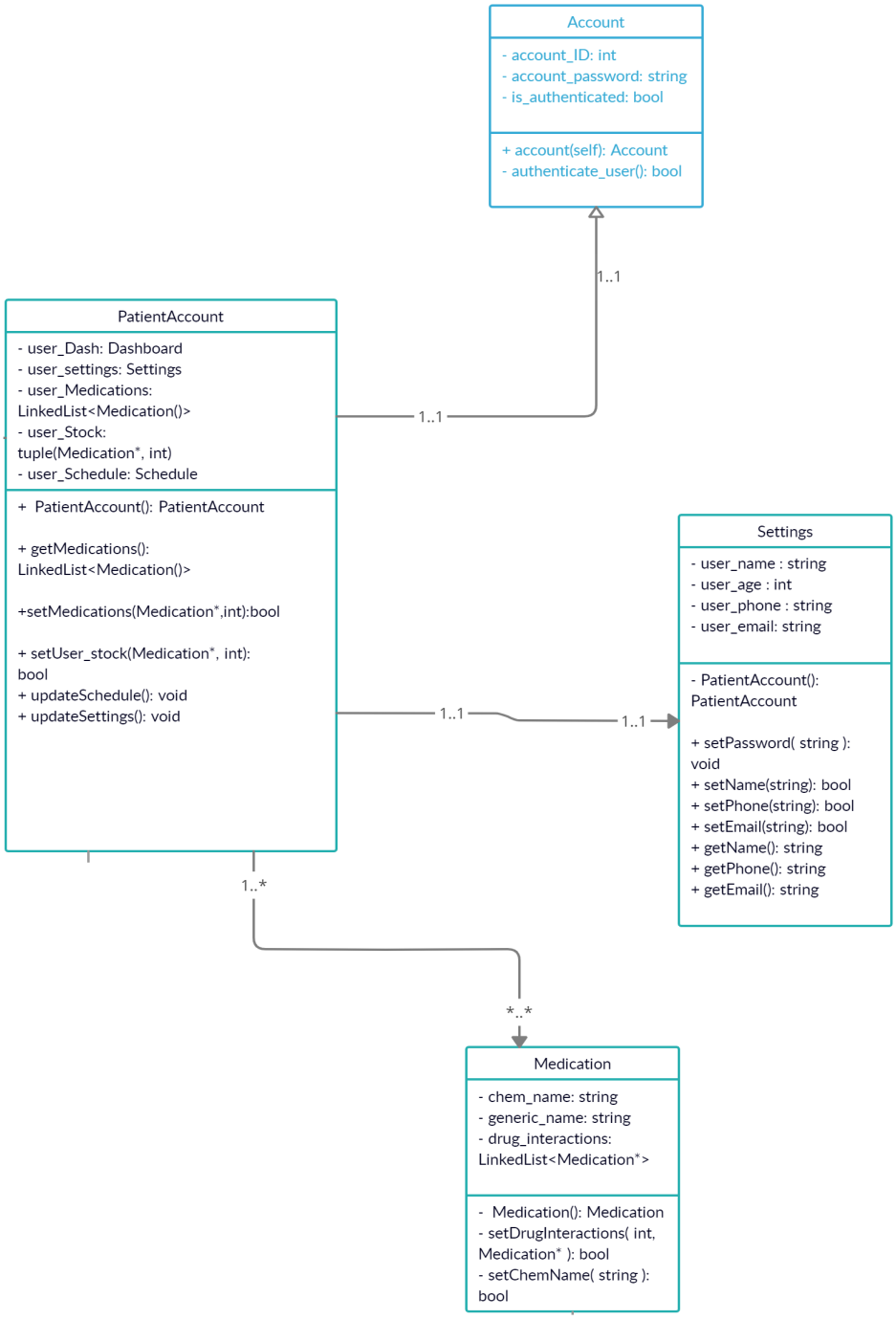


Update Settings Sequence Diagram

- After a user logs in, the front-end API displays the account using the appropriate settings (if it exists) by retrieving it from the device cache (alternatively, it may need to query the server)
- The account settings are retrieved by the backend API that has methods to get that information from the device cache. The backend API will also convert the SQL data to non-SQL data and pass it back to the front-end.
- Then, the account is displayed to the user, initialized by those settings.
- User makes changes to the settings, and these changes get passed to the back-end, then to the device to update the storage, and then to the Server to update that storage.
- The data is then converted, and the user account is displayed after initializing the new settings.

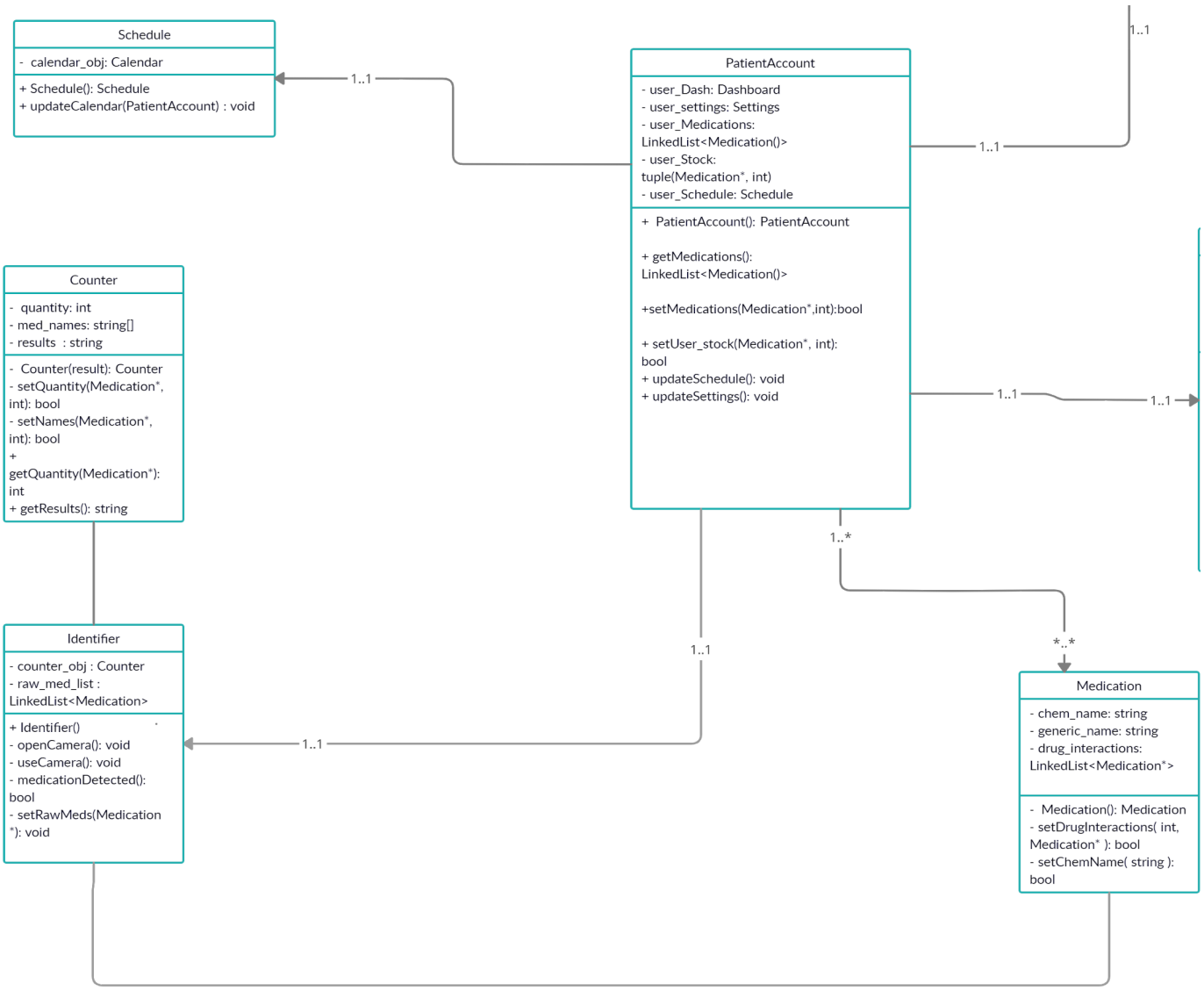
11. Structural Design

11.1. The following is a UML class diagram for the Patient Account class which inherits from the Account class and has associations to Medication and Settings classes.



Structural Design Diagram

11.2. The following is a UML class diagram showing the associations of the PatientAccount class to Schedule, Identifier, and Counter.



UML Class Diagram

- As shown in the System Overview, setting alerts (Schedule) are assumed to be requested by users with registered accounts (patients).
- Not shown here, but explained in System Overview, a user without an account can still use Counter and Identifier.
- The Identifier class will likely be split into different classes, one for generating queries for the SQL database, and the other for handling all non-SQL data. Thus, it is not shown how the SQL database interacts with these components in this diagram. But this diagram abstractly describes the classes' methods and fields.