**Trial Project: Frontend (BookMyAppointment)**

**You are allowed to use ChatGPT / any other Gen AI platforms to generate the Code.**

**This project uses following software’s:**

* **HTML, CSS and Bootstrap**
* **JavaScript**
* **ES6 and DOM Manipulation**
* **AJAX**
* **ReactJS**

**Overview**

**In this segment, we will take a deep dive into the project.**

**As you learnt in the previous segment, the application, at its core, will enable anyone to consult a healthcare specialist from anywhere in the world. As part of the application building process, you would need to add these functionalities to the front end:**

* **Landing page**
* **User registration**
* **User login/logout**
* **List of available appointments**
* **Doctors’ list**
* **Doctor details**
* **Booking an appointment**

**Your tasks would relate to writing code, which would solidify your understanding of the Reactjs framework even further.**

**Now, here is something you need to remember while working on the project: Do not get overwhelmed if you find it hard to progress at any point while trying to understand the problem statement. We have provided plenty of annotated comments inside the stub code.**

**These shall help you accomplish the major objectives of the project. All you need to do is follow the instructions mentioned here and carefully follow the comments in the stub code. So, with that being said, let’s get started!**

**In the next segment, you will get an understanding of the starter code and configuration details.**

**Code Structure**

**In this segment, you will get an overview of the project code.**

**The code structure and its purpose in life are as follows:**

* ***assets***
  + **Contains the icons to be used for the application**
* ***commons***
  + ***header***
    - ***Header.css*: Stylesheet for the common application header**
    - ***Header.js*: Contains the UI + JS for the common application header**
* ***screens***
  + ***appointment***
    - ***Appointment.js*: Contains the UI + JS for the appointment screen**
    - ***RateAppointment.js*: Contains the UI + JS for the rating screen**
  + ***doctorList***
    - ***BookAppointment.js*: Contains the UI + JS for booking an appointment**
    - ***DoctorDetails.js*: Contains the UI + JS for the Doctor details module**
    - ***DoctorList.js*: Contains the UI + JS for the Doctor list module**
  + ***home***
    - ***Home.css*: Stylesheet for the home screen**
    - ***Home.js*: Contains the UI + JS for the home screen**
  + ***login***
    - ***Login.js*: Contains the UI + JS for the login screen**
  + ***Register***
    - ***Register.js*: Contains the UI + JS for the registration screen**
  + ***Controller.js*: JS for handling routes in the application**
* ***util***
  + ***fetch.js*: JS for *GET* and *POST* requests to the back end**

**Moving ahead, in the next session, you will get an understanding of the tasks that you need to perform as part of this project.**

**Overview of Tasks**

**Welcome to the session on ‘Tasks and Approach’.**

**This session will help you understand the tasks that you would need to perform as part of this project. You will also get an understanding of the approach that you must follow to complete each individual task.**

**You would need to perform six tasks:**

* **Implement the header,**
* **Implement the home UI,**
* **Implement the login UI,**
* **Implement the registration UI,**
* **Implement the doctor list UI and**
* **Implement the appointment UI.**

**Moving ahead, in the next segment, you will get an understanding of the approach to complete the project setup task.**

**Starter Code and Configuration Details**

**The starter code is the code snippet that you must download for building the BookMyConsultation application. It will help you follow the appropriate configuration guidelines required for the application.**

**The zipped folder containing the starter code is provided below for downloading.**

1. **At all times, you need to ensure that your web application is configured properly. The application’s starter code has already been provided to you keeping the configuration guidelines in mind. You need to adhere to these guidelines whenever you make any change. The configuration guidelines for the project are given below:**
   * **‘Build’ folder**
   * **‘node\_modules’ folder (contains a list of all the packages)**
   * **‘public’ folder**
   * **‘src’ folder**
     + **‘assets’ folder**
     + **‘common’ folder**
     + **‘screens’ folder**
     + **‘Util’ folder**
     + **‘App.test.js’ file**
     + **‘index.css’ file**
     + **‘index.js’ file**
   * **‘package-lock.json’ file**
   * **‘package.json’ file**
   * **‘README.md’ file**
2. **You should always keep the following folders and the corresponding files inside the root directory of your application:**
3. **Remember to not modify the names of these folders and files and their locations inside the application. Doing so would make it cumbersome for the graders to evaluate your code based on the set evaluation rubrics. Also, you must take extra care to change only the required content of these folders and files. Failure to follow this guideline may prevent your application from running successfully, and this would also lead to it being evaluated incorrectly.**
4. **You must store all the common elements, components, stylesheets, scripts, etc. in the ‘common’ folder.**
5. **You must store all the resource files that you need to use in the application inside the ‘assets’ folder.**
6. **You must create and store all your pages (called screens) inside the ‘screens’ folder. For each page, you need to first create a folder by the name of that page. Inside this folder, you need to create one JavaScript file for each functionality. You may also include a stylesheet corresponding to that page.**
7. **For example, if you are creating the home page, then you can create a folder named ‘home’ inside the ‘screens’ folder. Inside this folder, you can create and store two files, ‘Home.js’ and ‘Home.css’, which are, respectively, the JavaScript and stylesheet files corresponding to the homepage of the application. Here is the folder hierarchy that you need to follow:**
   * **‘screens’ folder**

**‘home’ folder**

**‘Home.js’ file**

**‘Home.css’ file**

**Moving ahead, in the next segment, you will set up the backend.**

**Backend Set-Up**

**In this segment, you will set up the back end of the application.**

**For your application to be able to work end to end, you need to have a working back end. You will be using the same Capstone Backend project (Module named 'Capstone - Backend') that you have worked on and will be integrating it with this front end.**

**So, you need to go back to your back-end project and set it up again (if you haven't already). You can follow the steps mentioned in this segment as well.**

**You would need these tools to set up the back end of your application:**

* [**IDE**](https://airlock-on-edge.woolf.university/?url=https%3A%2F%2Fwww.eclipse.org%2Fdownloads%2F&resourceId=5649985&studentId=37746437-4865-4301-b4c6-2ec28dc342ab&token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjM3NzQ2NDM3LTQ4NjUtNDMwMS1iNGM2LTJlYzI4ZGMzNDJhYiIsImlzVmVyaWZpZWQiOnRydWUsImtpbmQiOiJvYXV0aCIsIm9yZyI6eyJncm91cHMiOltdLCJpZCI6Ijk3ZjhjNTRmLWVjZjctNGY1YS1iNGQ3LWM4NWEwMDI2ZGEwNyJ9LCJzY29wZSI6IioiLCJpYXQiOjE3NTAxMzI0NzcsImV4cCI6MTc1MDczNzI3NywiaXNzIjoidXJuOldvb2xmVW5pdmVyc2l0eTpzZXJ2ZXIvdXNlci9hY2Nlc3MifQ.O3c64N84ml_8K_kiL_ade27_lwaO7qXUcpCRf9_gjqg)
* [**MySQL Workbench**](https://airlock-on-edge.woolf.university/?url=https%3A%2F%2Fdev.mysql.com%2Fdownloads%2Fmysql%2F&resourceId=5649985&studentId=37746437-4865-4301-b4c6-2ec28dc342ab&token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjM3NzQ2NDM3LTQ4NjUtNDMwMS1iNGM2LTJlYzI4ZGMzNDJhYiIsImlzVmVyaWZpZWQiOnRydWUsImtpbmQiOiJvYXV0aCIsIm9yZyI6eyJncm91cHMiOltdLCJpZCI6Ijk3ZjhjNTRmLWVjZjctNGY1YS1iNGQ3LWM4NWEwMDI2ZGEwNyJ9LCJzY29wZSI6IioiLCJpYXQiOjE3NTAxMzI0NzcsImV4cCI6MTc1MDczNzI3NywiaXNzIjoidXJuOldvb2xmVW5pdmVyc2l0eTpzZXJ2ZXIvdXNlci9hY2Nlc3MifQ.O3c64N84ml_8K_kiL_ade27_lwaO7qXUcpCRf9_gjqg)
* **Java 11 SDK or higher (LTS version preferred)**
* **Apache Maven 3.3 build tool or higher**
* **Postman (optional)**
* **Coffee**

**Once you have the tools listed above, you can then proceed to start the setup process on your machine. You would need to follow these steps for this:**

1. **The first step is to import the project into your IDE.**
2. **Once you have imported the project, go to the file named ‘DBLoadScript.sql’ in this folder path: ‘src/main/resources’.**
3. **Inside this file, we have provided you with MySQL scripts.**
4. **These scripts will ensure that the database schema created on your machine is consistent with that of our graders.**
5. **Open Workbench and create a database named ‘consultation’, and then execute the scripts from the file ‘DBLoadScript.sql’ to create different tables and populate some of them with data.**
6. **You can use the ‘application.properties’ file inside the ‘src/main/resources’ to update the ‘username’ and the ‘password’ for the database you just created. You can also use the ‘application.properties’ file to add/edit other properties.**
7. **Next, go back to your IDE and build the project using the maven command ‘mvn clean install -DskipTests’. This will install all the dependencies required to run the project.**
8. **You can run the project using the command ‘mvn spring-boot:run’.**

**You can also refer to the file named ‘README.md’ in the path ‘src/main/resources’ for more information.**

**With the project set-up completed, you can now begin coding.**

**In the next segment, you will learn about the specifics of the header task.**

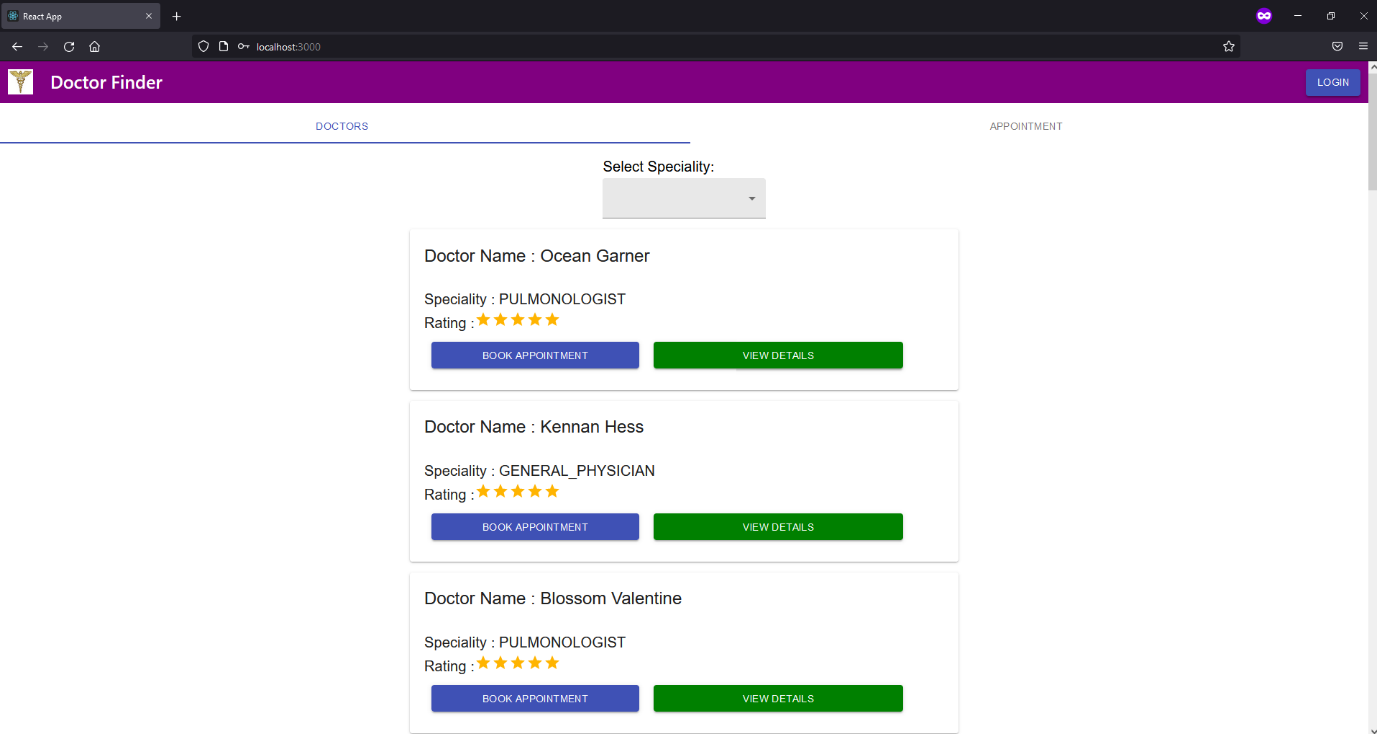
**Header**

**In this segment, you will learn about the specifics of the header task.**

**The header is the part that is present throughout an application. Only its components change depending upon the situation.**

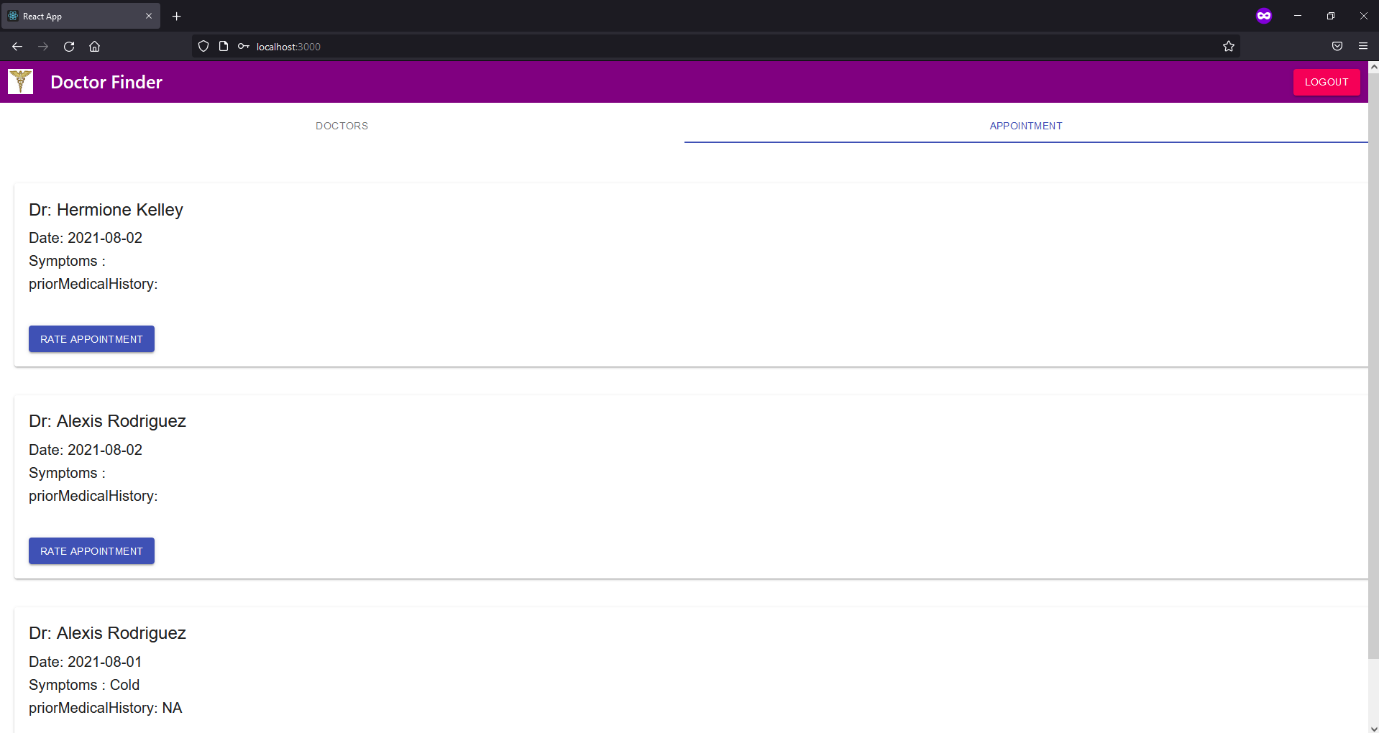
**In the images below, you can see how the header of the application changed when a user was logged in as opposed to when they were not.**

**Before logging in, the header looks like this.**

****

**Header Before**

**Now, this image shows what the header looks like after a user has logged in.**

****

**Header After**

**Here is a list of the points that you need to remember with regard to the header:**

* **You must create a folder named ‘header’ inside the ‘common’ folder. In the folder ‘header’, you need to create and store the files ‘Header.js’ and ‘Header.css’.**
* **The background colour of the header must be purple.**
* **It should have a height of 70 px.**
* **It must have a padding of 11 px.**

**Guidelines for the Logo :**

**First, you need to use this as the logo for the application.**

****

**Logo**

**You are provided with a JPEG image named logo.jpeg inside the folder ‘assets’. You must include this in the header with the following style properties:**

* **Its background colour should be #ff7f7f.**
* **It should have a height of 35 px.**
* **It should be on the left in the header.**

**Next, we have provided the guidelines for the Login/Logout button:**

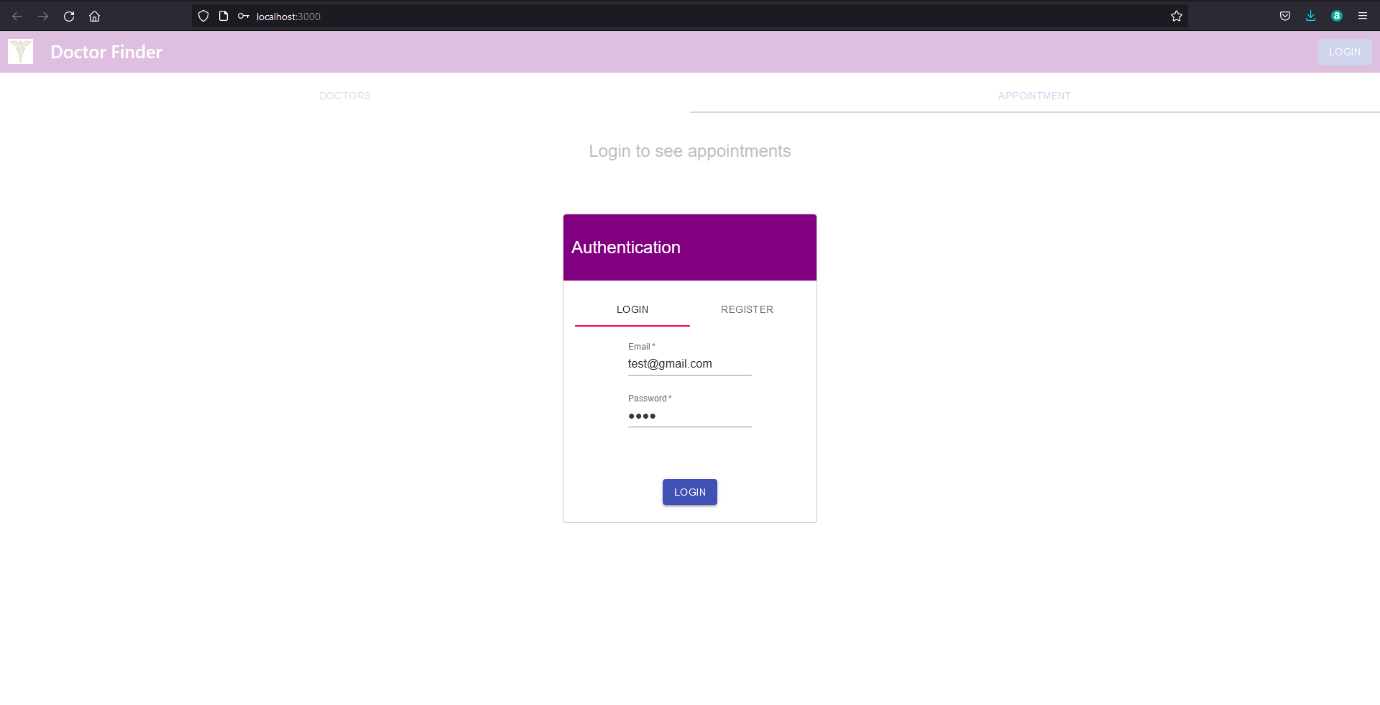
* **You must use the button component of the Material UI. Its variant should be ‘contained’. Its colour should be primary for the login button and secondary for the logout button. The name for the login button should be Login, and the name for the logout button should be Logout. It should be on the right in the header.**
* **When a user clicks on the login button, a modal containing a card with two tabs must be displayed: a ‘LOGIN’ tab for logging a registered user into the application and a ‘REGISTER’ tab for getting a user registered on it. The code for the modal must be implemented using the**[**react-modal**](https://airlock-on-edge.woolf.university/?url=https%3A%2F%2Fwww.npmjs.com%2Fpackage%2Freact-modal&resourceId=5649986&studentId=37746437-4865-4301-b4c6-2ec28dc342ab&token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjM3NzQ2NDM3LTQ4NjUtNDMwMS1iNGM2LTJlYzI4ZGMzNDJhYiIsImlzVmVyaWZpZWQiOnRydWUsImtpbmQiOiJvYXV0aCIsIm9yZyI6eyJncm91cHMiOltdLCJpZCI6Ijk3ZjhjNTRmLWVjZjctNGY1YS1iNGQ3LWM4NWEwMDI2ZGEwNyJ9LCJzY29wZSI6IioiLCJpYXQiOjE3NTAxMzI0NzcsImV4cCI6MTc1MDczNzI3NywiaXNzIjoidXJuOldvb2xmVW5pdmVyc2l0eTpzZXJ2ZXIvdXNlci9hY2Nlc3MifQ.O3c64N84ml_8K_kiL_ade27_lwaO7qXUcpCRf9_gjqg)**component, and the code for the tabs must be implemented using the Material-UI ‘Cards’ and ‘Tabs’ components.**

**Moving ahead, the next segment will help you understand how to approach the login UI task.**

**Login**

**In the previous segment, you created a header, to which you added a ‘LOGIN’ button. You also learnt that clicking on the ‘LOGIN’ button opens a modal with two tabs: ‘LOGIN’ and ‘REGISTER’.**

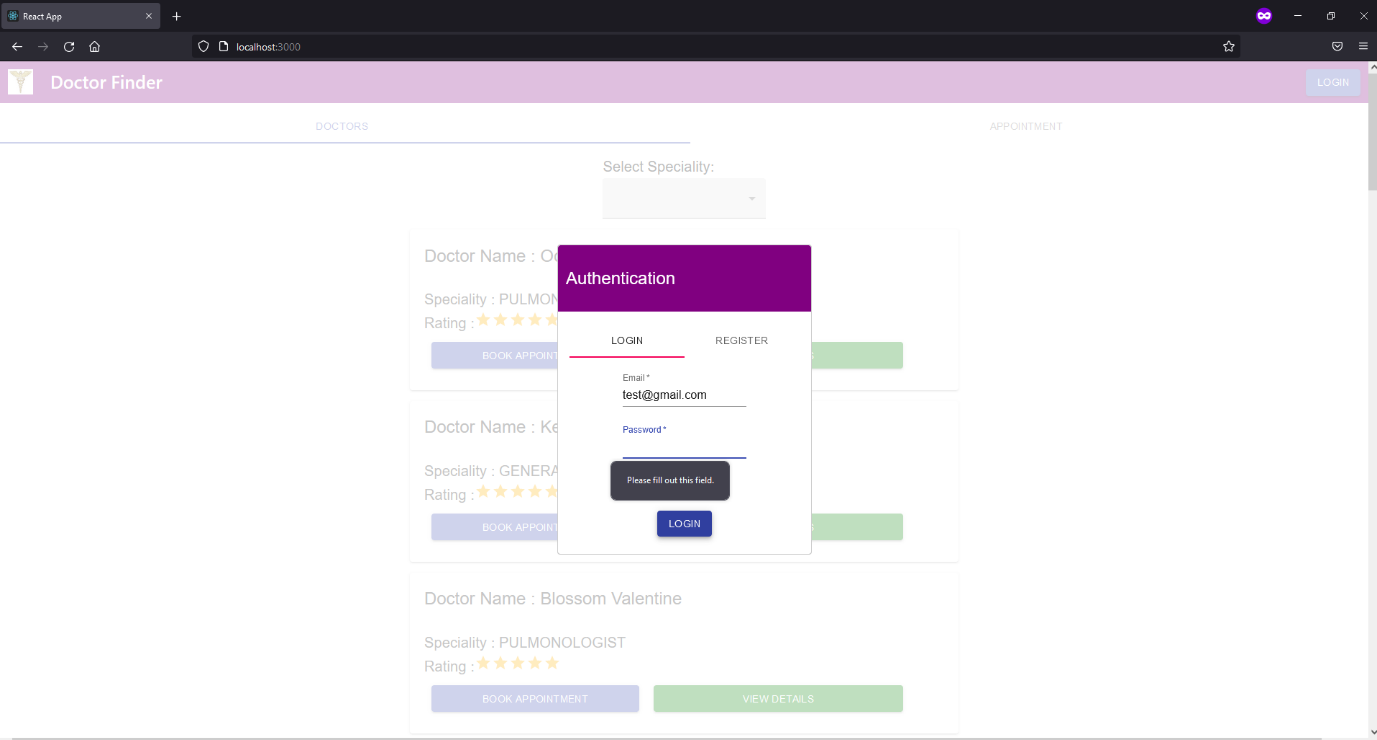
**Now, in this segment, you will implement the ‘LOGIN’ functionality. The ‘LOGIN’ UI should look like this.**

****

**Login**

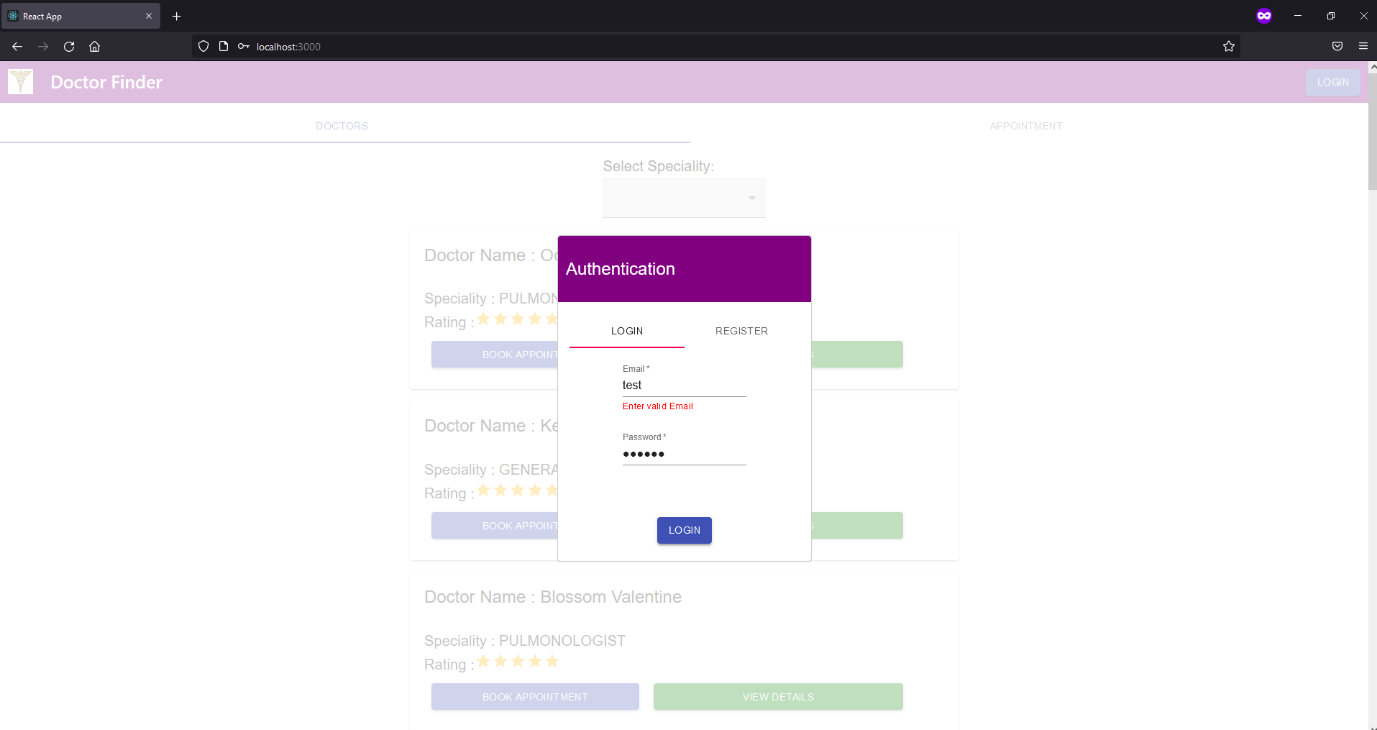
**To create the 'LOGIN' tab, you must follow these guidelines:**

* **The 'LOGIN' tab should have two input fields: Email and password. Upon clicking the 'LOGIN' button, users should be able to log in when they provide the correct username and password.**
* **The code for each text field can be implemented using the Material-UI ‘FormControl’, ‘InputLabel’ and ‘Input’ components.**
* **The login form must also have the required field validation. This means that if a user clicks on the 'LOGIN' button without entering a value in either of the required fields, then the application should display the text ‘Please fill out this field’ underneath the required text field(s) in which no value has been entered. This screenshot will help you understand this better.**

****

**Login Error**

* **When a user enters an invalid email and clicks on the Login button, the message “Enter valid Email” in red must be displayed below the Email field, as shown in the screenshot below. The message should disappear when the user starts retyping their email.**

****

**Login Error**

**Below the two text fields, there must be a login button with the text ‘LOGIN’. The code for this button must be implemented using the Material-UI ‘Button’ component, with the variant as ‘contained’ and colour as ‘primary’. When a user clicks on the 'LOGIN' button, the following API endpoint, which you created inside AuthenticationController in your back-end project, must be used to log them into the application:**

**'/auth/login'**

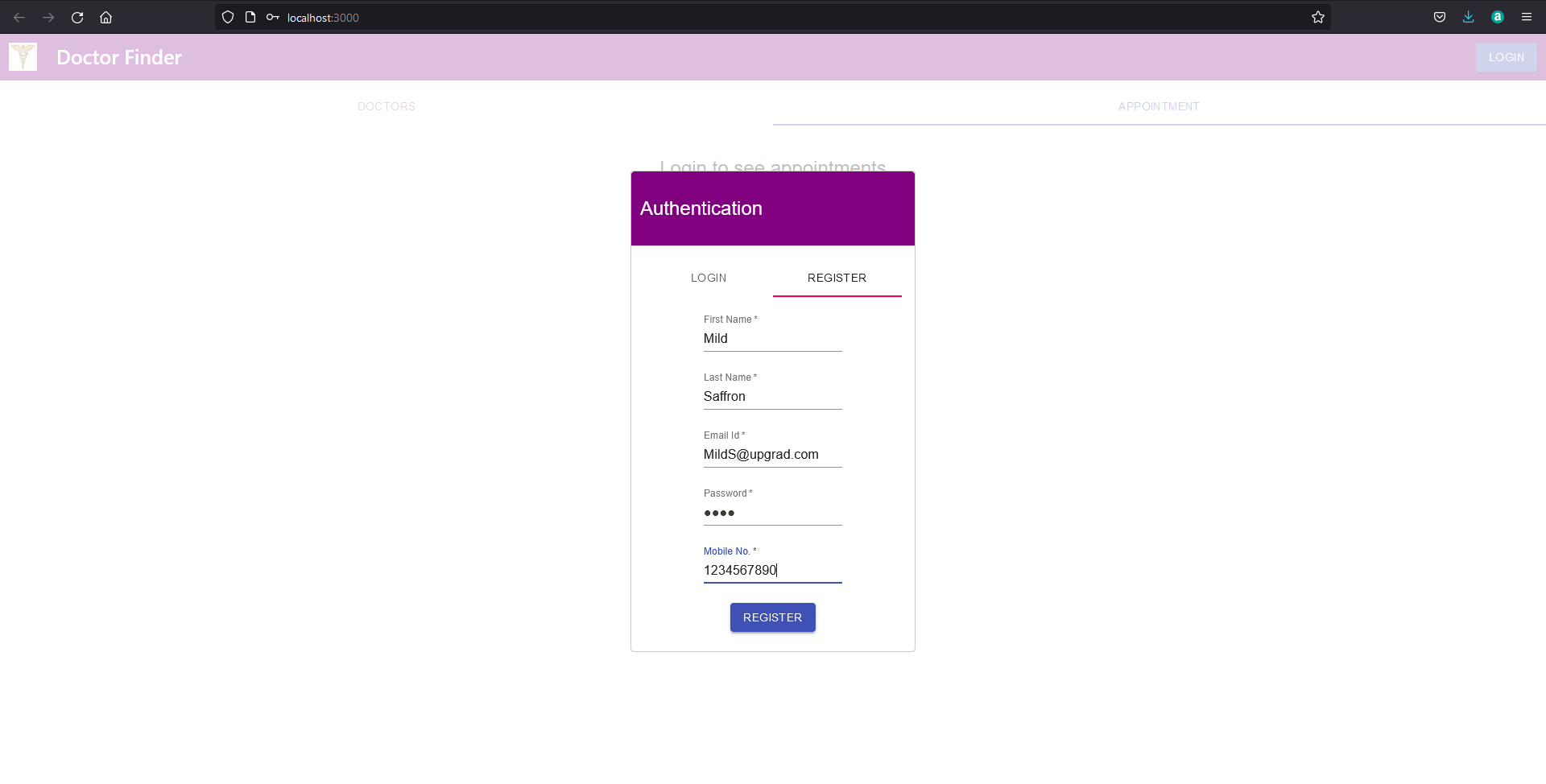
**If the text entered by the user in both the fields is valid and the corresponding information exists in the database, and if the user then clicks on the 'LOGIN' button, then they must be able to log in successfully. The modal must close and the 'LOGOUT' button must replace the 'LOGIN' button inside the header.**

**In the screenshot below, the text entered in both the text fields is valid information, and the user is logged into the application.**

**Registration**

**In the previous segment, you implemented the ‘LOGIN’ functionality.**

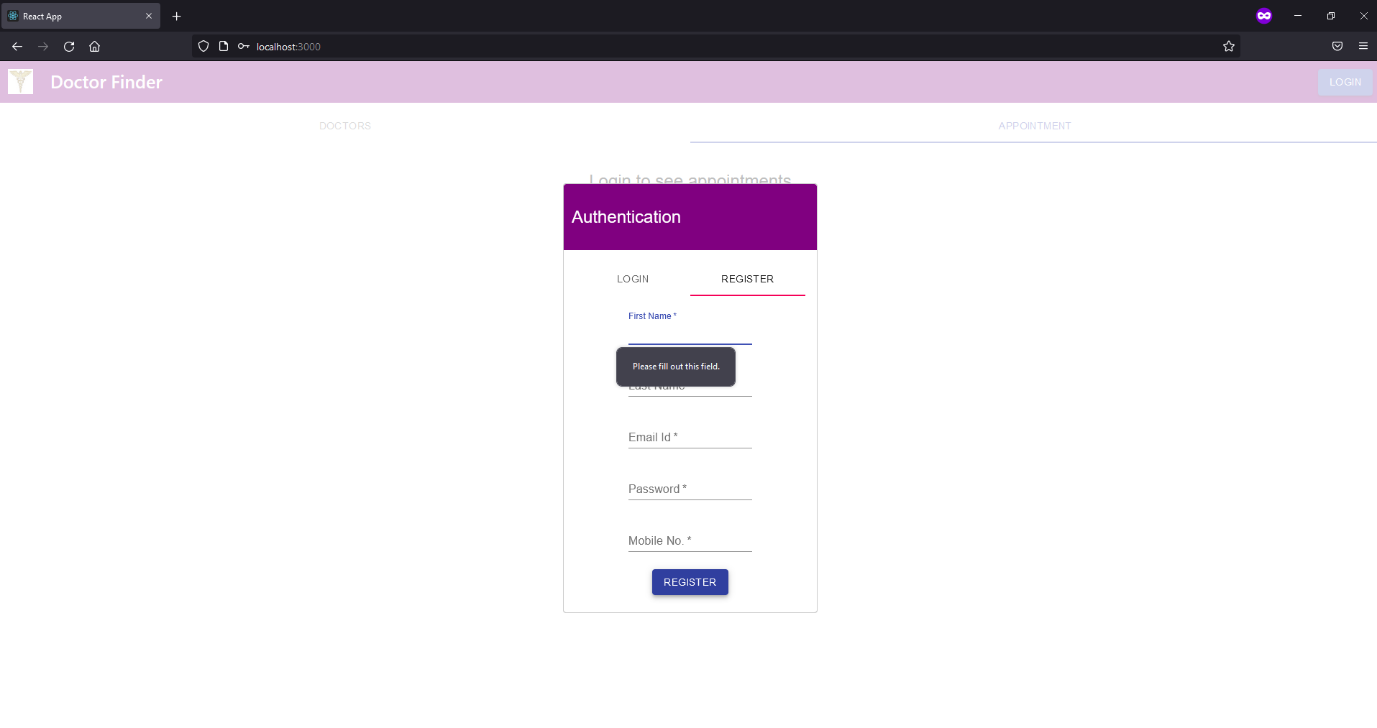
**Now, in this segment, you will implement the ‘REGISTER’ functionality. The ‘REGISTER’ UI should look like this.**

****

**Register**

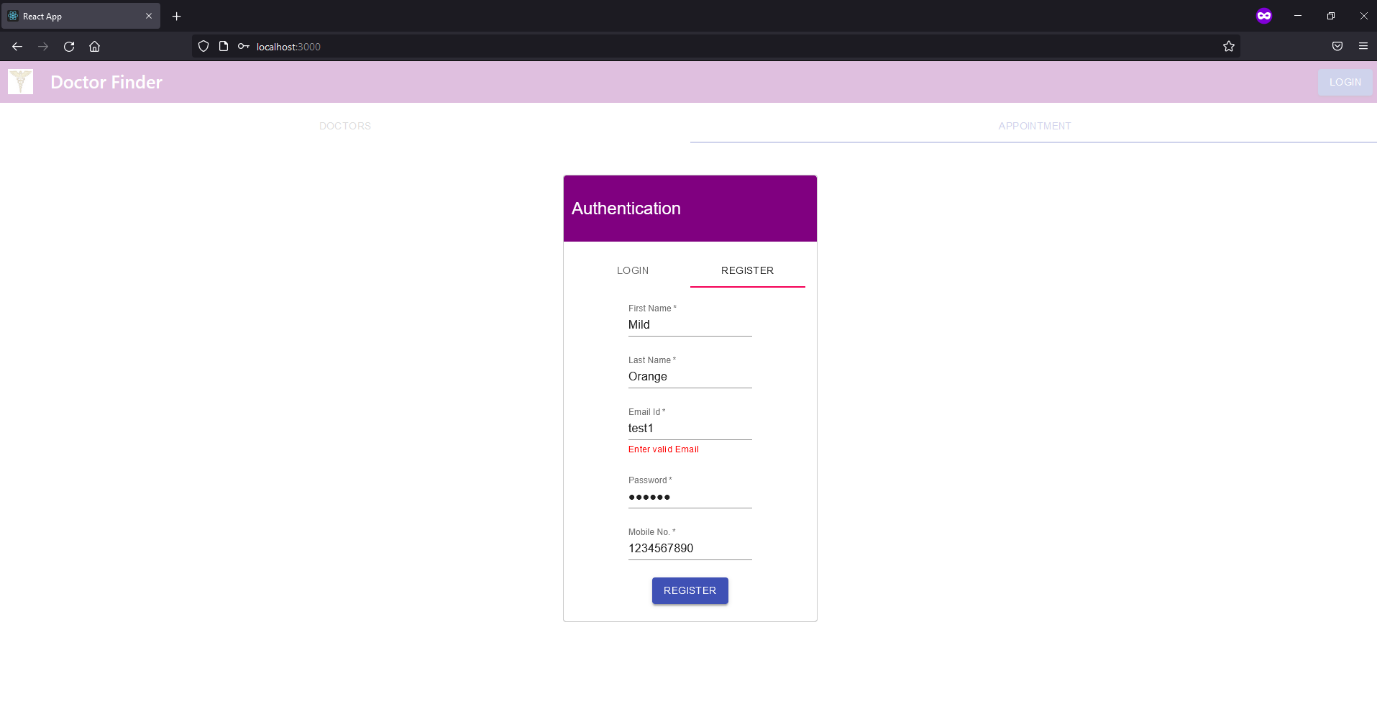
**To create the ‘REGISTER’ tab, you must follow these guidelines:**

* **The ‘REGISTER’ tab should have the following fields:**
  + **First name**
  + **Last name**
  + **Email**
  + **Password**
  + **Contact number**
* **If no information is entered in these fields, then the application should display the error message ‘Please fill out this field’. This is shown in the image given below.**
* **The code for each text field can be implemented using the Material-UI ‘FormControl’, ‘InputLabel’ and ‘Input’ components.**
* **The ‘REGISTER’ form must also have the required field validation. This means that if a user clicks on the ‘REGISTER’ button without entering any value in any of the required fields, then the application must display the message ‘Please fill out this field’ underneath the required text field(s) in which no value has been entered. The screenshot below will make this clearer for you.**

****

**Register Error**

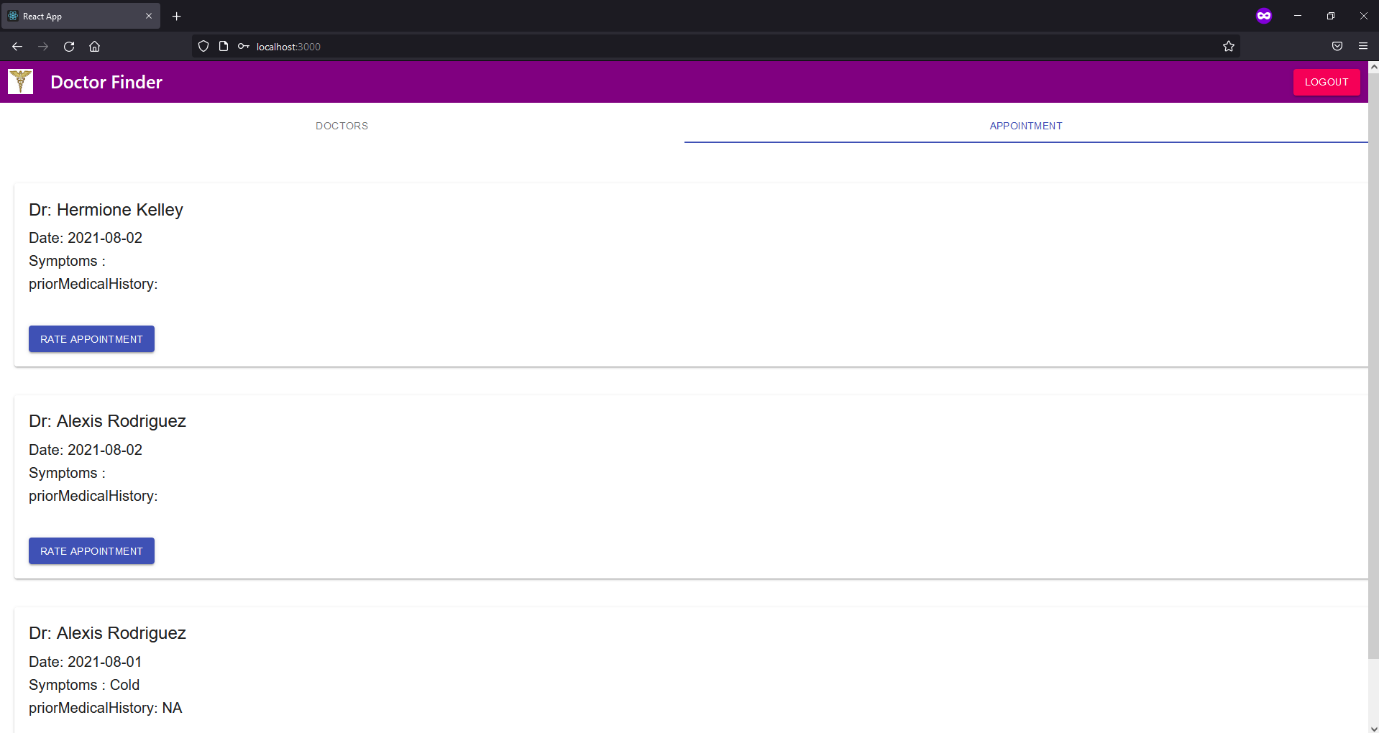
* **When a user enters an invalid email and clicks on the ‘REGISTER’ button, the message ‘Enter valid Email’ in red must be displayed below the Email field, as shown in the screenshot below. The message should disappear when the user starts retyping their email.**

****

**Register Error**

* **Similarly, when a user enters an invalid mobile number and clicks on the ‘REGISTER’ button, the message ‘Enter valid mobile number' in red must be displayed below the field 'Mobile No.'. The message should disappear when they start retyping their mobile number.**
* **Below the text fields, there must be a 'REGISTER' button with the text ‘REGISTER’. The code for this button must be implemented using the Material-UI ‘Button’ component, with the variant as ‘contained’ and colour as ‘primary’. When a user clicks on the 'LOGIN' button, the following API endpoint, which you created inside 'UserAdminController' in your back-end project, must be used to log them into the application: '/users/register'**
* **If the text entered by the user in all the fields is valid and they click on the 'REGISTER' button, they must get registered. The application should display the message ‘Registration Successful’ and log the user in.**

**Moving ahead, the next segment will help you understand your approach to the home UI task.**

****

**Login**

**Moving ahead, in the next segment, you will get an understanding of how to approach the registration UI task.**

**Home**

**The home page should be the first page to be displayed. This means when the application is run on the local development server at port 3000, the home page should be the first page that the user sees.**

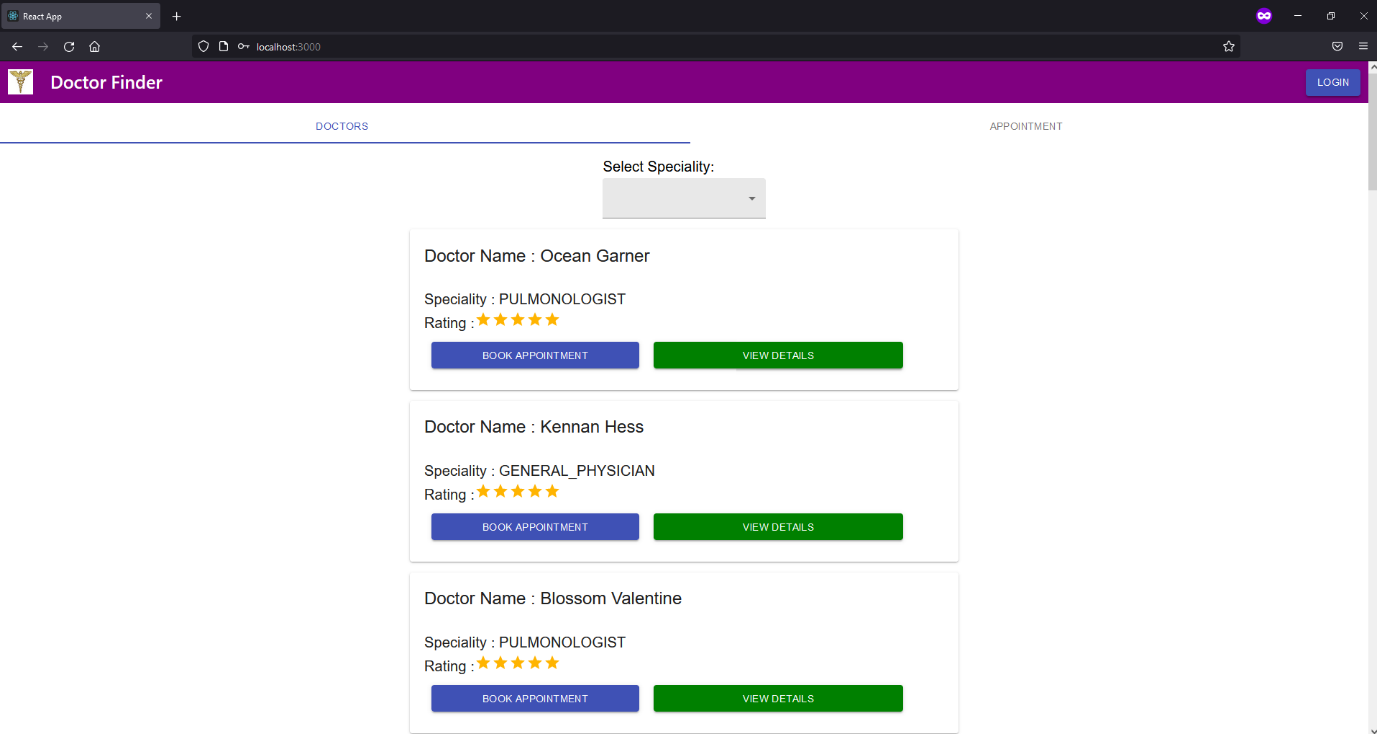
**You must create a folder named ‘home’ inside the ‘screens’ folder. Inside the ‘home’ folder, you need to create and store the files ‘Home.js’ and ‘Home.css’.**

**The home page of the application contains two tabs.**

**The first tab, named ‘DOCTORS’, displays a list of doctors. It also displays filters, which allow users to search for a specific healthcare specialist based on certain predefined parameters. For each doctor, there are two buttons: The ‘BOOK APPOINTMENT’ button to book appointments and the ‘VIEW DETAILS’ button to view the details of specific doctors.**

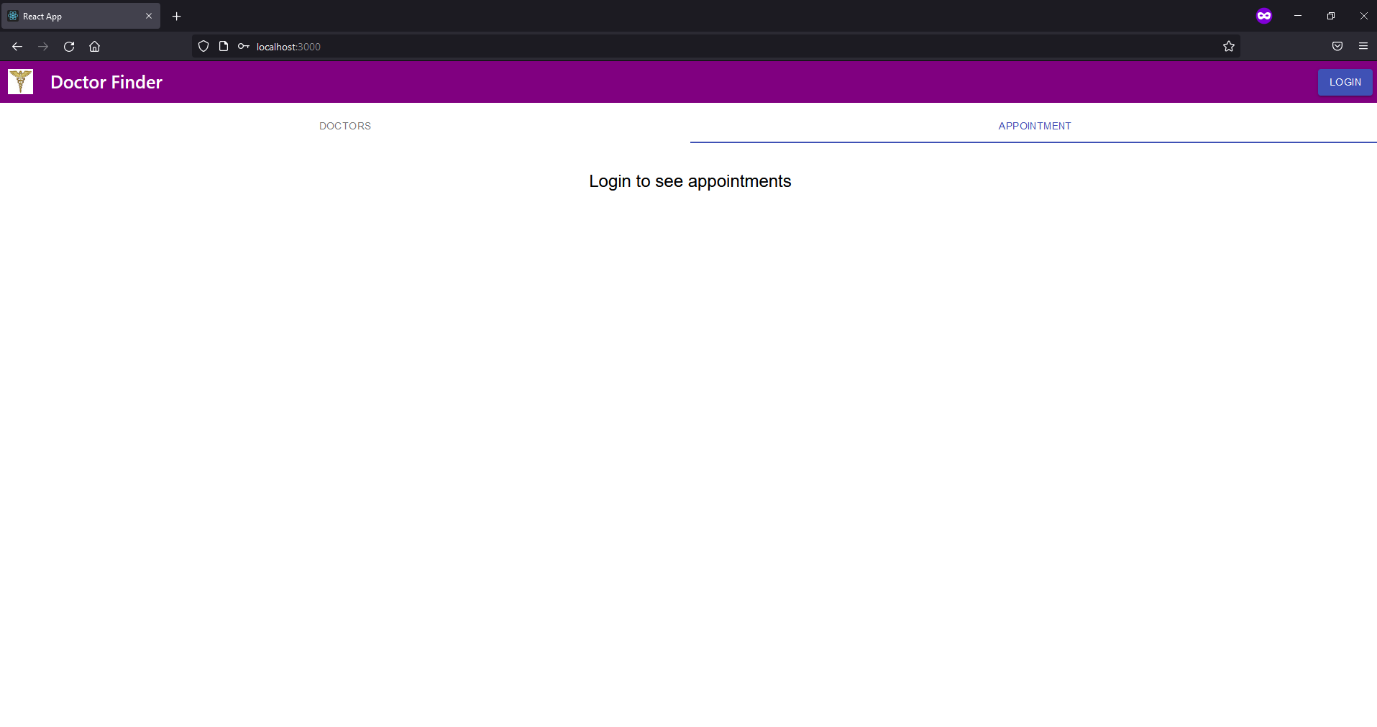
**The second tab, named ‘APPOINTMENT’, is where appointments will be visible once a user logs in.**

**Essentially, this is what the entire home page (with the ‘DOCTORS’ tab) should look like.**

****

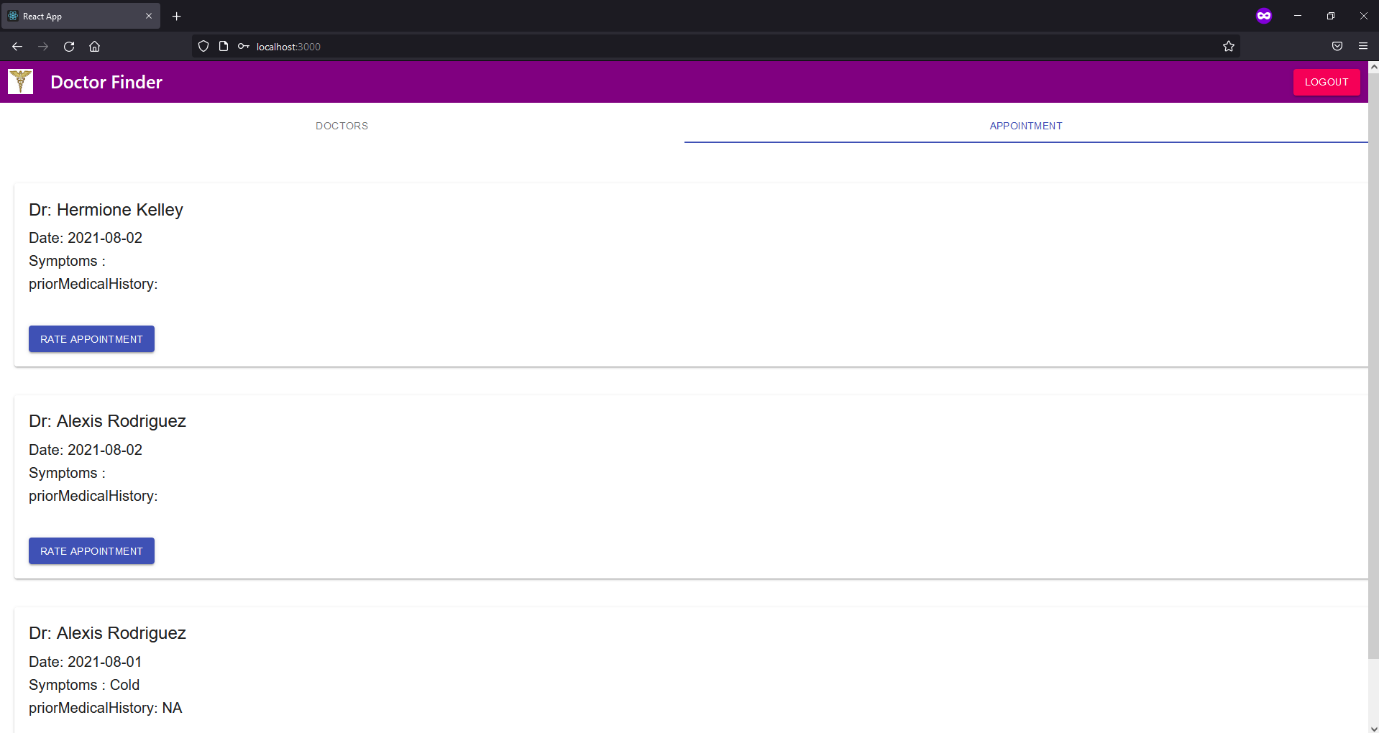
**Home**

**The entire home page (with the ‘APPOINTMENT’ tab) should look like this before a user has logged in.**

****

**Appointment Before**

**The image below shows what the entire home page (with the ‘APPOINTMENT’ tab) should look like after a user has logged in. The ‘APPOINTMENT’ tab should display a list of the appointments that a user has booked.**

****

**Login after**

**You must use the Material-UI ‘Tabs’ and ‘Tab’ components for creating the HOME UI.**

**Moving ahead, in the next segment, you will get an understanding of your approach to the doctor list UI task.**

**Doctor List**

**In the previous segment, you saw that the home page of the application contains two tabs.**

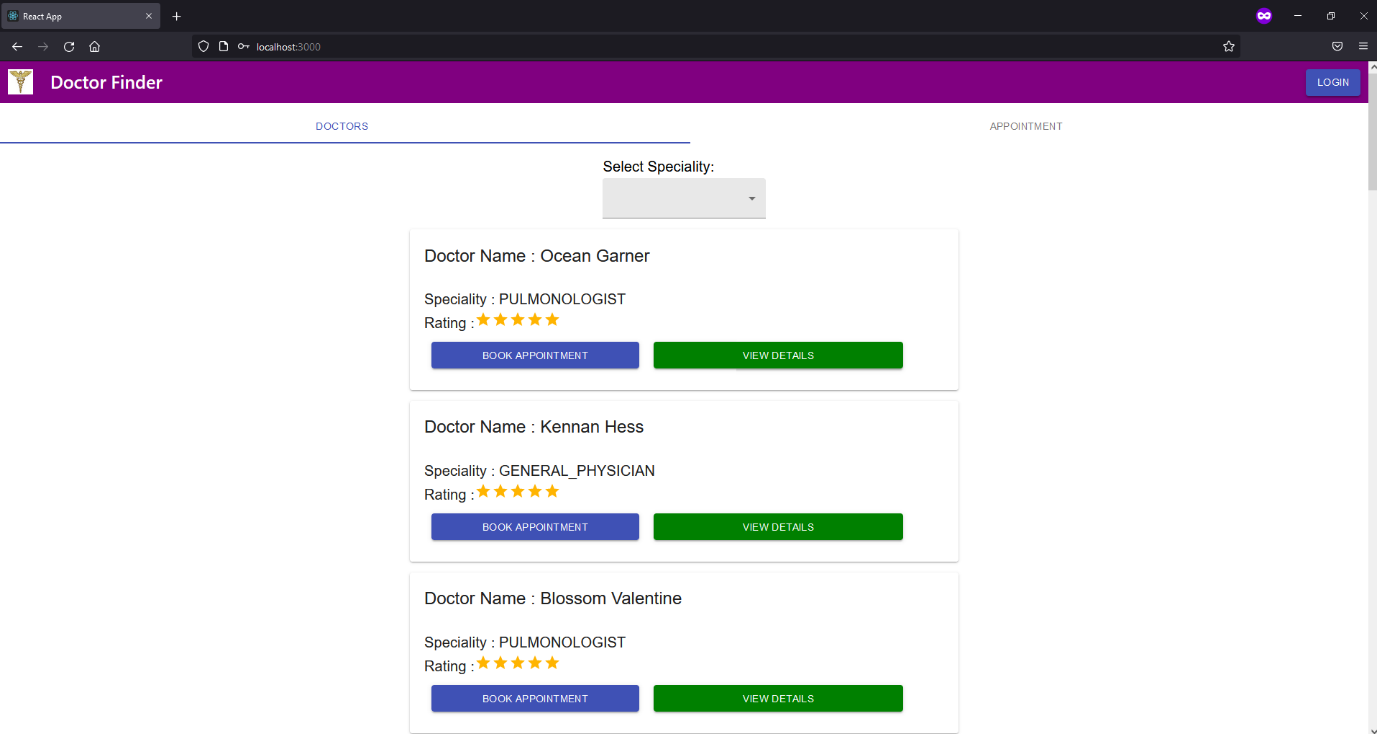
**The first tab, named ‘DOCTORS’, displays a list of doctors and the second tab, named ‘APPOINTMENT’, is where a user will be able to see their appointments once they log in.**

**Now, in this segment, you will get an understanding of how to build the ‘DOCTORS’ tab.**

**First, you must create a folder named ‘doctorList’ inside the ‘screens’ folder. Next, inside the ‘doctorList’ folder, you need to create and store three files:**

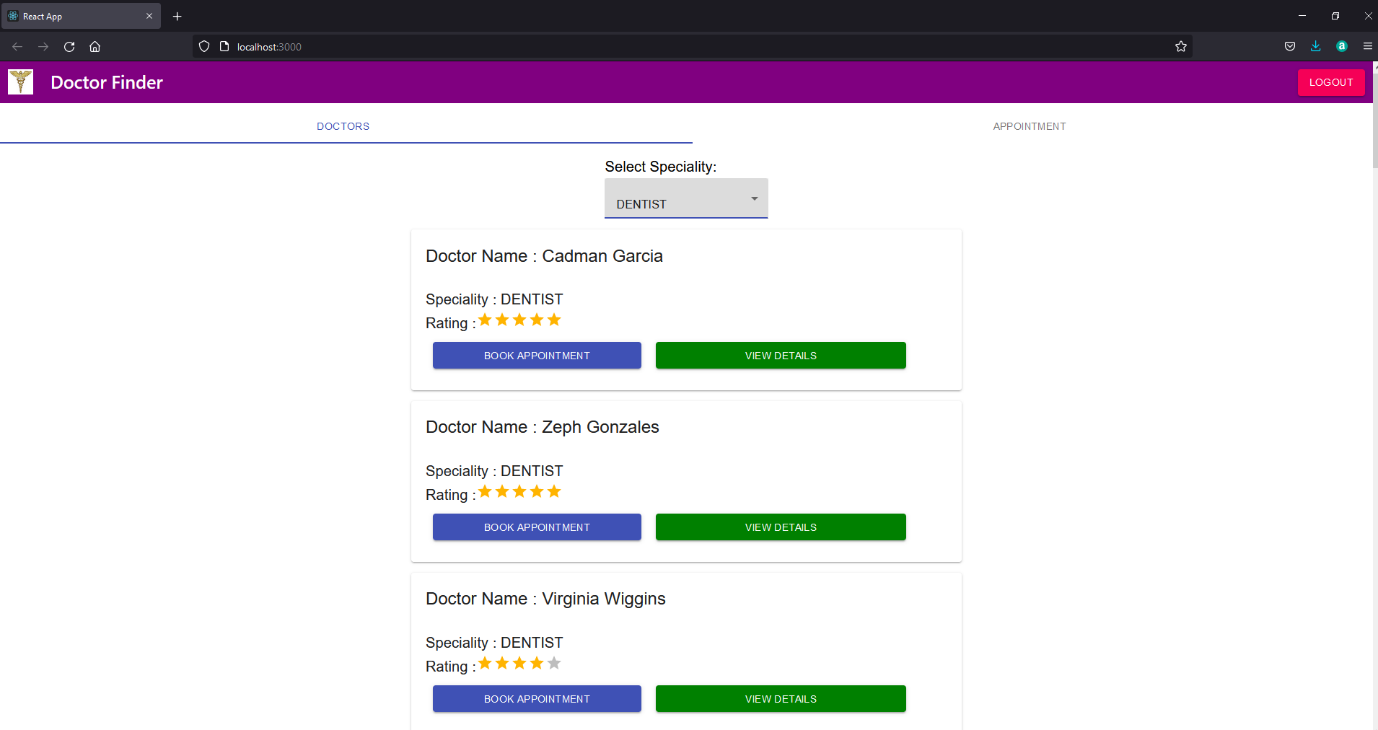
* **Doctorlist.js,**
* **BookAppointment.js and**
* **DoctorDetails.js.**

**The ‘DOCTORS’ tab should look somewhat like this.**

****

**Doctor Tab - 1**

**The UI should contain a drop-down, which would allow a user to select a speciality for viewing a list of the corresponding specialists. Once a speciality is selected, the list of doctors should get filtered based on it. It should look somewhat like this.**

****

**Doctor tab - 2**

**In the image above, you can see that the ‘DOCTORS’ tab has a list. Each item of that list contains this information:**

* **Doctor’s name**
* **Doctor’s speciality**
* **Doctor’s average rating**
* **Book Appointment button**
* **View Details button**

**Each item of that list can be created using the ‘Paper’ component. You should use the ‘Typography’ component to ensure uniformity in the theme of the application. The ‘BOOK APPOINTMENT’ and ‘VIEW DETAILS’ buttons should be created using the ‘Button’ component.**

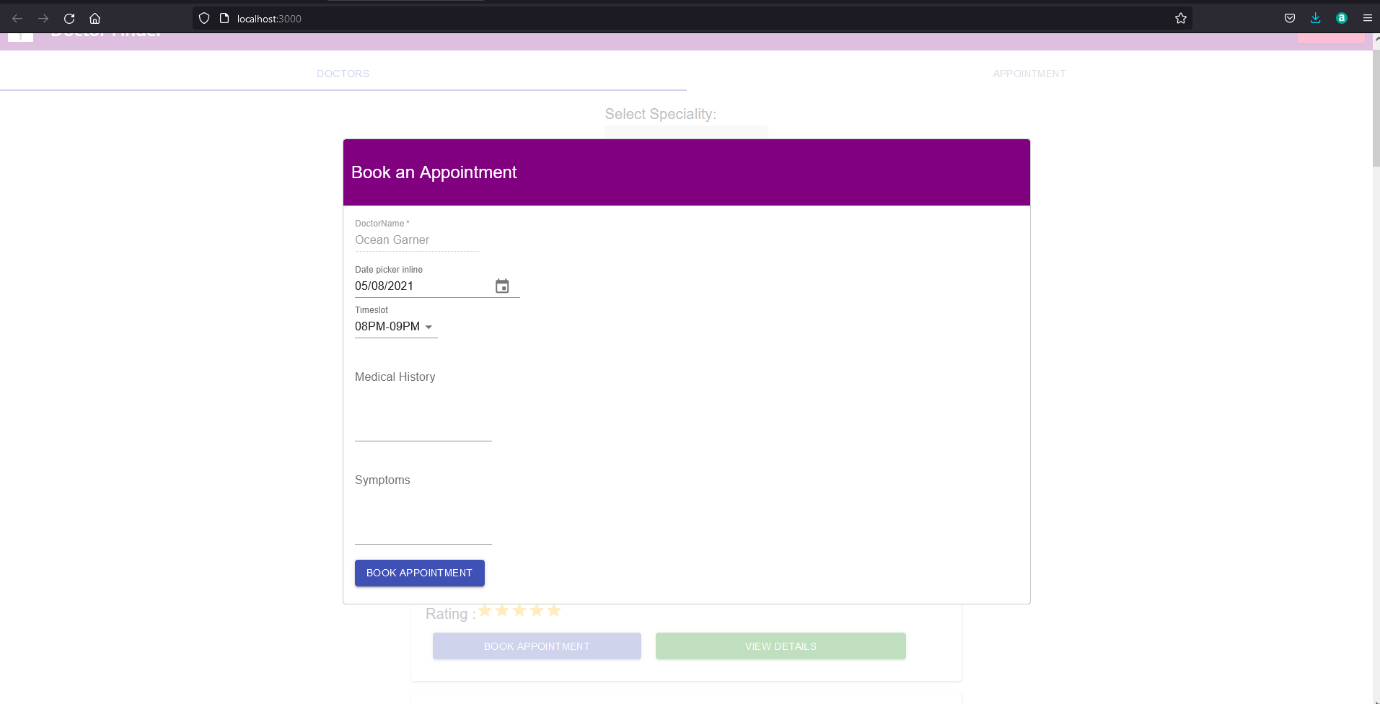
**You will be writing the functionality of the ‘BOOK APPOINTMENT’ button in the ‘BookAppointment.js’ file. When a user clicks on the ‘BOOK APPOINTMENT’ button, a modal containing a card component should open. The card component should contain the header ‘Book an Appointment’. The card content should include these details:**

* **A text field containing the doctor’s name, which the user cannot edit. You can use the 'TextField' component to create this text field.**
* **A component for the user to select a date for the appointment. You can use the 'MuiPickersUtilsProvider' component to create the date picker.**
* **Next, you need to add a component that would allow the user to select a time slot for the appointment. The time slot component can be created using the ‘Select’ and ‘MenuItem’ components.**
* **Finally, you should add two text fields: ‘Medical History’, for a user to input their medical history, and ‘Symptoms’, for a user to input their symptoms. You can use the ‘FormControl’ and ‘TextField’ components to create these text fields.**

**The CSS rules that you should follow for the ‘DOCTORS’ tab are as follows:**

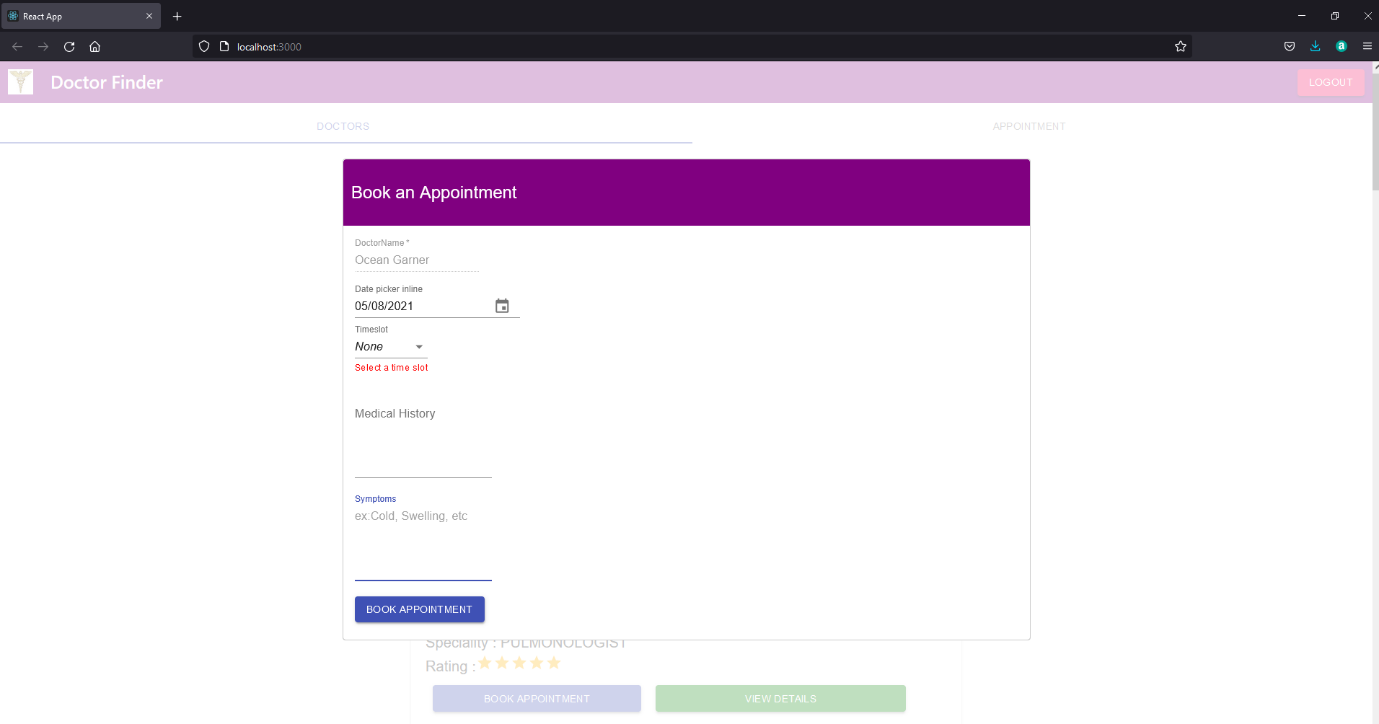
* **The doctor list should have a width of 40%.**
* **The ‘BOOK APPOINTMENT’ and ‘VIEW DETAILS’ buttons should have a width value of 40% and margin of 10 px. The ‘VIEW DETAILS’ button should have green as the background colour.**

**The UI for booking an appointment should look like this.**

****

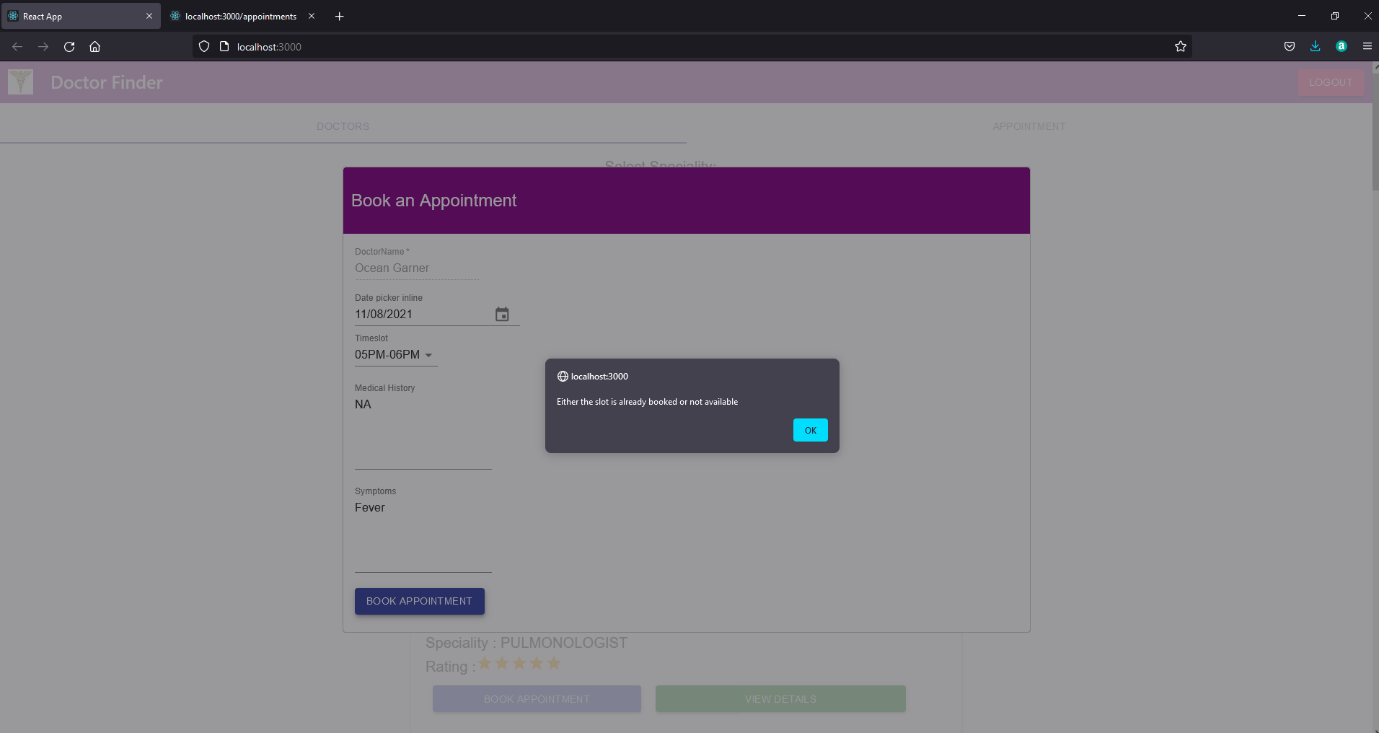
**Book Appointment**

* **The date picker should, by default, be set to the date on which a user is booking an appointment (i.e., today's date). If the user does not select a time slot, then the text ‘Select a time slot’ in red should be displayed right below the Time slot field, and it should look like this.**

****

**Book Appointment - 2**

* **When a user tries to book an appointment for a date–time slot for which they already have an appointment, the application should display the message ‘Either the slot is already booked or not available’. This image will make this clearer for you.**

****

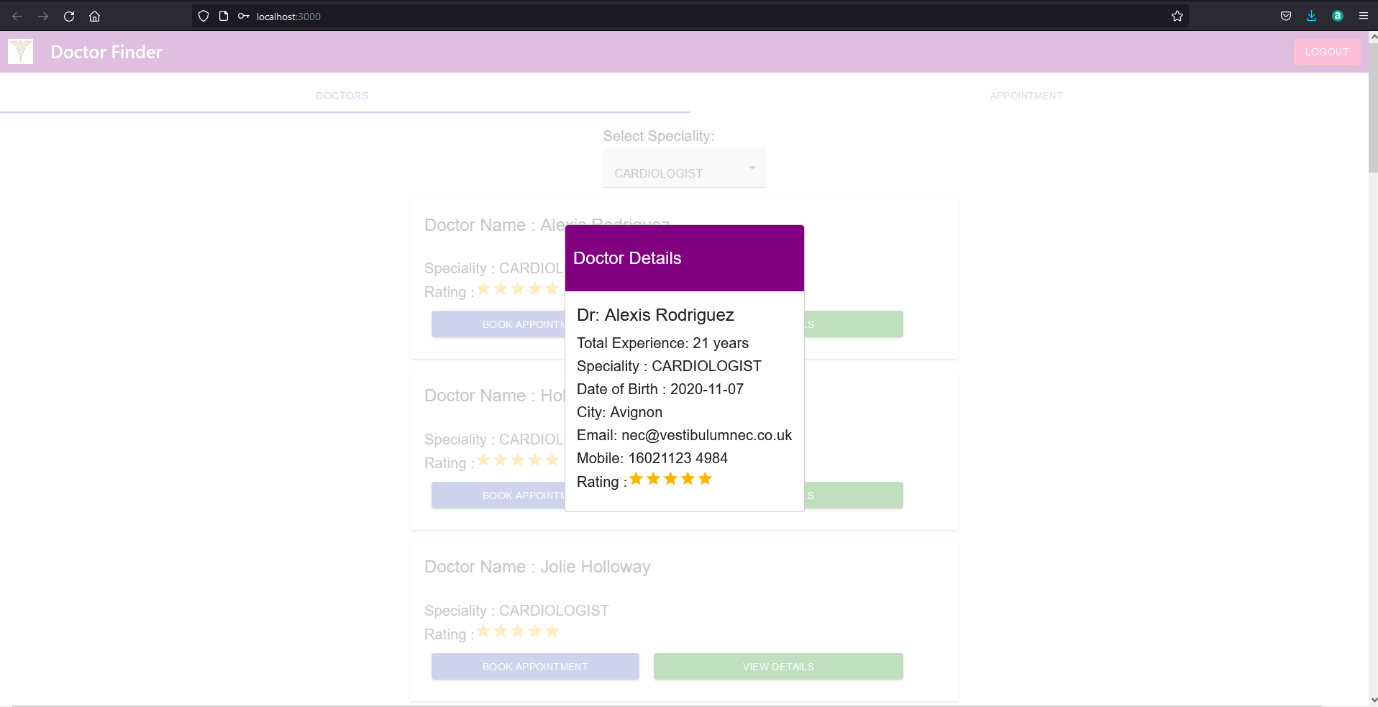
**Book Appointment - 3**

* **The ‘Medical history’ and ‘Symptoms’ fields are optional.**

**Next, you will be writing the functionality of the ‘VIEW DETAILS’ button in the ‘doctordetails.js’ file. When a user clicks on the ‘VIEW DETAILS’ button, a modal containing a card component should open. The card component should contain the header ‘Doctor Details’. The card content should include these details:**

* **Doctor’s name**
* **Total Experience: Doctor’s total experience**
* **Speciality: Doctor’s speciality**
* **Date of Birth: Doctor’s date of birth**
* **City: The city where the doctor lives**
* **Email: Email address of the doctor**
* **Mobile: Phone number of the doctor**
* **Rating: Average rating of the doctor**

**The UI for 'VIEW DETAILS' should look like this:**

****

**Doc details**

**You can use the ‘Typography’ component to ensure consistency in the text being displayed inside the card.**

**The stylesheet for 'BookAppointment' and 'DoctorDetails' can be as described below.**

**The CSS rules for the 'Paper’ component can be as follows:**

* **The text should be left-aligned.**
* **The margin value should be 15 px.**
* **Padding should be 20 px.**
* **The cursor type should be a pointer.**

**The CSS rules for the card header can be as follows:**

* **The background colour should be purple.**
* **Height should be 70 px.**
* **Padding should be 11 px.**
* **The colour should be white.**

**Moving ahead, the next segment will help you understand your approach to the appointment tab UI task.**

**Appointment**

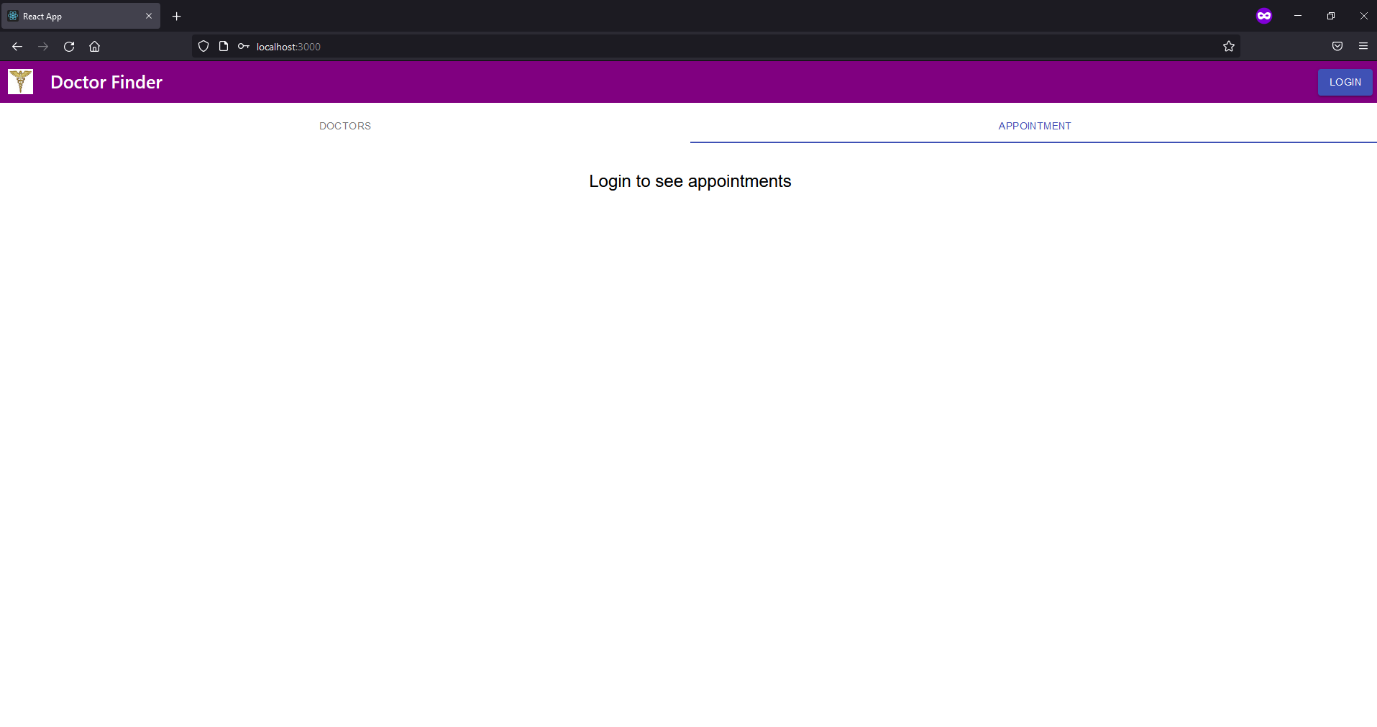
**In the previous segment, you learnt about the specifications for the ‘DOCTORS’ tab.**

**Now, in this segment, you will know more about the ‘APPOINTMENT’ tab.**

**First, you must create a folder named ‘appointment’ inside the ‘screens’ folder.**

**Next, inside the ‘appointment’ folder, you need to create and store the files ‘Appointment.js’ and ‘RateAppointment.js’. The file ‘Appointment.js’ would have the code for the 'APPOINTMENT' page, while the file ‘RateAppointment.js’ would have the code for rating appointments.**

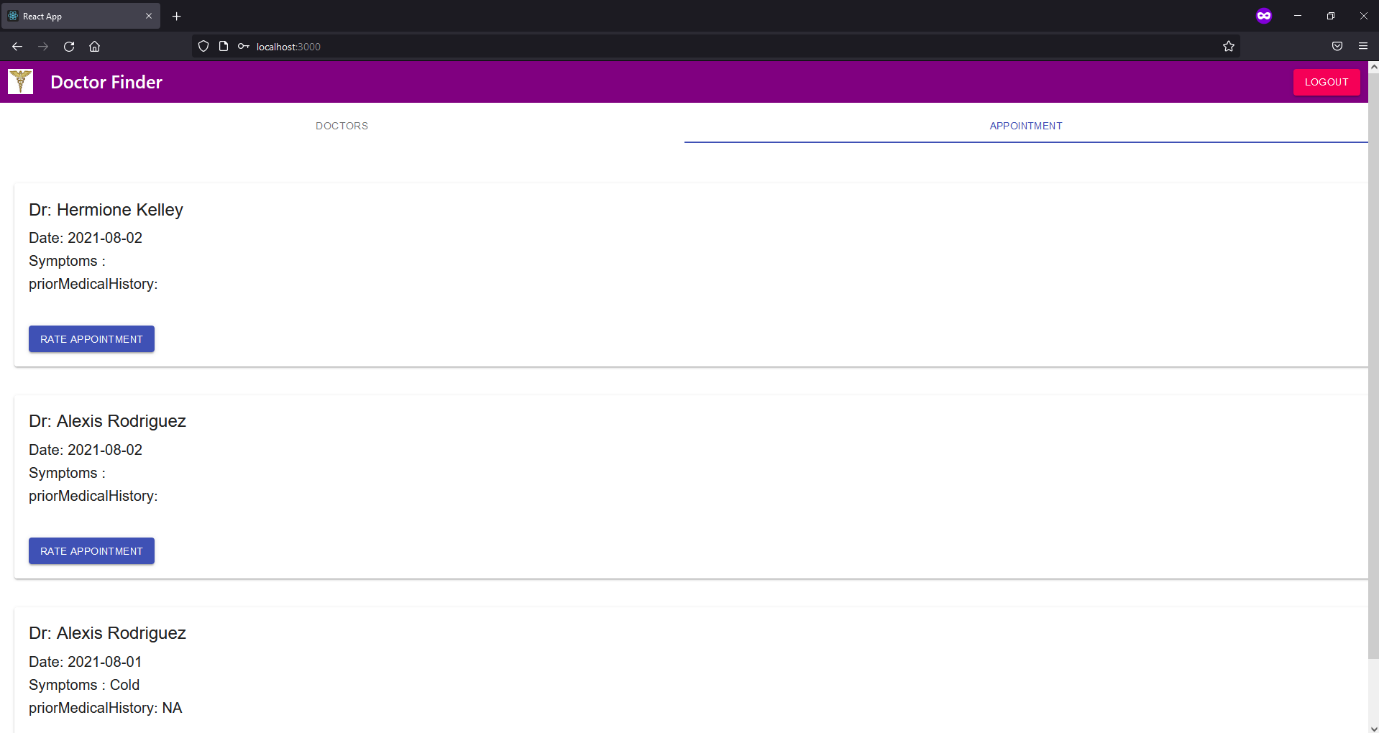
**The ‘APPOINTMENT’ tab should look somewhat like this before a user logs in.**

****

**Appointment - 1**

**It should display the message ‘Login to see appointments’ at the top.**

**The ‘APPOINTMENT’ tab should look somewhat like this after a user has logged in.**

****

**Appointment - 2**

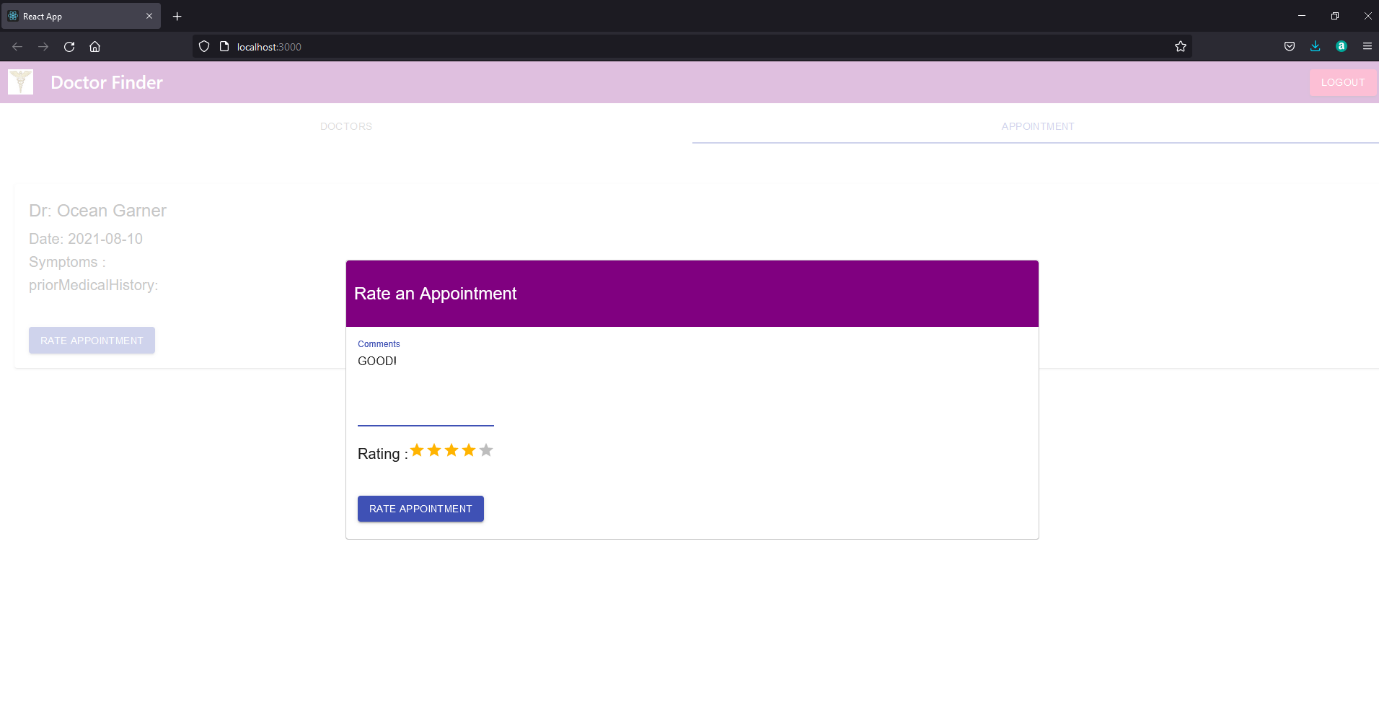
**Once a user logs in, the ‘APPOINTMENT’ tab should display a list of appointments, with each appointment being a paper component. Each item in the list should contain these details:**

* **It should have a doctor’s name.**
* **It should have an appointment date.**
* **It should display symptoms (if any).**
* **It should display the users’ previous medical history (if any).**
* **It should have a ‘RATE APPOINTMENT’ button.**

**Next, you will be writing the functionality of the ‘RATE APPOINTMENT’ button in the ‘RateAppointment.js’ file. When a user clicks on the ‘RATE APPOINTMENT’ button, a modal containing a card component should open. The card component should contain the header ‘Rate an Appointment’. The card content should include:**

* **Comments,**
* **The rating functionality and**
* **The ‘RATE APPOINTMENT’ button.**

**The ‘Rate an Appointment’ UI should look like this.**

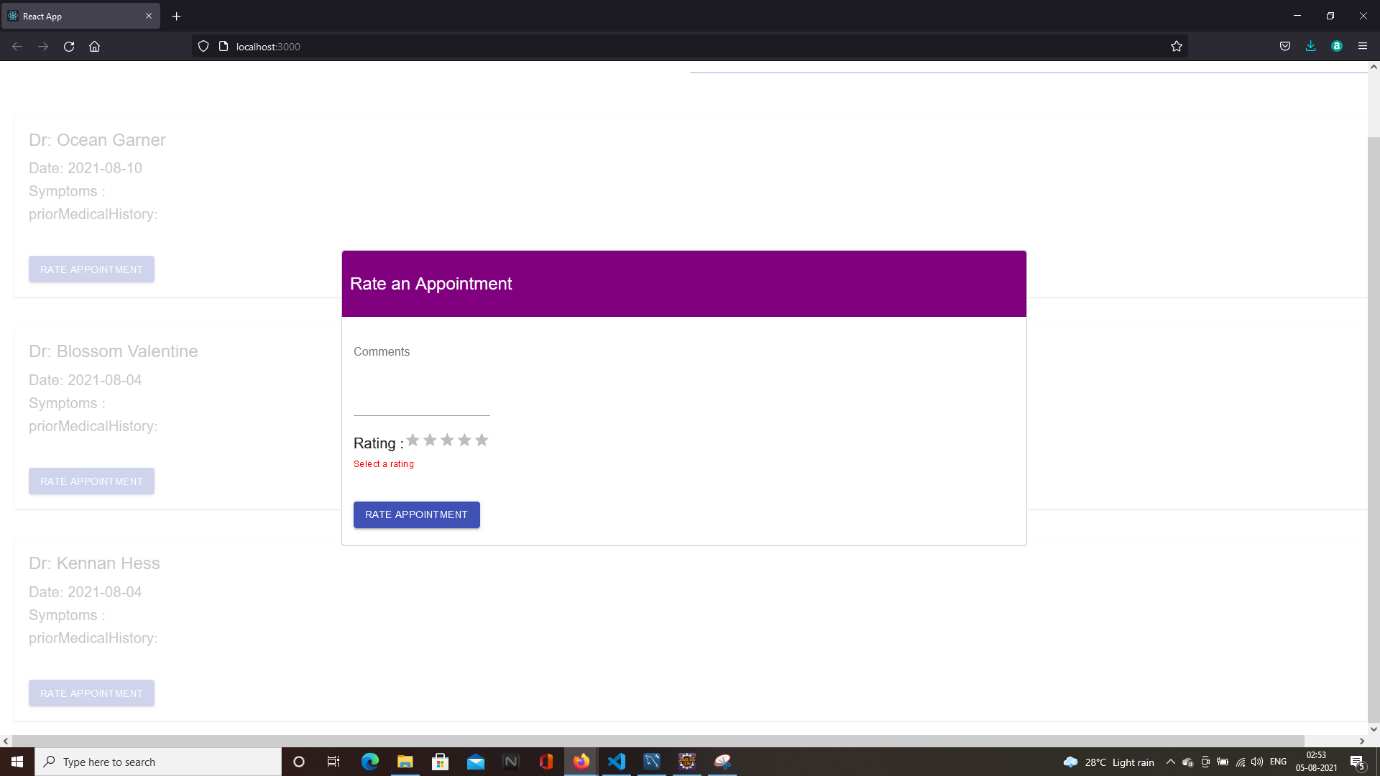
****

**Rate**

**The comments input field shall have a user’s comments on their experience with a doctor. You can create the comment field using the ‘FormControl’ and ‘TextField’ components. The comments field is optional for the user to fill.**

**Users should be able to rate their appointments with doctors using a star rating system. This means users should be able to give the doctors 1–5 stars based on their experience. This can be created using the ‘rating’ component.**

**If a user clicks on the ‘RATE APPOINTMENT’ button without rating a doctor, then the application should display the message ‘Submit a rating’ in red, just like in this image.**

****

**rate**

**The stylesheet can be as described below.**

**The CSS rules for the paper component can be as follows:**

* **Text should be left-aligned.**
* **The margin value should be 15 px.**
* **Padding should be 20 px.**
* **The cursor type should be a pointer.**

**The CSS rules for the card header can be as follows:**

* **The background colour should be purple.**
* **Height should be 70 px.**
* **Padding should be 11 px.**
* **The colour should be white.**

**Moving ahead, in the next segment, you will be provided with the general points to complete the project successfully.**

**Please take a thorough look at them before starting the project.**

**Q&A**

**Submission Guidelines**

**Once you have completed your project, please follow these guidelines to make your submission.**

**You would need to upload these files as part of submitting your project:**

* **frontend.pdf which is a document containing screenshots of the application**
* **Solution code**

**You need to ensure that you are not changing the file name as mentioned above. You have to put all the files in a folder and compress it (a zipped folder).**