Anthony Krivonos (ak4483)
Kenneth Chuen (kc3334)
Kevin Wong (hw2735)
William Pflueger (fwp2108)

October 21st, 2020
Adv. SWE Assignment T1
Prof. Gail Kaiser
Team: Starmen

Part 1

upmed is a telemedicine platform that encourages transparency between healthcare professionals. (HCPs including doctors, nurses and allied health workers) and their patients. On this platform, patients securely log into a portal where they may access their health records and lab results, schedule video appointments with their (HCPs) through embedded Zoom video conferencing function, and receive text notifications when lab results are ready or appointments are upcoming. *upmed* is targeted towards users during the pandemic who feel uneasy about going out for doctor's appointments and would prefer to see practitioners virtually.

To register, a user simply has to provide their basic credentials, address, profile picture, and past health history. The server returns a token that is saved in the browser and is valid for 24 hours, so the user, unless logged out explicitly, will stay logged in for that duration of time. This functionality can be easily demonstrated, as *upmed* is a web application that will be hosted on Firebase and requires no setup. On top of this, the patient portal that includes patient records and appointment schedule will also be easily shown via screen share. Finally, the text notifications feature can be demoed through a webcam, ensuring that this entire project may entirely be presented during class time.

The upmed backend will leverage Firebase as a nearly all-in-one suite of tools. Firebase Functions will be used to store API endpoints in the cloud. Firebase Analytics will gather statistical information on patient and doctor interactions with the platform. Firebase Hosting will host the GUI on an SSL-encrypted website. Finally, Firebase Cloud Firestore will be the no-SQL database used to store data. The data that is to be stored includes patient credentials, records on a per-appointment basis, test results, and an appointment schedule per user. Patient credentials, records, and test results will each be encrypted to ensure secure transfer of vital information on the platform. The Firebase API, which provides a Python 3 client, will be used in the backend for all the aforementioned services. In addition to this API, we will leverage the Zoom API to schedule appointments and the Twilio API to send text confirmations for appointments, all of which will be hooked up on our server. The frontend will utilize the endpoints on the server, calling the endpoints through a REST client, such as the built-in `fetch` package.

Part 2

Virtually consult healthcare providers (HCPs):

As a patient, I want to meet my HCP virtually so that I can obtain healthcare advice and consultation anywhere and free from COVID. My conditions of satisfaction are able to conduct video conferencing with my HCP.

Access patient's own health records:

As a patient, I want my health records to be kept securely with easy access so that I can take reference to them during my consultation. I should be able to input some health records such as blood pressure, glucose level and body weight. My conditions of satisfaction are able to review all of my records. Able to add and edit some of my own records at any time—before, during and after video consultation.

View, add, edit and delete patient's health records:

As a HCP, I want to view, add, delete, edit and modify my patients' healthcare records of vastly diverse types (lab results, radiology results, consultation notes). My conditions of satisfaction are able to conveniently view, add, update, edit and delete health records if I am assigned to my patient.

Keep a schedule of my appointments:

As a HCP, I need to well schedule appointments with all my patients so that I can meet my patients in an orderly manner and use my time efficiently. My conditions of satisfaction are able to use a calendar to schedule appointments and notify my patients about their appointments.

Part 3

Case 1: Virtually consult healthcare providers (HCPs):

We would use 2 devices—one for HCP and another for the patient. The HCP will initiate a video conference request to the patient. The patient will see the request and accept the request to start the consultation. The expected result to pass is a success to start a video chat with low latency, high resolution of video and audio chatting. A failed test includes failure to start the call despite accepting the request, poor video quality (in terms of high latency, excessive low resolution). A special case would be if the patient declined the video request, the HCP should receive the notification and cancel the video chat request accordingly. The HCP should then be able to leave a message to the patient for rescheduling.

Case 2: Access patient's own health records:

First, we would create patient data in multiple file formats: including lab results, radiology reports and consultation notes. The data will be added to the respective patient's profile. Then, we would log into the patient's account and attempt to view all lab records. Finally, we will attempt to add or edit some of the health records such as blood glucose, body weight, etc. Afterwards, this procedure will be repeated using an unauthorized account (such as another patient's account) to test out the authorization features. The expected result should be the ability to view data an account is authorized for, and unauthorized accounts being locked out of other patients' data. A failed test would be a failure to view the records using the first patient's account, or an unauthorized patient's account managing to view the first patient's information.

Case 3: View, Add, edit and delete patient's health records by HCP:

Several different formats of data would be prepared first as in the previous case for at least 2 different patient profiles including lab results, radiology reports, and consultation notes. They will be added to the respective patient's profile using the HCP's account to test the adding record function. Then, we would attempt to edit each record, such as changing the lab result to test the edit. Finally, we will delete the record to test the delete function. This procedure will be performed again using an unauthorized account (such as another patient's account or an unauthorized HCP's account) to test out the authorization features. The expected result should be successfully editing patient information when authorized. A failed test would be a failure to perform any of the functions using the HCP's account, or having the unauthorized account perform any of the aforementioned functions.

Case 4: Keep a schedule of my appointments:

The HCP will interact with a calendar to add an appointment, select a specific patient, write up details for the appointment, and select a time and date for the appointment prior to creating it with one click. Successful creation of an appointment will be reflected on both the HCP's and patient's calendars, where they can view all their appointments in a weekly or monthly calendar view. Failure to schedule an appointment due to an invalid patient, scheduling conflict, or incomplete information will not add the appointment to either actor's calendar.

Part 4

Individual Project Tool	Frontend Equivalent	Backend Equivalent
JDK	Node.js, TSC	Python 3, Flask
Eclipse	VScode	PyCharm
Maven	NPM	Pip
CheckStyle	Prettier	Pylint
JUnit	Mocha, Chai	unittest
Emma	Istanbul	Coverage.py
Spotbugs	SonarQube	SonarQube
SQLite	Firebase	Firebase

Kenneth and Anthony will be focused on developing the front-end in TypeScript using the React framework. The TypeScript will be compiled into interpretable JavaScript using the TSC compiler. The code will be developed on VScode and external libraries and packages will be imported via NPM. Prettier will be used for style checking and formatting. A combination of Mocha (for unit testing) and Chai (for assertions) will be used for unit testing, and Istanbul will be run to ensure code is covered. SonarQube will be used for checking bugs and the data will be stored on Firebase via the JavaScript/TypeScript Firebase client.

While Kenneth and Anthony are working on the frontend, Kevin and Pflueger will be working on the backend, which is a REST API created using Flask in the Python 3 language. The backend will be developed on PyCharm, importing packages using the generic Pip tool. Pylint will be used for style checking and the built-in unittest will be used for unit testing. Coverage.py

will be imported and run to ensure code coverage, and SonarQube will be used once again to spot bugs. Finally, the Python REST API will interface with the Firebase client to get and post data.