High Level Design

Smart Health Consulting

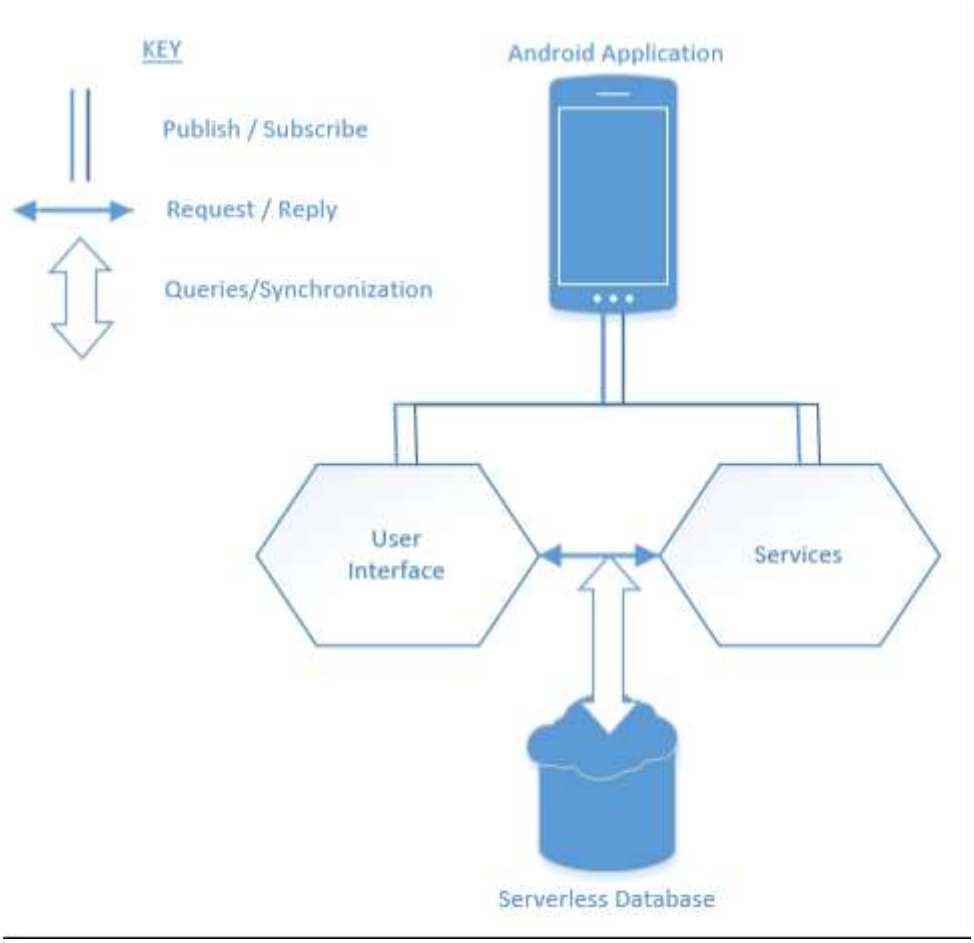
Team Members:

- Mario Osborn
- Sebastian De La Cruz
- Meenakshi Karthikeya
- Jonathon Rice
- Luke Myers
- Austin Peace

Contents of this Document

- High-Level Architecture
- System Interface Design
- Design Issues

High-level Architecture



This is a high level diagram of our system. The major components are:

- Android Application
    - The system will be a mobile application, running on Android OS. The development of this application will take place on the Android Studio SDK in Java.
    - The two main components of the Android app are as follows:
        - User Interface
          This consists of everything the client (or user) interacts with on the application. It hosts all of the GUI and handles all user interaction.

        - Services
          This component handles all background processes and every process that does not require a UI. It contains all of the backend logic to make the application work, and handles communication with the UI and database.

- Serverless Database
    - The system is employing the use of a serverless real time database called Firebase. It is a NoSQL database that stores and syncs all data necessary for the application. It interacts with the Services part of the Android application to answer queries or synchronize data.

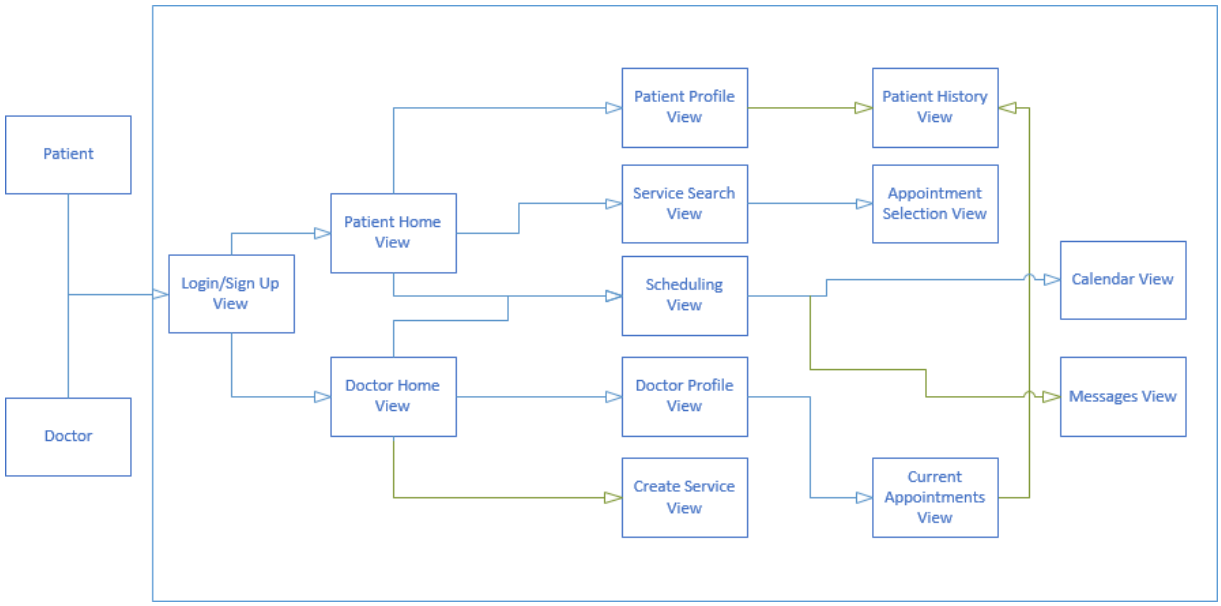
The types of architectural styles that were used in the formation of the high-level design are:

- *Client-Server*
    - This style employs two components, the client and server. It shows clients accessing server components by using a request/reply protocol. This architecture was used as our design has a clear distinction in client and server components.

- *Publish-Subscribe*
    - This style focuses on components interacting by broadcasting and reacting to events. This style was employed as the overall Android application development focus will be on strong support for evolution and customization. The components will be easy to reuse in system-wide events and will also have a shared repository in the form of a serverless DB so that all components share persistent data.

- *Repository*
    - This architectural style is made up of a central data storage, and a collection of components that operate on that storage to store, retrieve, and update information. This style was the easy choice to incorporate due to the serverless real time database portion of the system. Data is constantly synchronized with the application whenever there is a change, and queries are instantaneous.

System Interface Design



Our system is expecting to have two types of users, Doctors and Patients.

View	User	Action	Details
Login/Sign up	Doctor, Patient	Access	This view will have a username and password UI, to allow users to sign into their accounts (or make a new one), which will be fetched from the Database via Services.
Patient Home	Patient	Access	This view will have an icon UI for the user to select to get to other Patient views.
Doctor Home	Doctor	Access	This view will have an icon UI for the user to select to get to another Doctor views.
Patient Profile	Patient	Edit	This view will have all patient profile information and a icon to reach their patient history view. This view will allow the user to upload a profile photo or change their email via

			Services and the Database.
Service Search	Patient	Access	This view will have all the current services being offered, and ways for the user to filter and search through them. The services will be queried via Services and the Database.
Scheduling	Doctor, Patient	Access	This view will allow the users to select the calendar or messages view.
Doctor Profile	Doctor	Edit	This view will have all doctor profile information and a icon to reach their current appointments view. This view will allow the user to upload a profile photo or change their email via Services and the Database.
Create Service	Doctor	Edit	This view will allow doctors to create their services, and add costs and appointment times. It will then store this data via Services in the Database.
Patient History	Doctor, Patient	Access, Edit	This view allows the patient user to enter in brief medical history, and it is stored in the Database via Services. The doctor user can access this view if given permission by the patient user after accepting an appointment.
Appointment Selection	Patient	Access	This view allows patients to review the appointment they've chosen and send a request for it to the doctor via Services.
Current Appointments	Doctor	Access	This view allows the doctor user to see all their current appointment requests and the option of accepting or denying them. After accepting a request, they are also potentially allowed to access that patient's Patient History view.
Calendar	Doctor, Patient	Access	This view shows a calendar UI for patients and doctors, with their accepted appointments.
Messaging	Doctor, Patient	Edit	This view is a messaging UI for patients and doctors who have appointments with each other to converse, which happens via Services and the Database.

Design Issues

Listed below are the major design issues relevant to this project:

Reliability

With a local cached database within the Firebase services on the phone intermittent disconnects to the online database shouldn't be a problem, the user will just be working with possibly outdated information. As internet connection returns all new information on the local cached database will be uploaded to the online database. All of this is done through the Firebase services it gives us much less work to do and creates a more reliable application versus alternative options.

Reusability

Since this code will have no practical application, outside of meeting class requirements, our main focus for writing our code will not be reusability. However, we will be maintaining design discipline - A.K.A, our classes and functions will do what their names suggests and our designs will be modular.

Maintainability

The modularity of the app is in its pages. Each page has its own specific functionality that does little if any other changes to other pages. This means that each page can be changed or worked on independent of the others. In code documentation and descriptive variable naming will also help create code that is easy to understand and therefore maintain. Each page will be handled within its own class as per typical Android applications making modularity and maintainability easy versus other architectures.

Testability

Unit testing will be done for some of the major classes; however some issues that we may run into is testing the system as a whole or as a functionality. Therefore, some of the testing that we do will have to be done manually.

Performance

Most areas of the app will be simple enough to be fast on most Android phones in which all minimum requirements are met. The pages which will have the most processing on the app will be the Calendar, Service Search, Messaging, Patient History pages. The performance hit will increase as more data needs to be displayed. Prototypes will be needed to find the most optimal way of processing each page.

Portability

The project will not be portable and will only work within the Android Operating system.

Security

Communication to the database will be secured by Firebase services. Users passwords will be secured using the Firebase Authentication SDK. Confidentiality will be maintained by Firebase authorization rules specific to each class of user. Integrity will be maintained by Firebase data validation rules with authorization rules in place to protect write access to the database.

Prototyping

There will be different prototypes made for each of the major components of the program. For example, we will use prototypes to understand the use of a database class or set of classes that will be used by the program to fulfill the GUI's requests.

Detailed Design

Smart Health Consulting

Team Members:

- Mario Osborn
- Sebastian De La Cruz
- Meenakshi Karthikeya
- Jonathon Rice
- Luke Myers
- Austin Peace

Contents of this Document

Design Issues
Detailed Design Information
Trace of Requirements to Design

Detailed Design Issues

Reliability

Because Firebase is an extremely ironed out and mature product, there is almost no risks involved. Because both Android and Firebase are Google products, Firebase can be easily integrated into our Android application.

The design reason for using Firebase is that, while it is a relatively new service, it is easy to implement with almost no effort on the program designer's part in addition to being close to free if not highly affordable ($20 or less) and highly reliable.

Reusability

Once again, we decided that since this application will have no value outside of class requirements, we decided that we would not make writing reusable software our priority. Instead, we will be using good principles of modulation and cohesion our aim.

Maintainability

Using Firebase allows us to maintain the back end from anywhere and allows for features like usage statistics that can help identify potential problems. Any module that

needs to be fixed or changed can be done easily by editing the class governing that module. Modules can also easily be added on by creating a class and giving a user a way to navigate to the module.

Testability

Unit testing will be used to ensure the quality of units or in this case classes. Android Studio offers a great tool for unit testing your applications. The Android Events, which are Java-based software that is used to service the requests of the "Frames" or GUI, will have unit test written for it. However, the integration of the Events and Frames will make it difficult to create Unit tests. Partially because both components are written in different languages, Events in java and Frames in an HTML like form. Therefore, some of the testing of integrating both components will be manual. Our understanding, as of now, is that it is easier this way. But we will continue to learn and adapt.

Performance

Since Firebase is a well-established product and capable of handling large amounts of users, volume and stress tests will not necessarily be an issue. However, depending on the hardware of the phone running the application, some pages with large amounts of data might load slower when compared to phones with superior hardware. As a result, we will need to create a set of configuration tests that cover all supported Android SDK versions. Due to time constraints and the fact that more development time has been placed on creating a finished, working product, we might not be able to support fast response times on phones that run the minimum supported software. Since this is the case, that is why we have been careful in setting Android Marshmallow 6.0 as the currently accepted minimum standard. We expect phones capable of running this version of Android to be able to load data at a reasonable pace. As the configuration tests are underway, we will view the response times and determine whether we can decrease the minimum supported SDK version or not.

Portability

The project will not be portable and will only work within the Android Operating system. This decision was based on the fact that this application will only be used for the purpose of this class and there will be no reason to run it on any other platform.

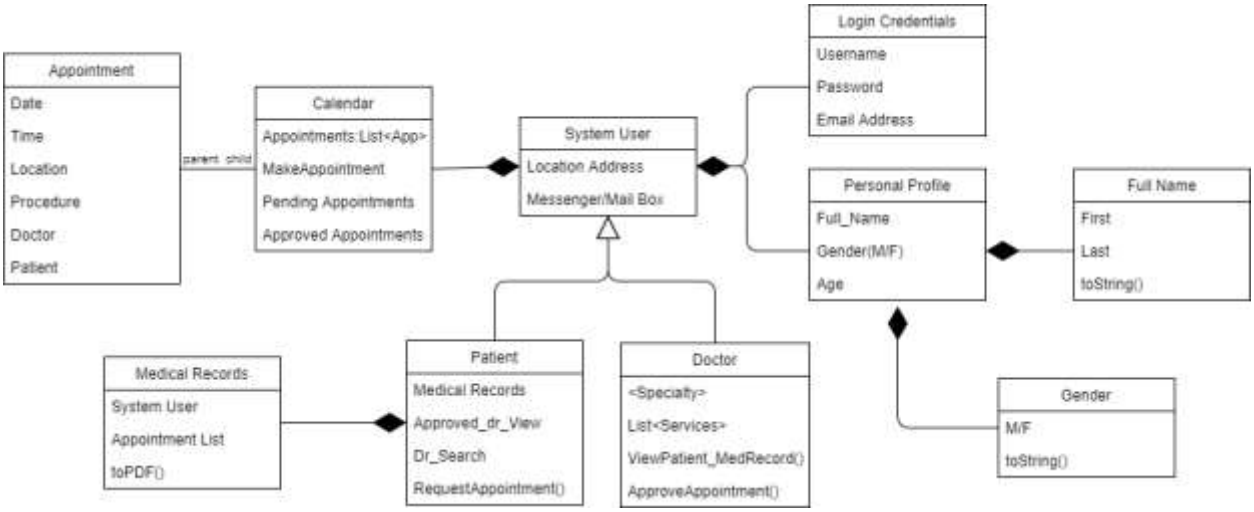
Security

Firebase provides a full set of tools for managing security and utilizes SSL (Secure Sockets Layer) and 2048-bit keys for certificates. The database follows the concepts of authentication, authorization, and data validation for securing information. This allows the system to identify users, control their access and prevent unwanted changes to information.

The user authentication is performed by using the industry standards of OAuth 2.0 and OpenID Connect. OpenID Connect handles user authentication by utilizing an identity provider to provide certificates of users. While OAuth deals with authentication by having the identity provider provide tokens for the users.

Detailed Design Information

The following class diagram lists most of the class structure that will be used for the project. However, there are some class structures that are missing. The Messenger/Mail Box class is mention, however it has not yet been elaborated on. The reason being that we will be working out this detail via prototyping to see what the best approach to the design is. The other class structure was very straight forward.



GUI Design

It is important to note that the above class-diagram only contains the logical and java-based portion of the project. The other portion is the GUI design.

Android applications are all composed of two basic structures: frames and events. Frames are the GUI component that deal with how the application will look, while events complete the more “back-side” process, serving the request brought on by the frames GUI-interface.

Frames is an HTML based structure that works with the graphical portion of the application, while Events are the more logical and java-based backbone. Both are needed to create a fully functioning android application. It is also worthy to note that, by having a division between the logic and the GUI, this logic (or java-based code) can be ported to a different platform that supports java, only requiring the GUI to be rewritten.

Trace of Requirements to Design

Requirement	Module
3.1.1 - 3.1.6, 3.2.2.3, 3.2.2.4, 3.8.9	Login Credentials
3.2.2.2, 3.4.4	Personal Profile
3.1.7, 3.2.1.6, 3.8.4, 3.8.5	System User
3.1.10 - 3.1.13, 3.2.1.1 - 3.2.1.5	Doctor
3.1.14 - 3.1.17, 3.1.19, 3.2.2.1, 3.4.5	Patient
3.1.18	Medical Records
3.1.8, 3.1.9	Calendar
3.8.2, 3.8.3	Appointment