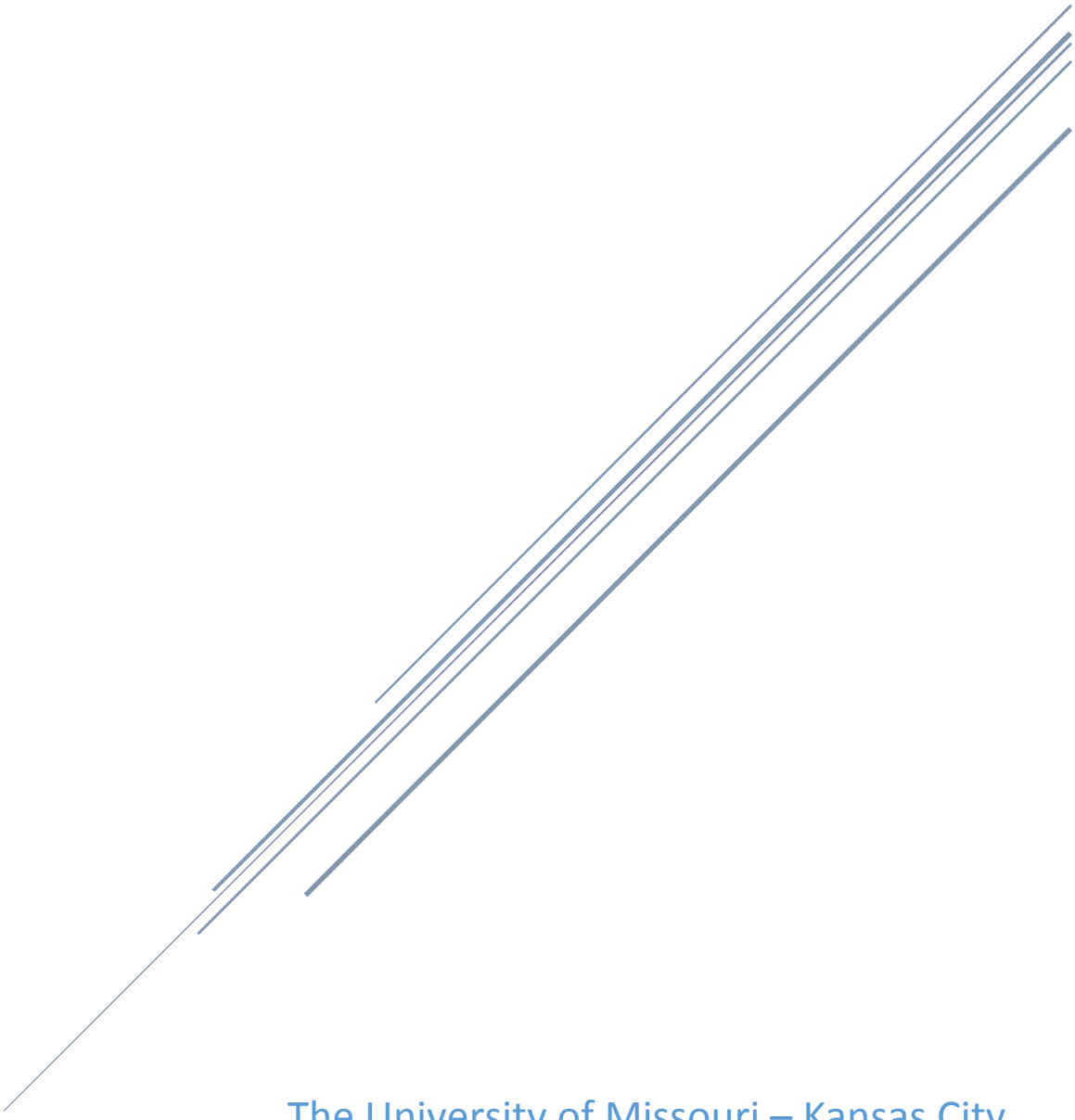
HEAL ASSIST: ITERATION 2

Hayden McParlane, Priyanka Bala and Komali Ghanta



The University of Missouri – Kansas City
CS 551 : Advanced Software Engineering

❖ Service Description

The drive to develop HealAssist comes in large part from the safety and security one obtains from having one's health data accessible from one's mobile device. Our application is intended to allow patients to locate helpful information related to their ailments as well as providing access to their physician-ordered treatment plans. For example, simple elements of treatment such as the dosage of one's medication, physical appearance of a prescription pill or liquid, frequency of administration of the medication and direction to support groups for various medical conditions are a few of the features that we plan on including in our application. We are designing the system such that it is independent of the Electronic Medical Record used allowing for integration of as many patient physicians as is necessary. Once physicians start to pick up on the value of linking patients to select aspects of their medical record, we will add functionality on our server side allowing patient access to personal records. As a result, HealAssist will become a centralized location at which one can monitor one's health status from many different physicians. Allowing parents access to their child's pediatric health record, allowing one access to their cardiac treatment plan in the same location as their internist's treatment plan. This high level of flexibility will allow users of HealAssist to maximize their health and, thus, to maximize their life.

❖ System Report

In its most general form this project is a client-server application that will use Twitter Bootstrap, JQuery and JavaScript at its front-end and a servlet on its back-end. Use of the Metronic theme template will beautify our user interface using Bootstrap and JQuery, which will be integrated with our server using custom-written JavaScript on the client side. Communication with our servlet will take place by use of REST. In other words, URLs will be sent to our server address requesting access to resources. While the front-end will be relatively simple, the back-end will be far more complex.

Our back-end system, as it currently stands, will consist of multiple high-level components. The *Client Service Component* functions as an interface by which clients can retrieve and manipulate their data. Database access will be controlled by a *Data Service Component* utilizing the adapter pattern to decouple the data access interface from the underlying database technology used. Due to the importance of information availability and consistency in health applications, our server will contain an *Information Management Component* which will manage information-related tasks. An example of such a task includes pulling data from the EMR at certain time intervals to ensure data consistency between the remote EMR and the internal storage on the server. Assisting communication with external APIs that rely upon the OAuth

authentication protocol, an *OAuth Management Component* will be designed. As extraneous as this seems, this is necessary because communication needs to occur between Google's OAuth servers and our server during the authentication process. Many APIs require use of OAuth for application authentication against the API. Furthermore, OAuth is heavily used by application developers. It allows for cross-application communication, so the presence of a full-blown system component dedicated to the protocol seems necessary given the likelihood of using this client-server application with additional APIs later on in the future. Finally, the *Data Routing and Retrieval Component* holds the responsibility of communicating with external data sources and retrieving data in multiple formats. This is our design in its current state. However, as with most software projects, certain aspects will change. Despite that volatility, one central concern of ours is that our system be as extensible and modular as possible, decreasing the likelihood of dependencies on external APIs, specific database technologies, etc. Many of which could cripple our development effort later on in the process were those components tightly coupled.

While implementation of all of the components mentioned above is ideal, we are going to approach development in phases. For the time being we will make simplifying assumptions eliminating the need for further development and implementation of the *Information Management Component*, *Communication Module* and *OAuth Component*. Completion of the front-end along with the *Client Service Component* and the *Database Access Component* in the back-end will spark discussion of the additional components. We want to ensure a high-quality prototype by the end of the semester and, due to this projects size and complexity, that will require such phase-based development. If we progress at a much faster rate, we will continue on with development of the *Information Management Component*, *Communication Module* and *OAuth Component*.

❖ Implementation Details

Our implementation of REST services is achieved through use of a dynamic web application within Apache Maven developed through use of the Eclipse IDE for Java EE developers. Our front-end design will consist of customized JavaScript sending REST resource requests to servlets present on our server. For diagrams detailing system design see the "System Diagrams" section of this document.

❖ Imported Services / APIs

Our system has undergone many changes since our previous iteration. It seems necessary at this point in time to add simplifying assumptions to our project to eliminate highly risky

complexity. Resulting from this, we are going to have to re-evaluate the APIs that we use. One assumption that we are going to make is that the data present in the database is current. Doing so will mean that pulling that data no longer becomes necessary because we'll populate the database once with the exception of client interaction and modification of client-side-specific fields. Despite this simplifying assumption, APIs are still being investigated for use. Assuming we are able to implement a large portion of our features, eventually we will add functionality that utilizes the Google Calendar API in order to provide clients with automatic calendar population for items such as taking medications and recommended excercise. Doing so will, in turn, require use of an API offering OAuth 2.0 application authentication and authorization. These two APIs are most likely going to remain in use. However, APIs that were originally introduced to populate our database with medical information in real-time will no longer be necessary such as the DrChrono API. The PillFill API used to retrieve pharmaceutical information will still be used along with the Chrome Alarm API.

❖ Completed Stories

The following are the user stories that we have implemented thus far:

1) As a user, I should be able to log in to my account providing assurance that I'm the only person who can access my medical information.
2) As a user, I should be able to register for the service.
3) As a user, if I fail to enter all the required fields during registration it should be obvious which fields I need to include in the registration process so that registration is quick and easy.
4) As a user, I should be able to view my personal information so that I can verify that it's correct and recent.
5) As a user, I should be able to quickly navigate the service by use of search functionality so that finding specific items is quick and intuitive.
6) As a physician, I should be able to view personal details associated with patients of mine so that I can enter, edit and update their profile.
7) As a physician, I should be able to attach different cases to patient profiles to expand upon their treatment.
8) As a physician, I should be able to attach different prescription medications to patient files and information related to their prescription.
9) As a physician, I should be able to add details to a patient file and those details should be saved for access at a later time.
10) As a user, feedback should be given to ensure that I don't accidentally permanently delete important information.

❖ System Diagrams

While we are extremely proud of our project and its size, reflection upon project risk is extremely important and size and complexity increase project risk. After careful thought and deliberation we have decided that we need to simplify our project until more features are successfully implemented and are high in quality. The result will be a sizable prototype depicting the concept driving our project and a massive reduction in risk of inadequate results. Unfortunately, that reduction will largely eliminate the need for many of the class diagrams present in the following section. Specifically, the *Information Management Component (IMC)* and the *Communication Module (ComModule).* However, when successful implementation of numerous features is accomplished and it becomes apparent that project risk has lowered significantly we will resume implementation of those components.

Figure 1: High-level component diagram depicting the HealAssist server architecture.

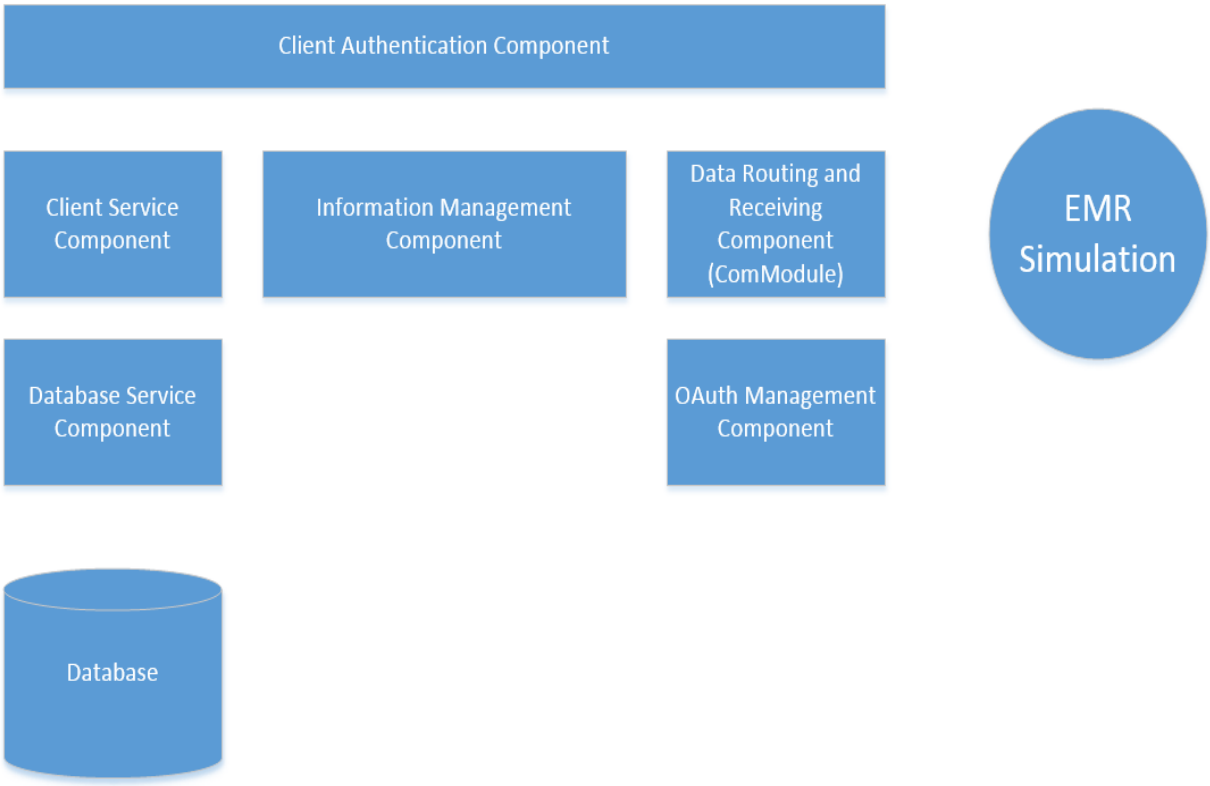


Figure 2: HealAssist Client-Server Architecture.

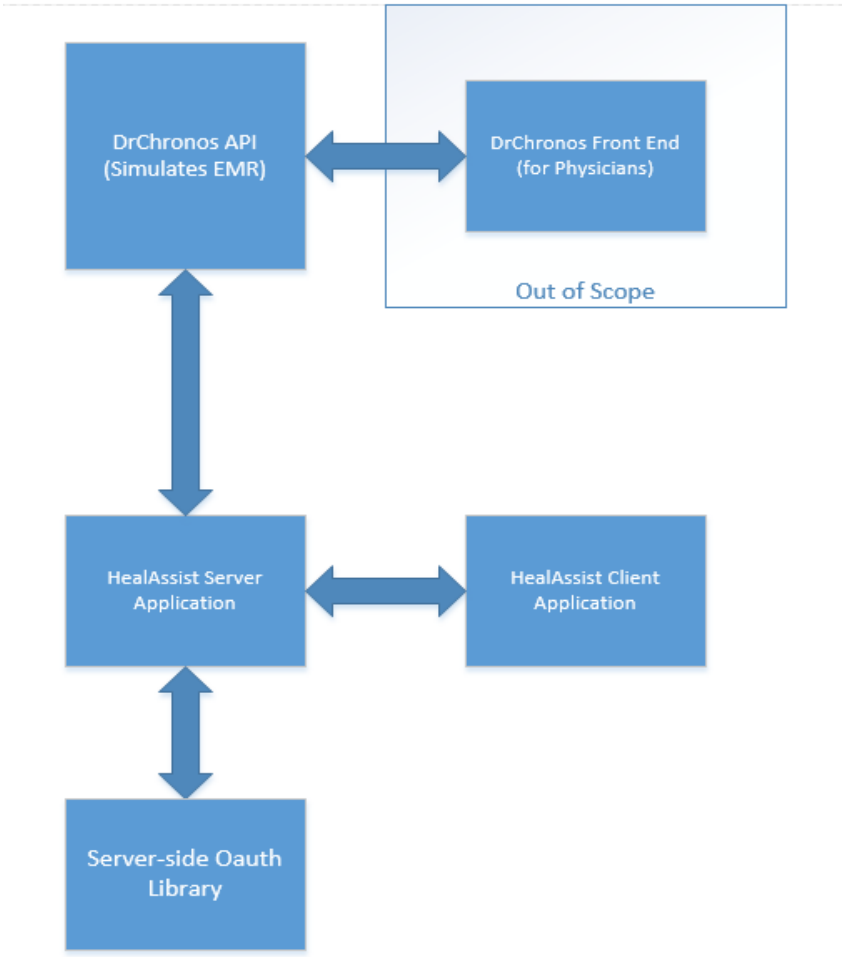


Figure 3: High-level diagram depicting encapsulation of database within a database access interface.

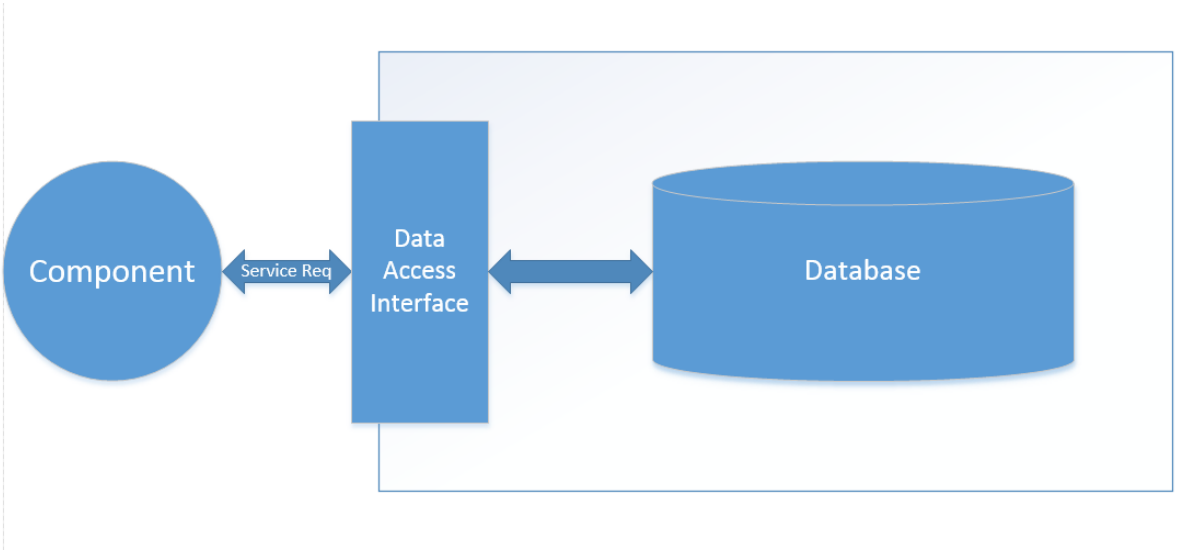


Figure 4: High-level depiction of the data routing and receiving component responsible for gathering data from external APIs.

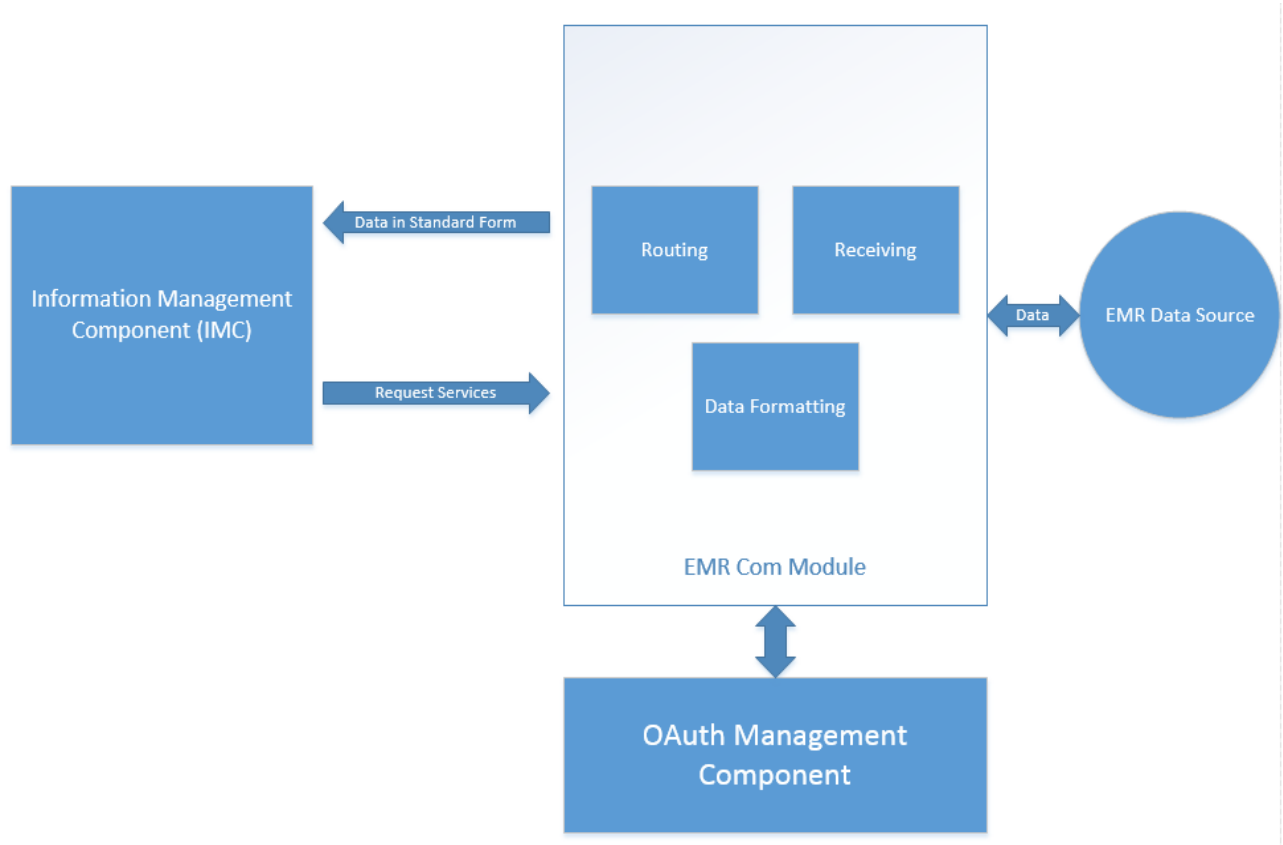


Figure 5: Class diagram depicting the communication module responsible for gathering data from external sources. Conceptually, the TaskDelegator creates parallel DataFetchers which execute REST requests via the many routers provided by the system. After completion of a data fetch the resultant data is packaged and loaded onto a queue visible to the Information Management Component (IMC).

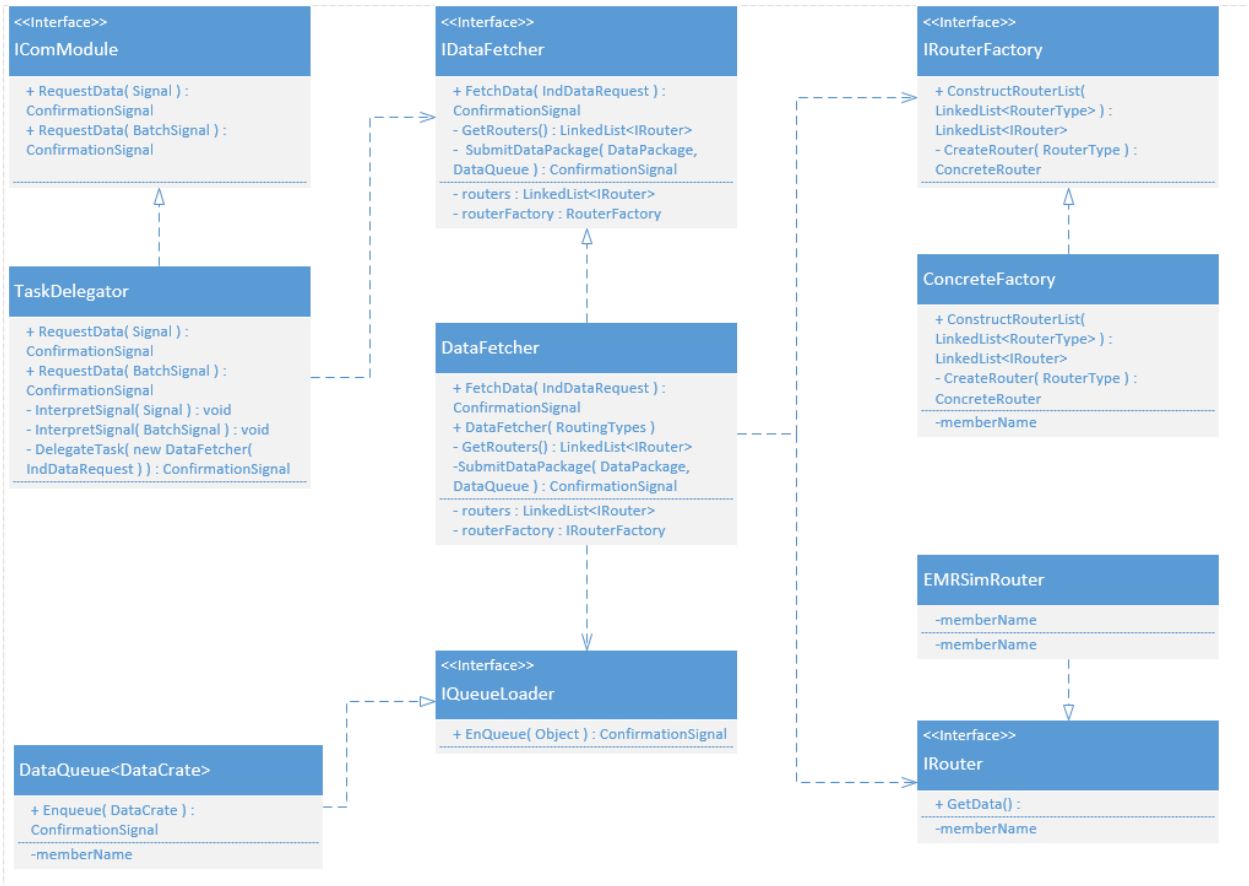


Figure 6: The IRouter portion of the communication module. This interface is used by the DataFetcher to execute commands against registered APIs. This diagram is incomplete.

Figure 7: The main Information Management Component (IMC). Its central purpose is ensuring that user data located on the HealAssist server is synced with remote data sources as well as servicing requests made by the clients for data from remote APIs. The InfoReceiver receives DataCrates packaged in the communication module and routes them to their destination.

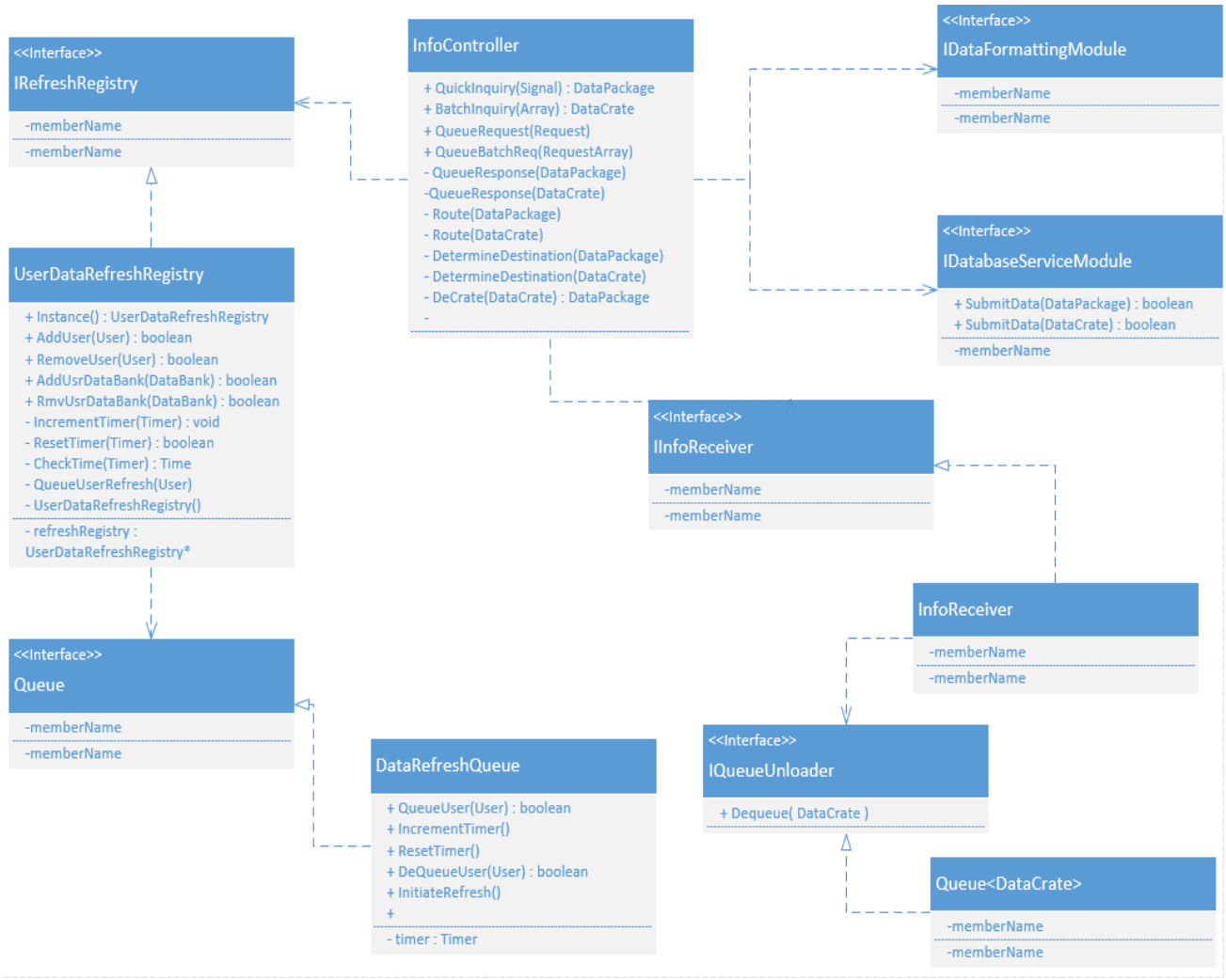


Figure 8: The Refresh Registry. This figure depicts the refresh registry used by the IMC to track data age. The refresh registry is a singleton that holds objects representing every user and entity for which data refresh is required. Upon elapse of a predetermined time interval, the RefreshManager extracts the expired entity and places it in a queue. The IMC then pulls that entity off of the queue and uses it as a signal to the communication module indicating the entity on behalf of which data should be fetched.

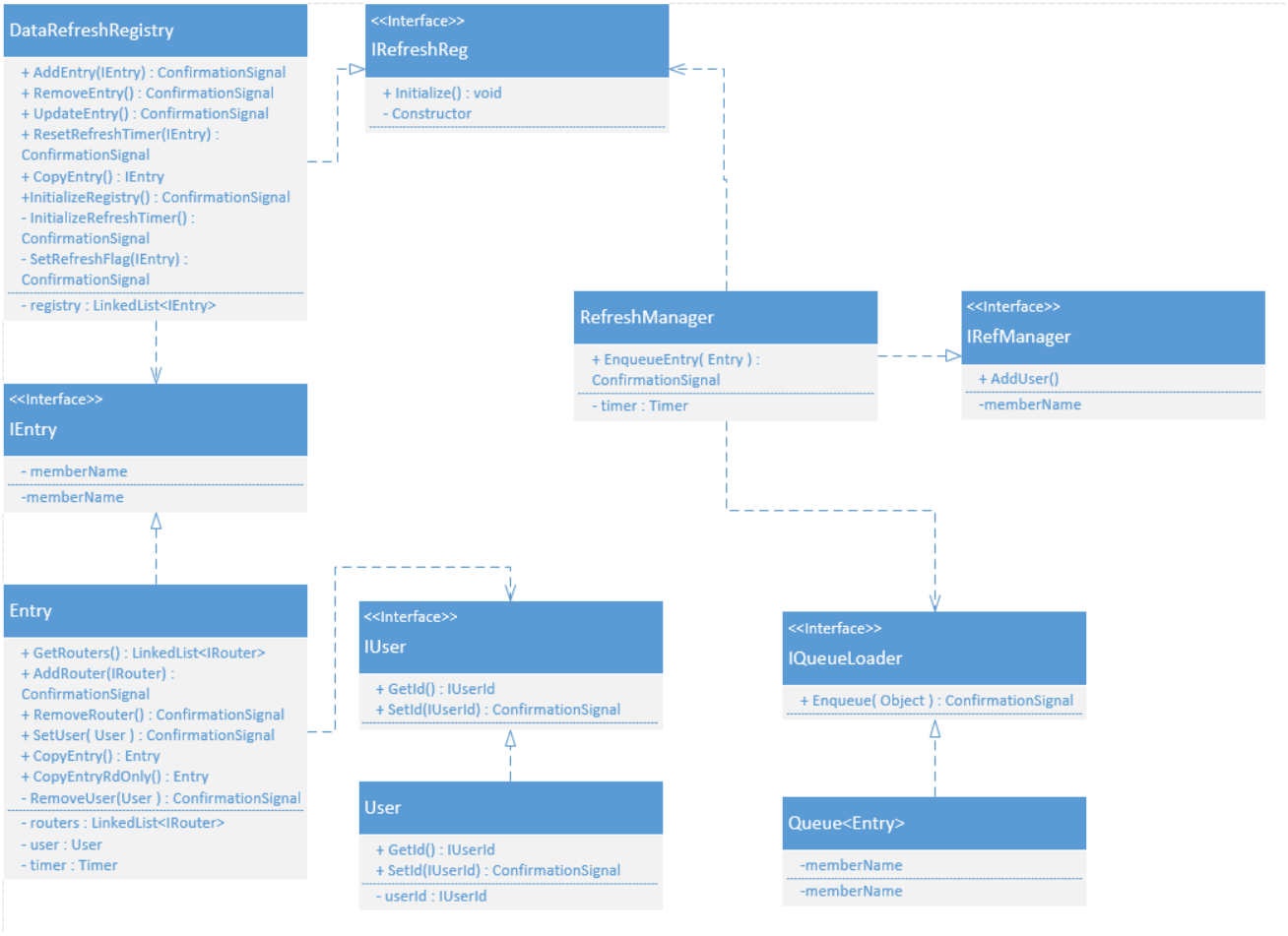
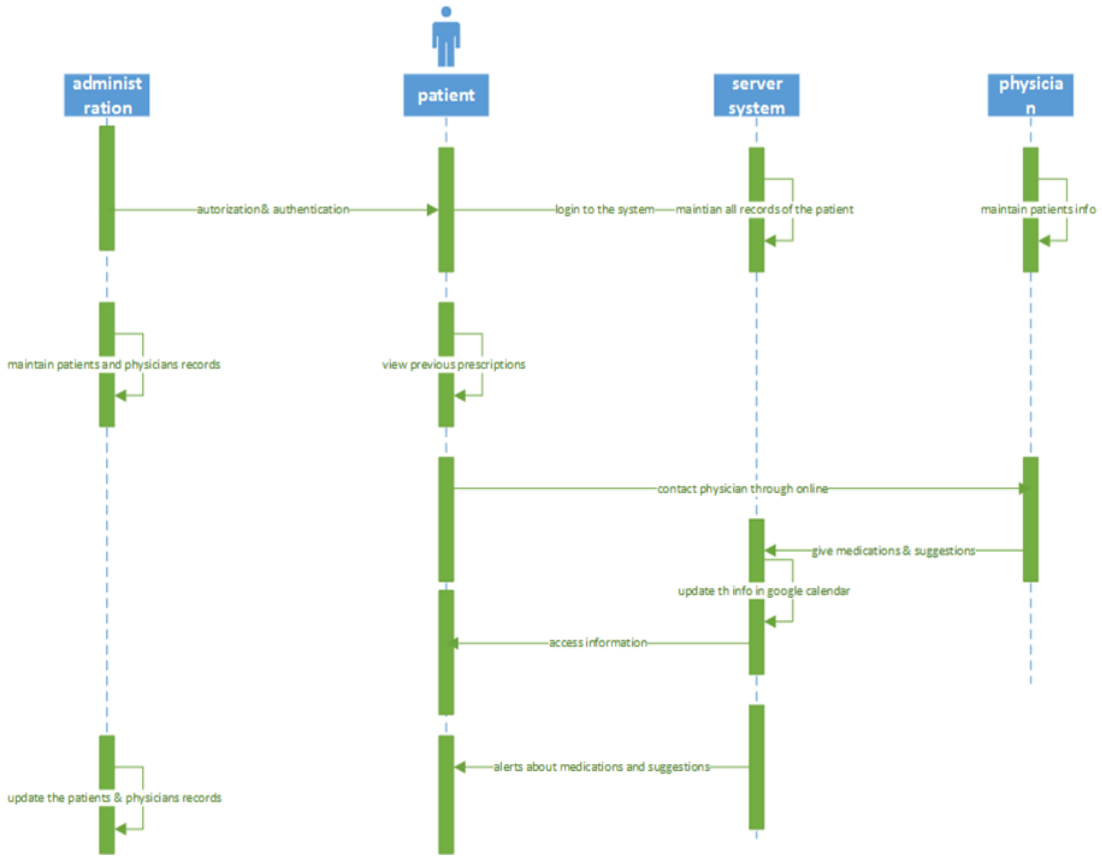


Figure 9: System sequence diagram.



❖ Screen Shots

Figure 10: The HealAssist login page for user authentication.

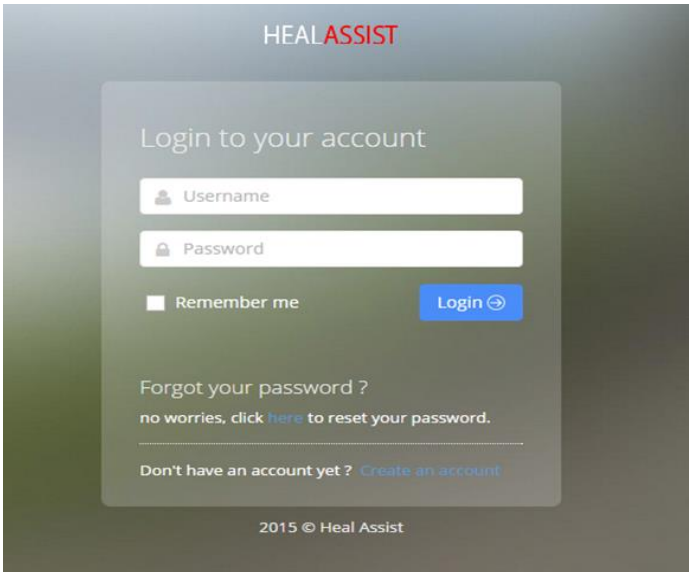


Figure 11: The top of the HealAssist registration page.

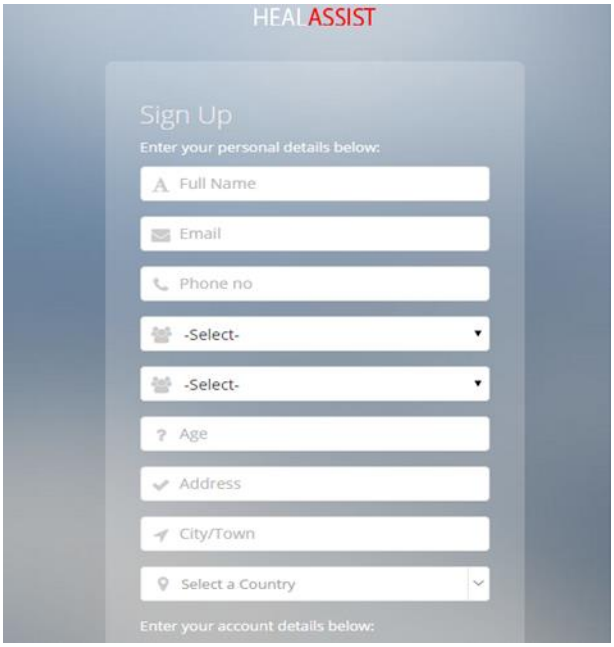
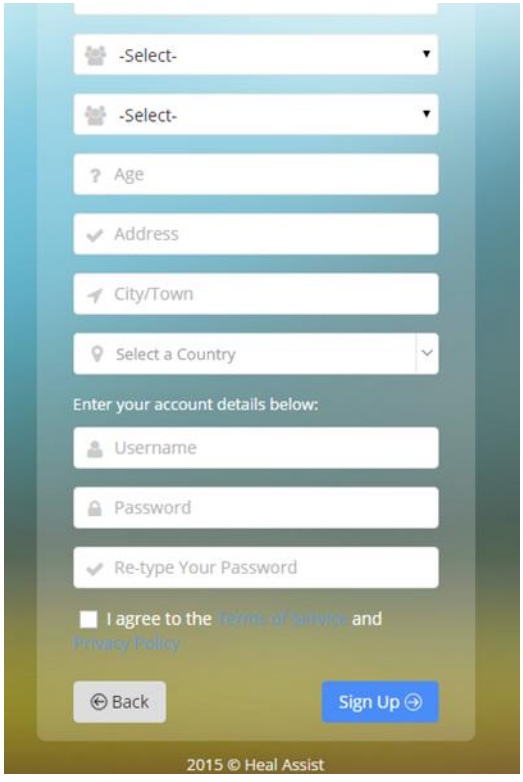


Figure 12: The bottom of the HealAssist registration page.



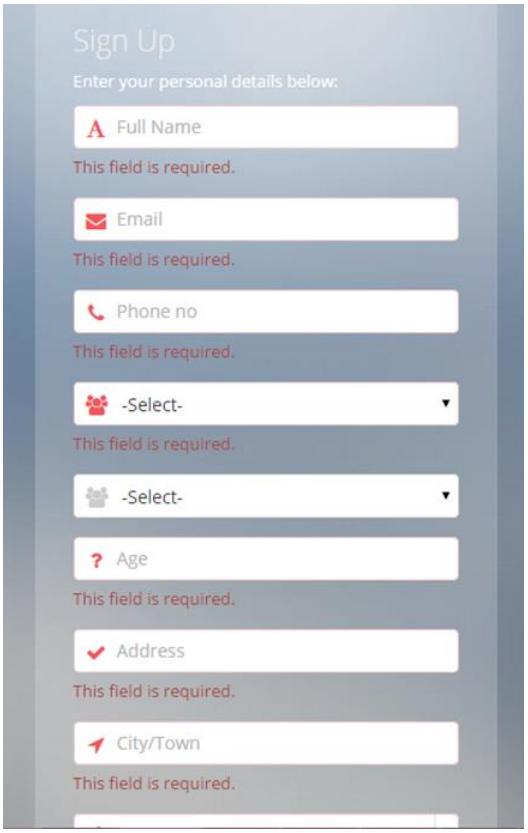


Figure 13: Input validation as seen on the HealAssist registration page (to left).

Figure 14: The patient management page for registered users after authentication.

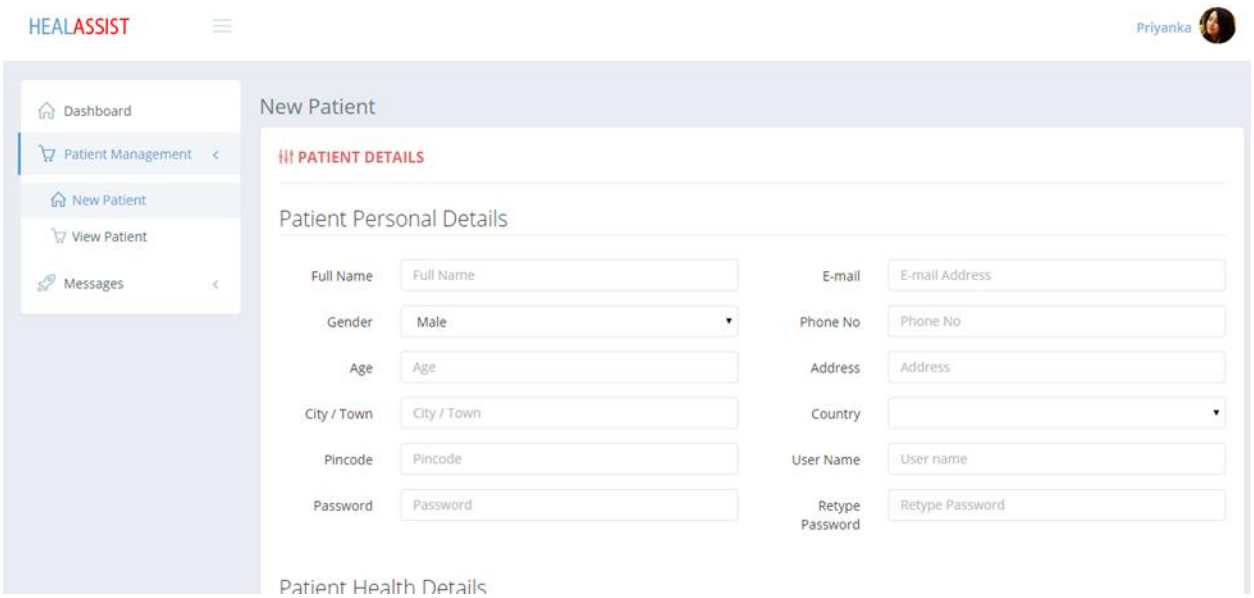


Figure 15: The search option provided by HealAssist.

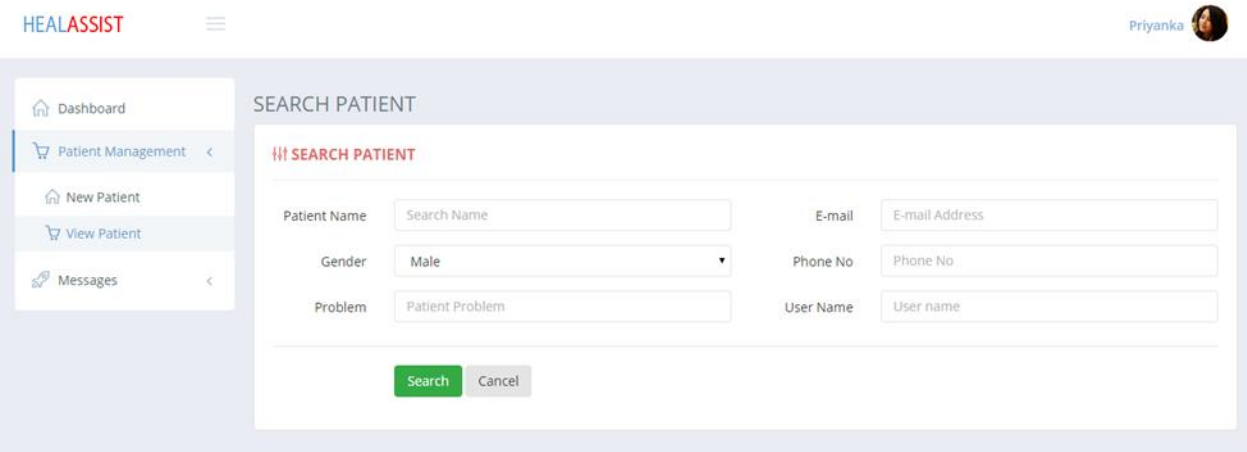


Figure 16: View patient personal details.

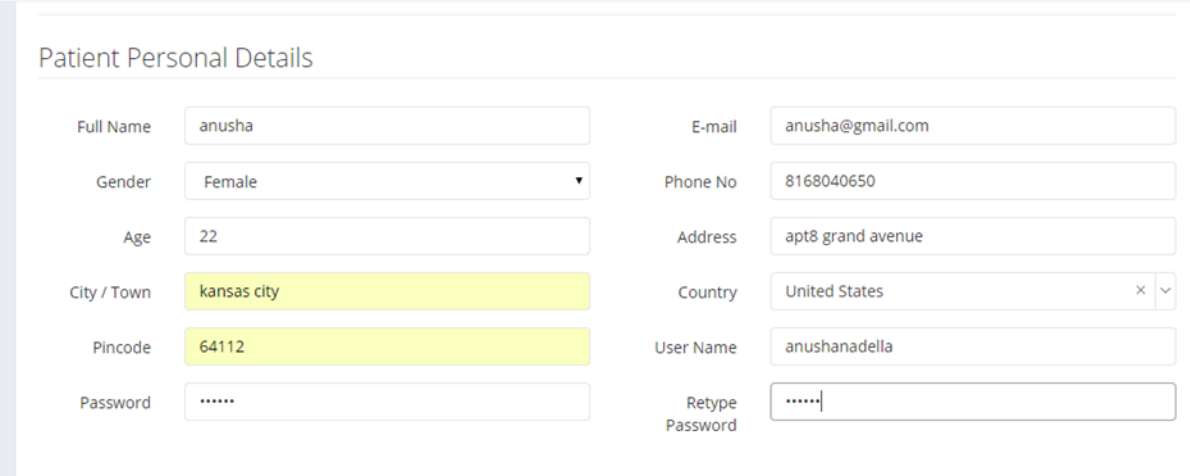


Figure 17: View patient cases and prescription details.

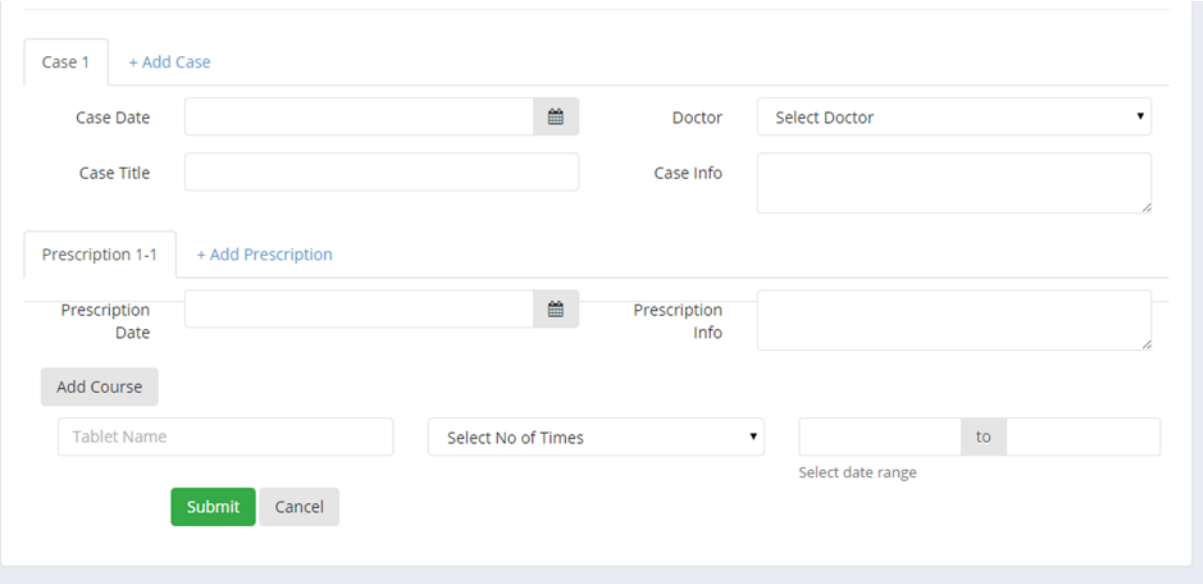


Figure 18: Add additional cases to a patient's chart.

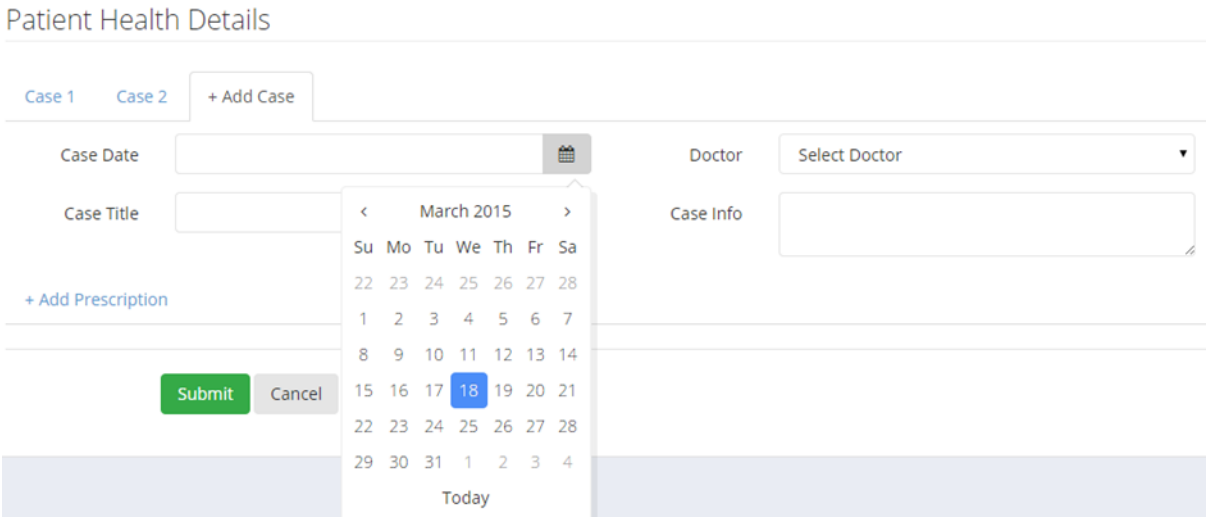


Figure 19: Add details to the patient case.

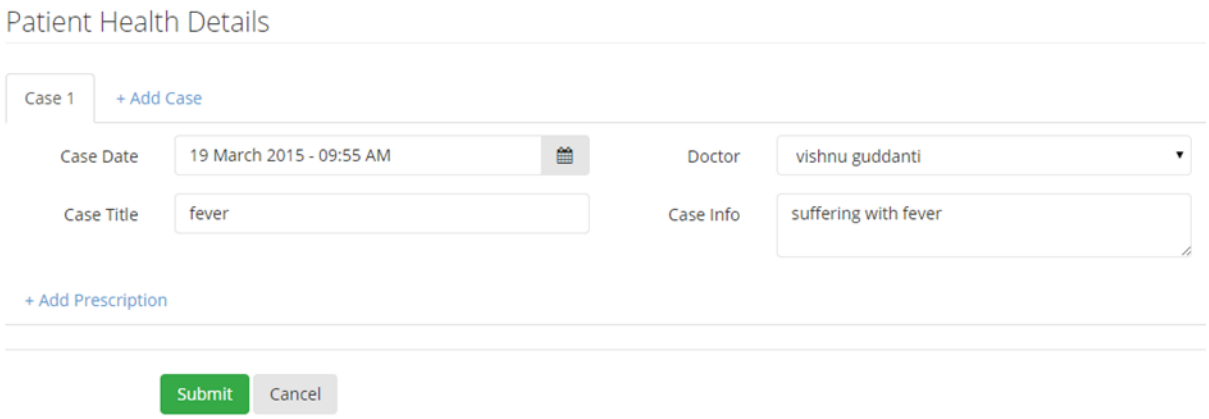


Figure 20: Add information to cases and prescriptions.

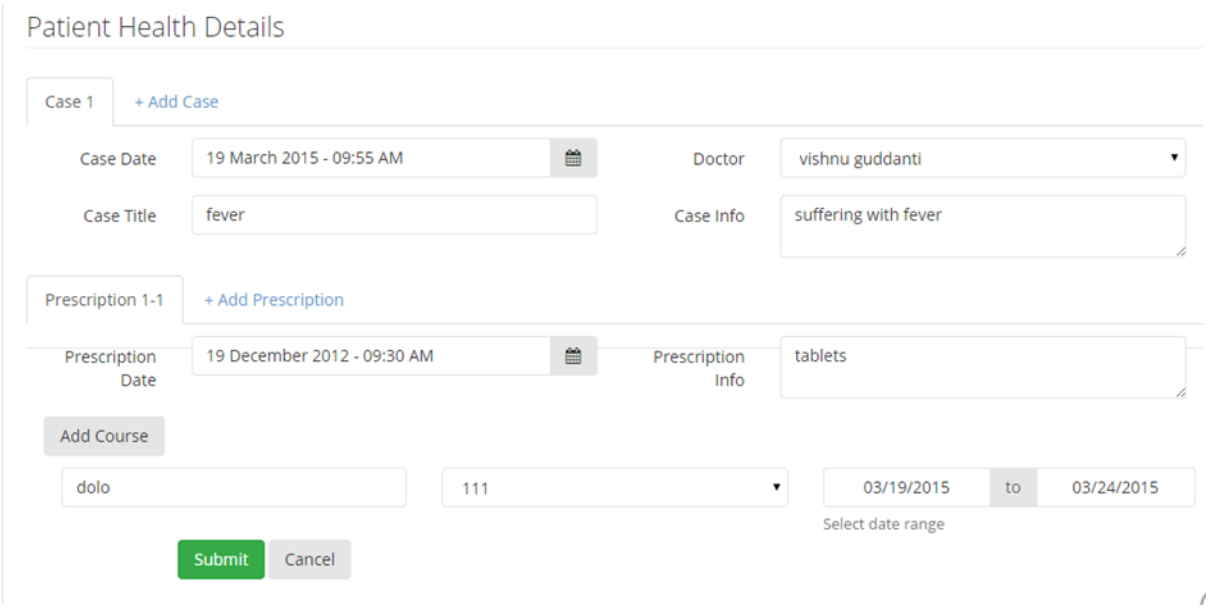


Figure 21: Add additional prescriptions to a patient's profile.

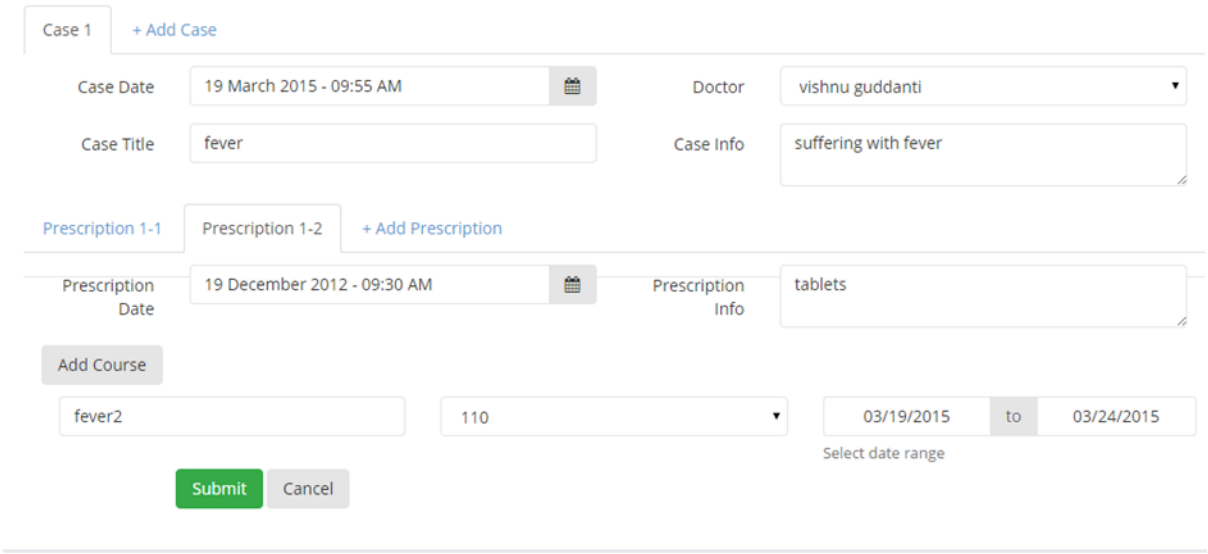


Figure 22: User feedback alert. If removal of case details is initiated alerts are used to ensure removal was intentional.

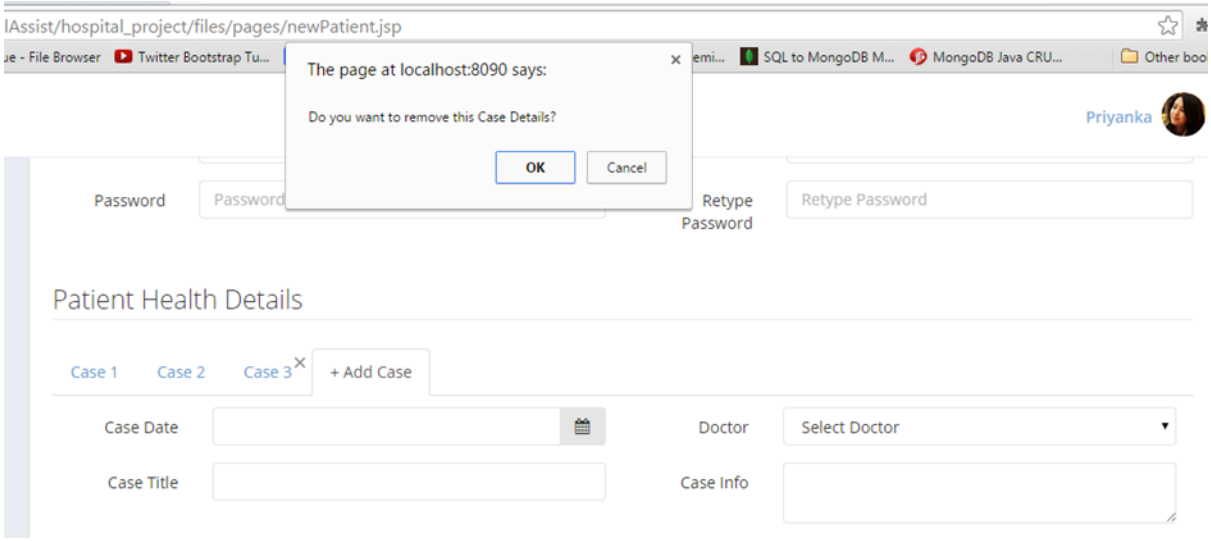


Figure 23: User feedback alert. If removal of prescription details is initiated alerts are used to ensure removal was intentional.

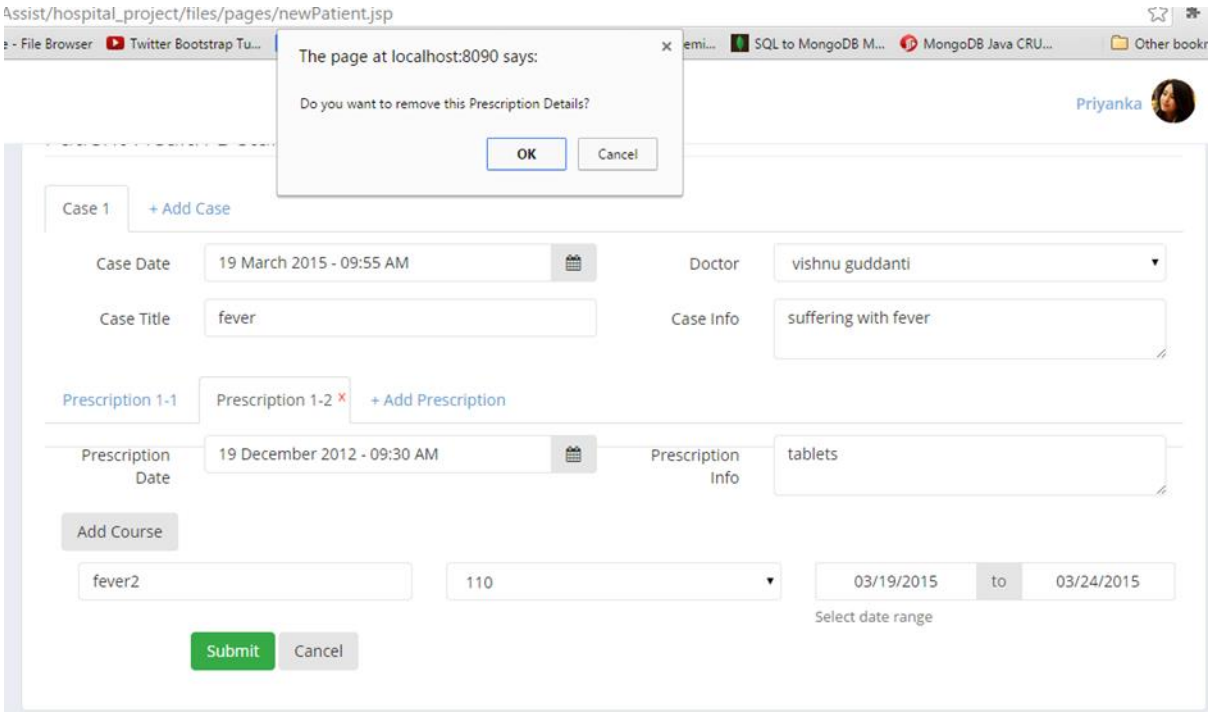


Figure 24: JUnit tests on beans package.

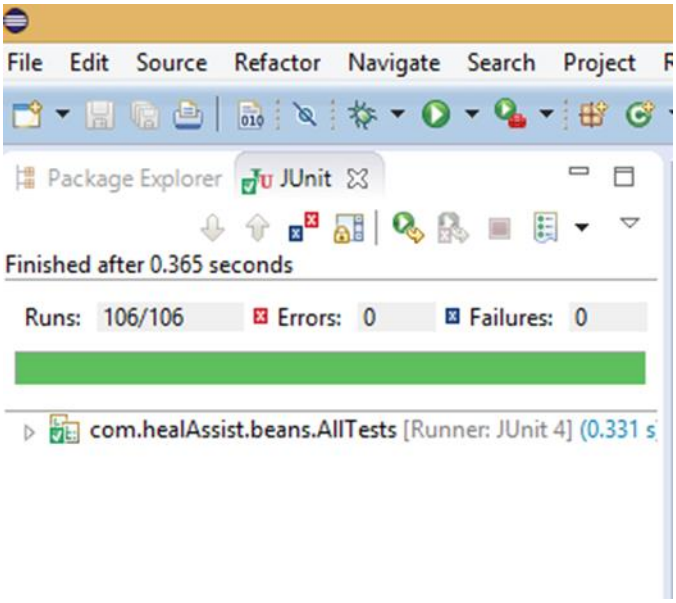


Figure 25: JUnit test on patient consumption class.

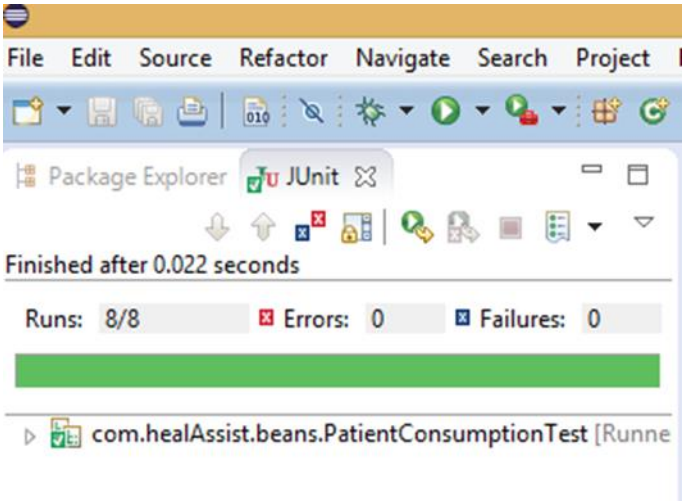


Figure 26: JUnit test on Contact Messages class.

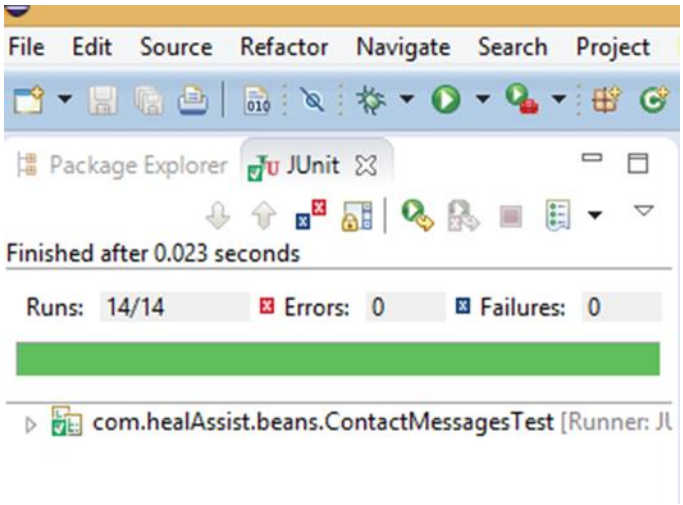


Figure 27: JUnit on login details class.

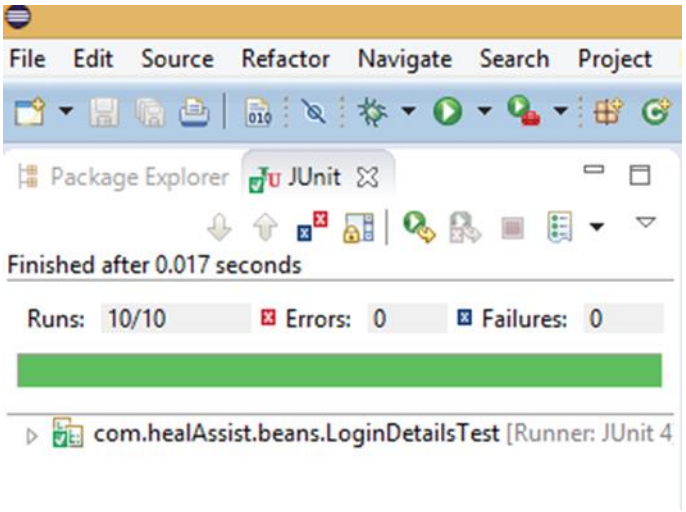
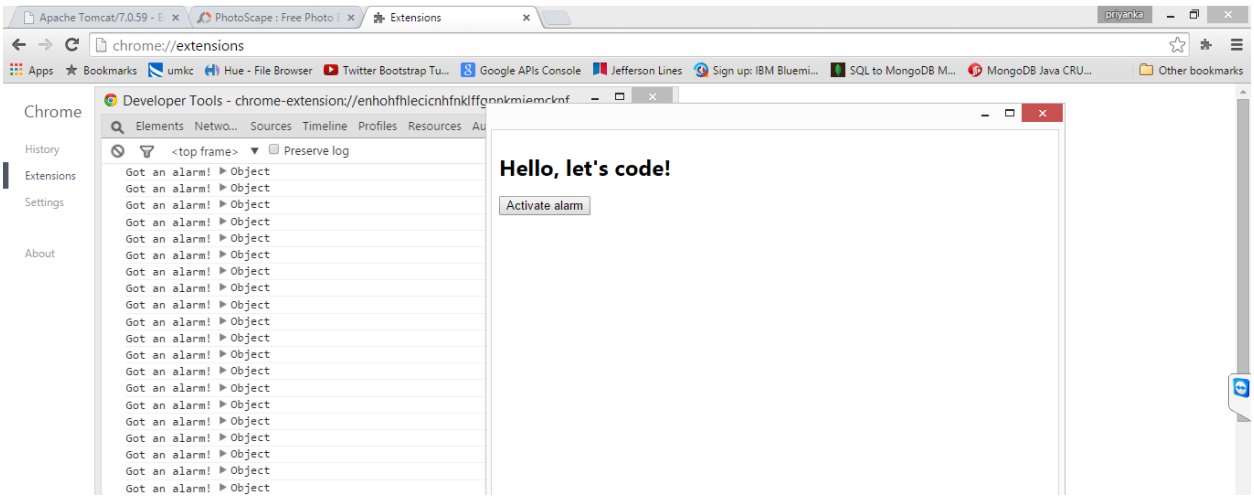


Figure 28: Screen shot depicting testing of the Chrome Alarm API.



❖ Deployment Sites

ScrumDo. https://www.scrumdo.com/organization/healassist/dashboard#

GitHub. https://github.com/HealAssist/HealAssist

❖ Project Management

Status Report

Weeks of implementation revealed the sheer size and depth of this project. While we believe the project can be completed while offering high quality services given our current time frame, it is an enormously large task and requires a huge amount of time. Such uncertainty introduces large amounts of risk. The chances of reaching the end of the semester without making sufficient progress rises. In accordance with risk management and smart project execution we are going to introduce additional simplifying assumptions in order to reduce the complexity and risk.

Originally, our plan was to design and implement our system such that remote data could easily and frequently be gathered from APIs. This server-side feature would request and respond to external APIs, manage user data refresh within the database, manage additional client requests and much more. In an effort to reduce risks associated with this project those aspects will be eliminated. Instead, we will populate our database with prototype data while applying the simplifying assumption that that data is current. This eliminates a large amount of complexity allowing for quick implementation of additional features and refinement of the client service and database access components. Despite the unfortunate need to reduce project complexity, our project remains sizeable and our implemented features will be greater in quality.

Unfortunately, simplifying the project as we plan will eliminate the need for the Information Management Component in addition to the Communication Module. Thus, large amounts of work have been wasted. However, the fact remains that we have covered a lot of ground up to this point and through division of labor and smaller tasks we will proceed at a much faster pace making the alteration desirable and smart. Time permitting, we may implement the features that are currently being removed if quality standards are met for our primary features well before the semester ends.

❖ Work completed

Description

- Designed the database schema and created the tables.
- Designed UI screens.
- Setting up environments.
- Analyzed the template and understood different components.

- Customized and developed login, registration, view patient, search patient screens.
- Stored registration details and authenticated user entering the system.
- Added functionality allowing physicians to add patient cases.
- Added functionality allowing physicians to add patient prescriptions.
- Added functionality alerting physicians before making persistent changes such as deleting important patient information (prescriptions and cases).
- Designed and implemented JUnit testing of multiple classes.
- Ran the JUnit tests.
- Researched and began implementation of Chrome Alert API and PillFill API.
- Refined design associated with API access (Communication Module).
- Refined design associated with information management (Information Management Component).
- Began implementation of Communication Module and Information Management Component (unfortunately, these components will be eliminated by simplifying assumptions so that the projects is reduced in scope).

Individual Contributions

- Priyanka – Primary front-end developer, user interface development, back-end development using a J2EE dynamic web project. Implemented dynamic JQuery tabs for adding specific details to patient cases and prescriptions. Implemented a sample program using the Chrome Alarm API. Heavy research and development on the PillFill developer API. Documentation.
- Hayden – Primary software engineer and architect. Project management. Refined design of Info. Management Component (added queue use for data passing between components, etc), User Registry and Communication Module. Began implementation of those components (projects are incomplete, located on GitHub). Documentation.
- Komali – Primary unit test designer and developer, back-end developer. Developed JUnit test cases and ran tests. Implemented hashing algorithms for user passwords. Heavy research and development on Chrome Alarm API. Documentation.

❖ Work Remaining

After working through the first and second iterations we have recognized that our project is too large in scope. As a result, we are eliminating the Information Management Component and the Communication Module by using the simplifying assumption that the data present in our database is current (i.e, does not require frequent updating). Resulting from that change, we will need to make slight modifications to our design. Additionally, we need to implement more features such as integration with Google Calendar through the Google Calendar API, additional patient features such as access to support groups relevant to their conditions, pictures of prescribed pills and other prescription verification inclusions. We also need to fully implement the APIs we have chosen (the PillFill and Chrome Alert API).

Responsibility:

- (User interface designer, Front-end developer, Server-side developer, Database Design and Implementation - Priyanka Bala Guddanti)
- (System Software Engineer and Architect, Server-side Developer, Project Manager, Documentation – Hayden McParlane)
- (Front-end Developer, Server-side Developer, Database Developer, Documentation - Komali Ghanta)