

CSCI5308 - QUALITY ASSURANCE

Project Report

MED ACCESS

Authored by

Karthik Rammoorthy	B00790749
Vidhyashree Boopathy	B00790060
Selvapriya Sampath	B00769290
Sricharan Ramasamy	B00790079

1. INTRODUCTION

At the present scenario, there are not many options that are provided to the users to search for doctors in their locality and book appointments with them online. MedAccess aims at providing one stop solution to let patients book appointments with doctors they wish to consult effortlessly. Our application also offers multiple features like letting users search for doctors, ability to view previously booked appointments and providing feedback to doctors based on their consultation experience. We have given utmost consideration to quality of the code in both front and back end. Our team implemented design patterns and refactored the code multiple times. Also, we have. We gave importance to the validations to the data and wrote test cases. The look and feel of the website have been considered carefully to provide cordial experience to the users. Also, we have designed the UI components in such a way that it will be easy for users to comprehend the features without any hassle.

2. TARGETED AUDIENCE

The application is mainly targeted to general users who wish to book appointments with physicians in and around their locality. The doctors can sign up with their licence number, speciality and hospital details in order to make their details visible to the patients.

3. TECHNOLOGIES USED

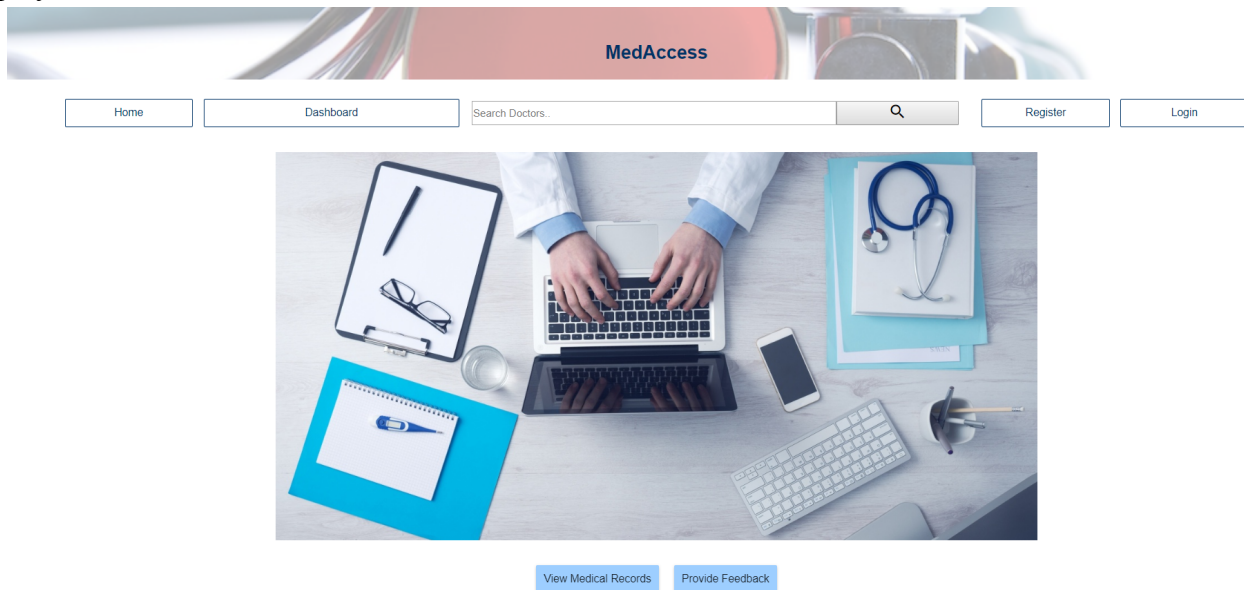
Component	Software Specification
Front End Framework	Angular 6 with HTML and CSS
Bank End Framework	Spring Boot REST API
Version Control	GitHub
Testing Tool	Mokito, JUnit
Database	MySQL
Host Server	Heroku

Front End Libraries	Angular Material, FlexLayout API, Express for Heroku
Collaboration Tool	Slack
Project Management	Trello

4. FUNCTIONALITY COMPONENTS

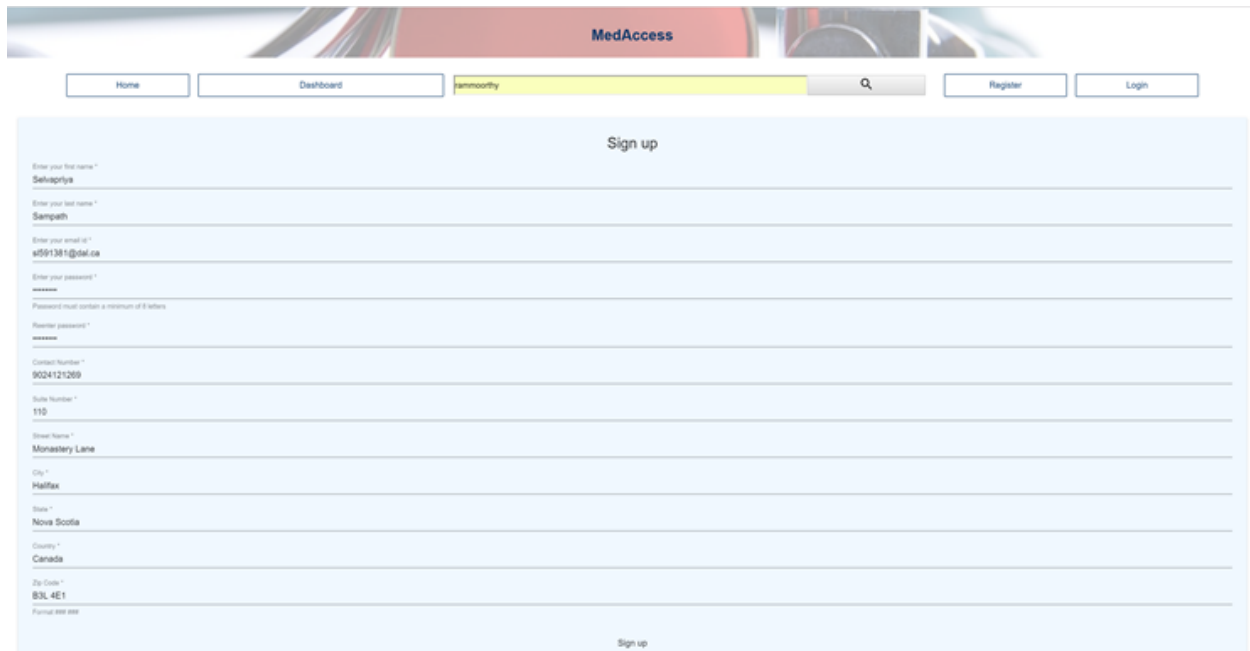
4.1 Home Page

This page has all the components like register, login and dashboard to provide easy navigation to the users. The page also has a search bar which lets the users search for doctor and the search results are displayed.



4.2 Register Page

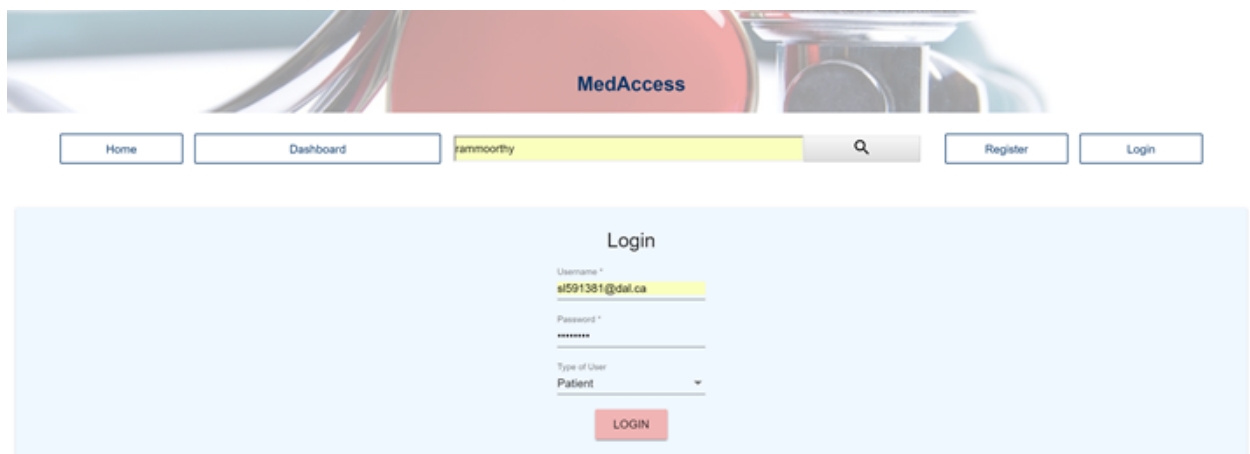
This page allows the user to register to the website. The field “Type of User” allows the user to choose if they are a doctor or patient. If the user is doctor, extra fields like licence number and speciality are displayed. Validations are provided to all the fields in this page. Once the user is registered, the details are stored in the database.



The image shows the 'Sign up' form on the MedAccess website. The header features the 'MedAccess' logo and navigation links: Home, Dashboard, jammoorthy (highlighted), a search icon, Register, and Login. The form itself is titled 'Sign up' and contains the following fields: First Name (filled with 'Selvapritha'), Last Name (filled with 'Sampath'), Email (filled with 's591381@dal.ca'), Password (masked with asterisks), Confirm Password (masked with asterisks), Contact Number (filled with '9024121269'), Suite Number (filled with '110'), Street Name (filled with 'Monastery Lane'), City (filled with 'Halifax'), State (filled with 'Nova Scotia'), Country (filled with 'Canada'), and Zip Code (filled with 'B3L 4E1'). A 'Sign up' button is located at the bottom right of the form.

4.3 Login Page

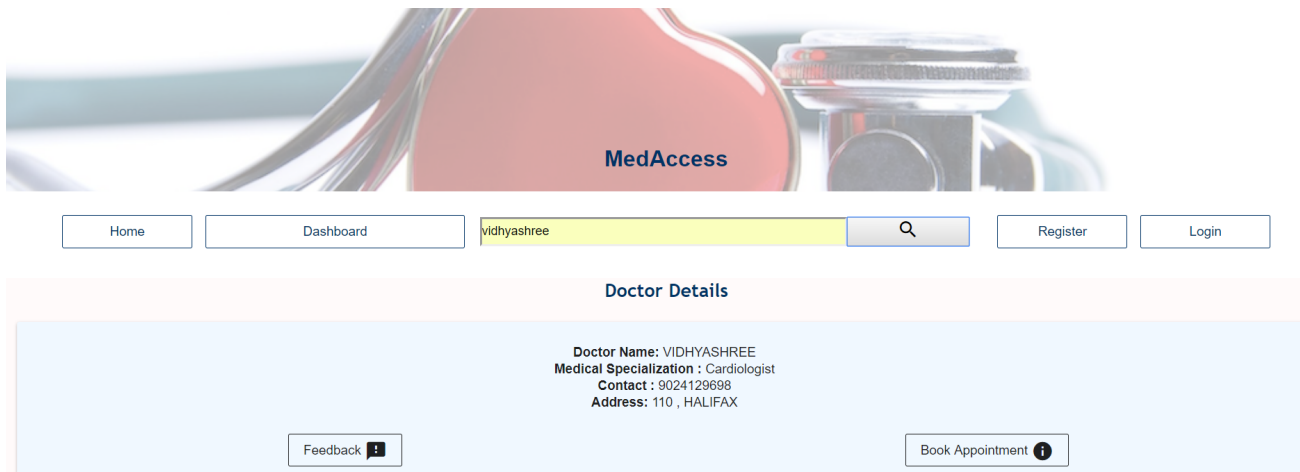
The login page allows the users to login. Username, password and the type of user has to be provide to login into the site. The login credentials are being stored in local storage and will be available until the user logs out.



The image shows the 'Login' form on the MedAccess website. The header is identical to the sign-up page, with the 'MedAccess' logo and navigation links. The form is titled 'Login' and contains the following fields: Username (filled with 's591381@dal.ca'), Password (masked with asterisks), and a dropdown menu for 'Type of User' (set to 'Patient'). A red 'LOGIN' button is positioned at the bottom center of the form.

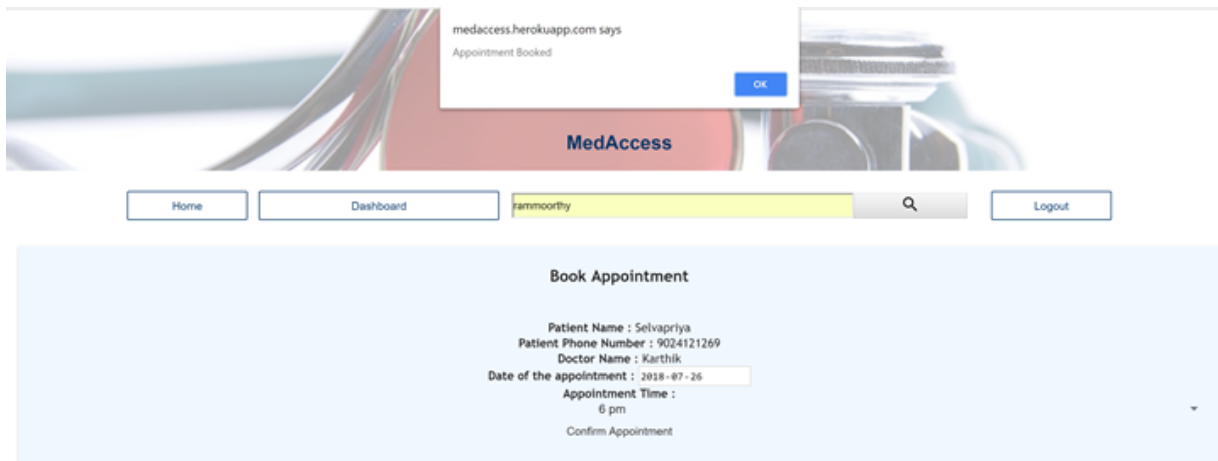
4.4 Search Results Page

Once the user searches for doctor in home page and the doctor details are fetched from database accordingly and displayed in this page. “Book Appointment” button is provided which when clicked navigates to Book appointment page for that particular doctor. This functionality is considered an important one, since the feedback and Book Appointment functionalities can be accessed only from here.



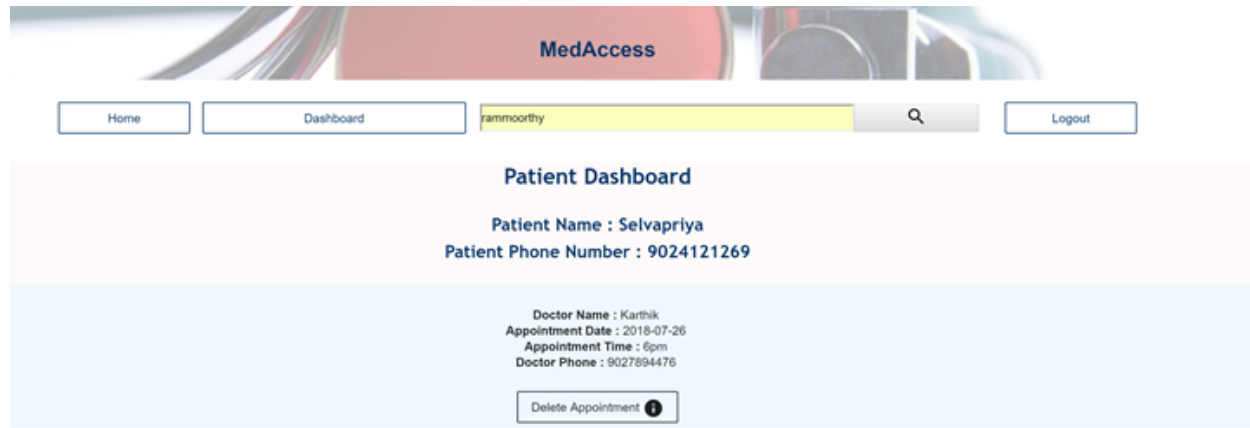
4.5 Book Appointment Page

Once the doctor details shows up in the result and the user clicks on “Book Appointment” button it navigates to Book appointment page which provides option to the patients with regards to appointment date and time.



4.6 Patient Dashboard

Once the patient logs in he/she will be navigated to the patient dashboard which displays the list of appointments previously booked. The user also has the option of deleting the appointment.

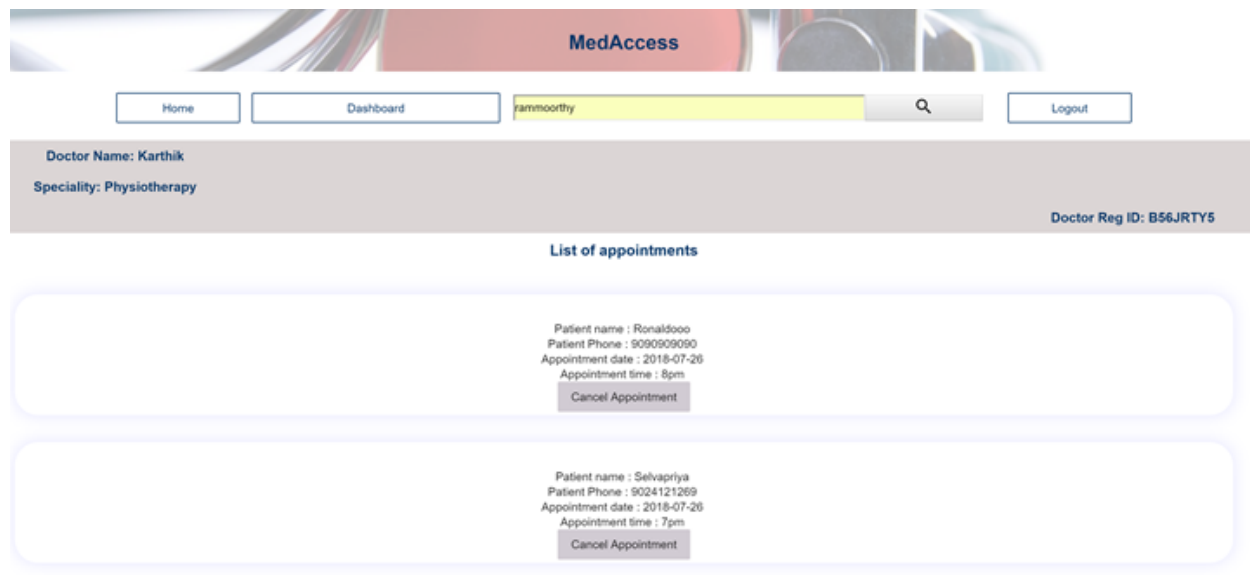


The screenshot shows the MedAccess Patient Dashboard. At the top, there is a navigation bar with 'Home', 'Dashboard', and a user profile 'rammoorthy' with a search icon and a 'Logout' button. The main content area is titled 'Patient Dashboard' and displays the patient's name 'Selvapriya' and phone number '9024121269'. Below this, it shows the doctor's name 'Karthik', appointment date '2018-07-26', appointment time '6pm', and doctor phone '9027894476'. A 'Delete Appointment' button with an information icon is at the bottom.

Patient Dashboard	
Patient Name : Selvapriya Patient Phone Number : 9024121269	
Doctor Name : Karthik Appointment Date : 2018-07-26 Appointment Time : 6pm Doctor Phone : 9027894476	
Delete Appointment ⓘ	

4.7 Doctor Dashboard

Once the doctor logs in he/she will be navigated to the doctor dashboard which displays the list of appointments made for him/her. The doctor has also the option of deleting the appointments in case he is not able to consult at that particular time.




The screenshot shows the MedAccess Doctor Dashboard. The navigation bar is identical to the patient dashboard. The main content area displays the doctor's name 'Karthik', speciality 'Physiotherapy', and 'Doctor Reg ID: B56JRTY5'. Below this is a section titled 'List of appointments' containing two appointment cards. Each card shows the patient's name, phone number, appointment date, and time, along with a 'Cancel Appointment' button.

Doctor Dashboard	
Doctor Name: Karthik Speciality: Physiotherapy Doctor Reg ID: B56JRTY5	
List of appointments	
<div>Patient name : Ronaldooo Patient Phone : 9090909090 Appointment date : 2018-07-26 Appointment time : 8pm Cancel Appointment</div>	
<div>Patient name : Selvapriya Patient Phone : 9024121269 Appointment date : 2018-07-26 Appointment time : 7pm Cancel Appointment</div>	

4.8 Provide Feedback

The patients can provide feedback to the doctors based on the consultation. It basically lets the user to provide rating to the user.



[Home](#) [Dashboard](#) [Logout](#)

Feedback

User Name
Sricharan

Email ID
charan@gmail.com

Doctor's Name
cersei


Leave a Comment

[SUBMIT](#)

[READ FEEDBACK](#)

4.9 View Feedback

The Patients can also be able to view the feedbacks given for that particular doctor. The View Feedback button is present on the feedback page.



[Home](#) [Dashboard](#) [Logout](#)

User Feedback		
Name: Sricharan	Email : charan@gmail.com	Description : Not that great
Name: arya	Email : arya@gmail.com	Description : I would not recommend

5. TEST DRIVEN DEVELOPMENT

We worked on agile methodology, where we pushed code after each sprint. For each sprint, there was a scrum master who was responsible for pushing code into master from develop branch in git. The role of scrum master was assigned to everyone of us in turns every week. The development process of our application was “Test driven development”. We brainstormed on the requirements of each functionality and converted them into test cases and then developed.

The following were the steps performed to follow the test driven development.

- **Writing test cases** : The test environment was setup and test cases based on the requirements for each functionality were written.
- **Running test cases** : The test cases were run to analyze the functionalities' behaviour.
- **Writing code** : Based on the test cases the code was written to perform the functions and the test cases are run again to check if they pass.
- **Refactoring the code** : After the functionality is written code was refactored to ensure the quality.

6. DESIGN PATTERNS

The following design patterns were implemented.

- Observer Pattern
- Chain of Responsibility

6.1 Observer Pattern

Angular provides us features to develop a single page and progressive web application. All the components are modular with its CSS, HTML and Typescript classes. For the components to communicate and react dynamically, Subject-Observer pattern is implemented. This helped a lot for login functionality where menu bar component observes when login component performs any operation.

6.2 Chain of Responsibility

The Chain of Responsibility Pattern is achieved for the Loggers implementation. The Logger levels used in the application are INFO, DEBUG, and ERROR. Each Logger Level is assigned with a value. The Chain of Responsibility for each Logger is set in the ChainOfRespDesignPattern.java. When the requester sends a request to write Logs, the receiver object of the abstract class captures the request and sends it to the respective receiver class. The Receiver classes are: ConsoleLogger, ErrorLogger, and FileLogger. This decouples the sender and receiver objects based on the input type.

7. PRINCIPLES USED

7.1 Dependency injection

Since we developed the application using MVC pattern, we had to refer other classes. For example, Controller class passes the request to Service classes which calls the DAO object. We created a base interface for the different components and created interface objects for communication. We used Autowired functionality for reference purposes. This way, tightly coupled class were removed and the system became more cohesive.

7.2 Interface Segregation principle

Interfaces was created for all the functionalities separately instead of creating a one big interface. For example, separate interfaces was created for each of the functionalities in DAO and Service classes.

8. CONFIGURABLE BUSINESS LOGIC

We created validation rules for registration component such as the minimum number of digits for password, zip code pattern, etc. All these validation rules are stored in database as a table. The register component in frontend extracts the validation rules employs it for validation. The valid input is sent back to backend for insertion process.

9. LOGGING FUNCTIONALITY

The logging functionality is implemented using the tool log4j. The entire flow of the application is closely recorded. This functionality also uses one of the Behavioural Patterns which is Chain of Responsibility. The different levels of log captured for this application are INFO, DEBUG, and ERROR.

10. NAMING CONVENTIONS

- Classes were named as nouns and in a manner that it will make the user easily understand about its function. For example, class which holds the behaviour of Doctor connection to database is named as DoctorDAO.
- Methods were named as verbs with regards to what the method will do. For instance, a method which performs the activity of getting patient details by ID is named as `getPatientbyID()`
- Meaningful variable names were created. It will help anyone who reads the code to understand what exactly the variable does.
- Package names were created as per the standards and was followed throughout the entire code. For example, `com.MedAccess.Dao` was created to hold the DAO classes.

11. REFACTORING

Though we were cautious to follow all the refactoring techniques during code development, we missed on several things. For example, we built large monolithic classes and added unnecessary references in our classes. Once we completed the basic functionalities of the project, we ventured into refactoring the code. We deleted the unnecessary reference. Also, we replaced SQL statements

with SQL stored procedures. We performed Extract technique to divide the classes and methods to make the code more modular. We also refactored God classes into DAO, Service, Model and Controller classes.

12. INDIVIDUAL CONTRIBUTION

The individual contribution with respect to the modules are as follows. Each of us in our team performed development for front and back end and writing test cases.

S No	Module Name	Contributor	Action Performed
1	Patient Register Component	Karthik Rammoorthy	<ul style="list-style-type: none">• Designed Angular forms with asynchronous validation.• Built corresponding business logic and deployed as a REST endpoint from backend• Send the validated data to backend via REST API with JSON.• Performed unit tests on the component.
2	Search Result Component	Karthik Rammoorthy	<ul style="list-style-type: none">• Designed Angular toolbar with search bar• Built corresponding business logic that compares the keyword with database and responds. This is also deployed as a REST endpoint from backend• Send the parameter and receive the response body from backend via REST API with JSON.• Performed unit tests on the component.
3	Configurable Business Logic	Karthik Rammoorthy	<ul style="list-style-type: none">• Stored all the validation rules in database• Extracted the data to frontend and employed it to validate user data.

			<ul style="list-style-type: none"> Performed unit tests on the component.
4	Provide Feedback Component	Sricharan Ramasamy	<ul style="list-style-type: none"> Designed Angular forms with Patient details already available on the fields. Sent the validated data to backend via REST API with JSON. Performed Unit Tests on the operations.
5	View Feedback Component	Sricharan Ramasamy	<ul style="list-style-type: none"> Consumed the feedback details from the backend via the REST API for the particular doctor. Displayed all the feedbacks for that particular doctor in separate card. Performed unit tests on the component.
6	Login Component and Local Storage (Session)	Sricharan Ramasamy	<ul style="list-style-type: none"> Login details given in the forms are validated by checking the values with the backend by consuming the REST API. Maintained Session management throughout the flow of the application by implementing Local Storage into the project. Validated all the components when the user tries to access without logging in with the use of local storage.
8	Doctor Register Component	Vidhyashree Boopathy	<ul style="list-style-type: none"> Designed Angular forms with asynchronous validation. Built corresponding business logic and deployed as a REST endpoint from backend Send the validated data to backend via REST API with JSON. Performed unit tests on the component.
9	Doctor Home Component <ul style="list-style-type: none"> Display Patient 	Vidhyashree Boopathy	<ul style="list-style-type: none"> Designed Angular forms with asynchronous validation. Built corresponding business logic

	Appointments <ul style="list-style-type: none"> • Display Profile Details • Cancel Appointment with Patients 		and deployed as a REST endpoint from backend <ul style="list-style-type: none"> • Send the validated data to backend via REST API with JSON. • Performed unit tests on the component.
10	Login Component	Vidhyashree Boopathy	<ul style="list-style-type: none"> • Front-end Angular design of the Login component using material design • Front-end validation of the login page with reactive error messages.
11	Design Pattern - Chain of Responsibility	Vidhyashree Boopathy	<ul style="list-style-type: none"> • Chain of Responsibility Pattern was implemented for the Loggers.
12	Loggers	Vidhyashree Boopathy	<ul style="list-style-type: none"> • Logs were recorded for the classes to track the services and workflow. • Different Levels of logs like Info, Debug, and Error were tracked throughout the application
13	Unit Test Cases	Vidhyashree Boopathy	<ul style="list-style-type: none"> • JUnit Test cases were developed for the classes using the tool Mockito to mock the data.
14	Book Appointment component	Selvapriya Sampath	<ul style="list-style-type: none"> • Designed the front end for the module. • Built the back end logic to add and delete appointments using REST • Performed unit tests on the component
15	Registration page front end unit testing	Selvapriya Sampath	<ul style="list-style-type: none"> • Front end unit testing was performed for the registration component using Karma and Jasmine frameworks
16	Patient Dashboard component	Selvapriya Sampath	<ul style="list-style-type: none"> • Designed UI for the component to display the patient and appointment details • Developed the backend logic to retrieve the appointment details and display it in the dashboard. • Send the validated data to backend via REST API with JSON. • Wrote unit test cases for the

			component.
--	--	--	------------

13. CHALLENGES FACED

- As all of us were new to **Angular 6** and it posed as challenge initially to get accustomed with the new technology
- All of us in our team has experience in developing code and later writing test cases for them. So, it was little difficult for us to work on the **Test Driven Methodology** for which we later got comfortable.
- Deciding on what **design patterns** would be best suited for our project was challenging.
- Our team was new to **Full stack development** and it took us a while to understand how it exactly works.
- Our team faced several challenges while deploying the code in Heroku. Though hosting backend was easy, hosting Angular proved time consuming.

14. TECHNICAL DEBT

- We chose Node.JS to implement the business logic for the application in backend. Since it was not a suitable technology, we switched to Java Springboot API. We lost several work hours on this, However, because of the agile workflow, we adapted and completed the tasks without lag.
- We implemented JPA for model and DAO communication. However, it provided templates which is not permitted by course rule. Therefore, we removed that dependency and performed MVC pattern.
- Refactoring and dependency injection also caused technical debt.
- Had we planned effectively at first, we would have prevented all these issues. However, it was a learning curve for us to prevent these issues from happening in the future.

15. CONCLUSION

It has been a wonderful experience developing this project. Though it was not easy for us in the early stages, we were able to adapt as the project progressed. We realised how Test Driven Development was not a bane but a boon for us. We were able to find many errors early and correct it. Since we were working in agile environment, there was an opportunity for us to retrospect our work and discuss the difficulties faced. We were fortunate to have got this opportunity to develop a full stack project, implemented with different design patterns and principles.