Eötvös Loránd University Faculty of Informatics Department of Media and Educational Informatics



Dochouse

*Supervisor:*

## Illés Zoltán

Associate Professor - PhD, habilitation

*Author:*

## Sadi Mamedov

Computer Science BSc

# Contents

1. Introduction 2
2. User Documentation 3
3. Developer Documentation 4

Chapter 1

# Introduction

Over the past decades web development has been reached to advanced improvements, growing its influence as a proper solution and environment to build real world applications. Since the software development period, applications used to require specific software and hardware requirements only to serve features under the local machine environment. After the introduction of web services, several desktop applications have been migrated to the web environment for the reasons that it provides platform independence, device accessibility, and demanding few system requirements.

## 

## 1.1 Background

Nowadays, information technology plays a significant role in human daily life to such an extent that enables us to manage time efficiently. Moreover, using digital applications we can solve issues related to public services. Traditional Health care services are considered one of the most time consuming, crowdy and nerve-racking for both doctors and patients due to long waiting queues, registering patients, scheduling workflow, unavailability of staff members, etc. On the other hand, It’s very difficult to find professional doctors based on specific medical fields. Consulting with a doctor is one of the common activities in society, whereas due to aforementioned problems we consider it tedious practice that most of the time leads to ignoring our health issues.

## 1.2 Solution to the problem

Since web platforms are accessible from various types of devices including laptops, tablets, mobile phones, considering the large audience and necessity of this problem, I decided to build the application over the web surface. By accessing the homepage of this application, patients may benefit from making a medical appointment with qualified doctors, also from receiving answers to their health-related questions through written communication. At the same time, doctors may register to the trusted platform and continue to provide their services to the patients in need. The application intended to serve two sorts of user audience: patients and doctors, who are required to register in order to use services, whereas, guests will be able to collect general details and information about the provided facilities.

Chapter 2

# User Documentation

There will be two different portals with distinct interfaces corresponding to the user type.

Through the documentation, patient type users will be referred as ‘users’ , while doctor type users will be referred as ‘doctors’. Usage of the program consists of following functionalities:

1. Common interface:

All kinds of users including unauthorized users referred to as ‘guests’ can register, navigate to the login transition page and homepage of the application. After the registration, the type of user is determined.

1. User interface:

Users may sign in, sign out, search doctor, edit user profile, navigate to the doctor’s profile, make an appointment request, delete settled requests, review doctor, make payment, and initiate communication via chat.

1. Doctor interface:

Doctors may sign in, sign out, edit doctor profile, accept a request, reject a request, delete settled requests, join communication via chat.

Some system requirements must be satisfied in order to use the application.

## 2.1 System requirements

Devices must have network access having bandwidth greater than 50Kb for page size. Also, from hardware minimum 2 GB of RAM, and 64-bit environment 2 GHz dual processor on MS-Windows (7 and later), MacOS (versions 10.2 and later) , latest Linux (64 bit) distributions, for displaying standard resolution of (1024 x 768) VGA are required.

### 2.1.1 Software requirements

All modern browsers support react-built web applications. For better performance latest releases of Chrome, Mozilla or Safari recommended. In order to run the client and server side application Node package manager (npm) must be installed in the operating system.

## 2.2 Installation process

In order to run the application in local environment follow instructions below:

1. Copy the following URL of github public repository of the project.

<https://github.com/MoneiBall/DocHouse.git> [1]

1. On your local machine, clone the repository to the working directory using

‘git clone <https-URL>’ command.

1. Navigate to the cloned repository, and install all dependencies for the server application using the ‘npm install’ command.
2. Navigate to the client repository using ‘cd doc-house’. Install all dependencies for the client application using ‘npm install’ command.
3. To test both client and server applications run ‘npm test’ command, otherwise skip this step.
4. Navigate back to the root repository, and run ‘npm start’ which will run both client and server application concurrently.
5. If there have not occurred prior errors until this step, npm successfully will redirect you to ‘Dochouse’ homepage on your default browser.

To review live demo version of application follow the URL below:

<https://heroku.com/MoneiBall/DocHouse> [2]

## 2.3 Common Interface Guidelines

In this part of the chapter, a detailed explanation of the common user interface will be presented using figures. The common interface can be accessed by all kinds of users after triggering the client side application.

### 2.3.1 Home page

After starting the program, the homepage of the website will appear in your default browser.

Users may obtain general information about the application and its services. One can review

all the details using component scroller. With the help of navigation bar , users may proceed to login transition page (top right corner), and by clicking Menu button (top left corner) to open Sidebar which provides navigation to the other pages (**Figure** **2.3.1**).

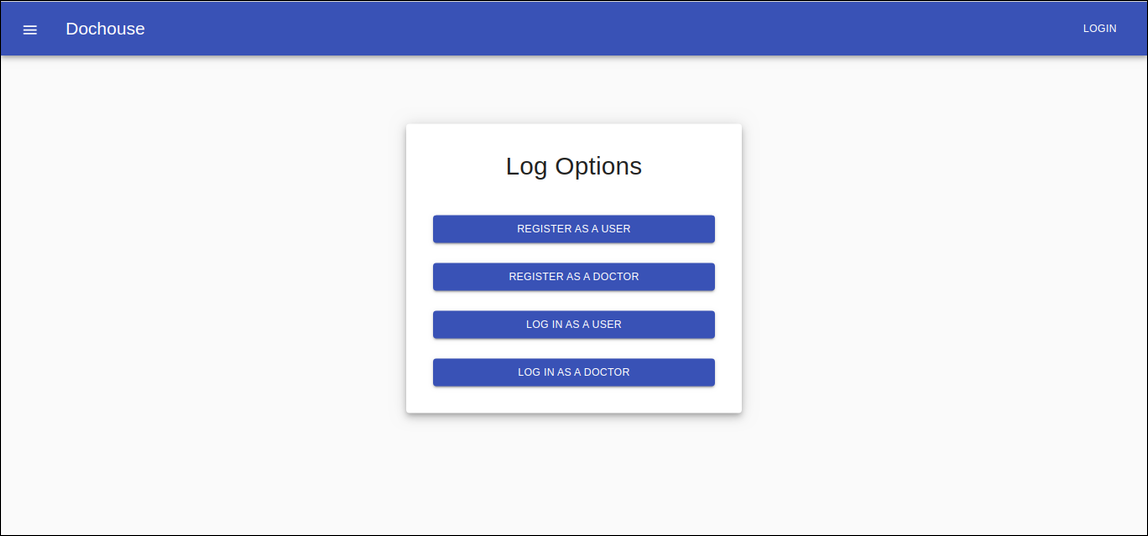


**Figure 2.3.1**: Home page of the website

Pages, other than those which are not allowed to access from unauthorized users are explorable from the home page.

### 2.3.2 Login Transition

Transition page provides account options for users and doctors. By clicking on the named buttons, one will be redirected to the dedicated interface in order to register and sign in to the application **(Figure 2.3.2)**.



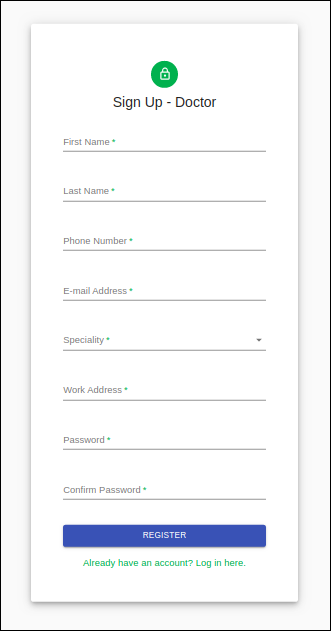
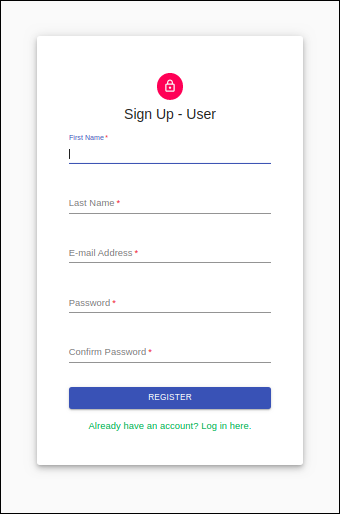
**Figure 2.3.2**: Login transition page of the website

### 

### 2.3.3 User and Doctor Registration

After pressing on *Register as a user* buttondepicted on **(Figure 2.3.31),** users lead to the registration page. Users must fill out required fields and then submit the registration form. To continue further , users will be tempted to enter their data until all the validation rules are satisfied. In case valid data inserted, within the form snackbar appears illustrating successful registration. As the next step, By following the snackbar link, users may sign in to the application.

Similar to users, doctors are also required to register by filling out their personal data. Validation rules will apply until correct standard input is provided, and then a successful registration indicator will pop up. The provided data by user and doctor, will be used in the authentication process to identify log entities **(Figure 2.3.32)**.

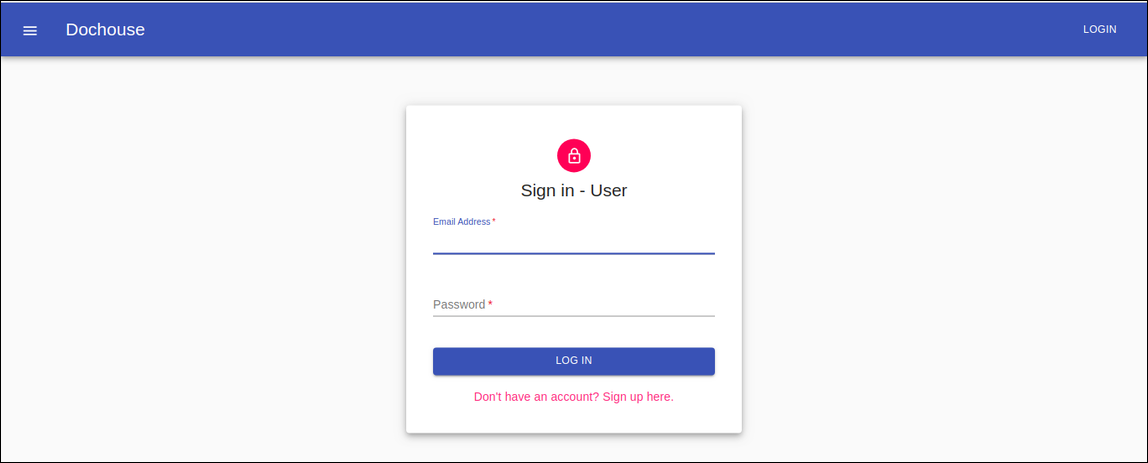


**Figure 2.3.31**: User sign up component **Figure 2.3.32**: Doctor sign up component

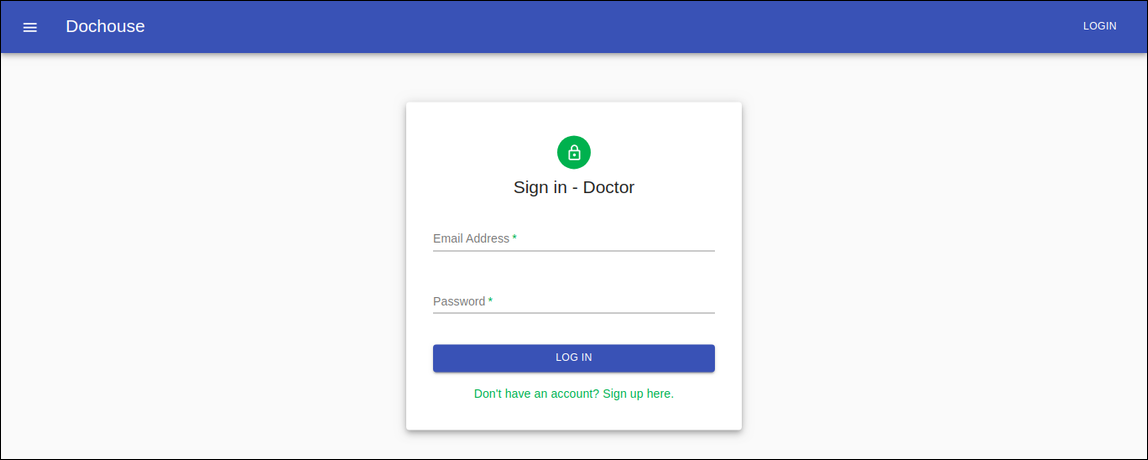
### 2.3.4 User and Doctor Login

By following any of the routes to login pages mentioned in previous parts of chapter, users and doctors can access the corresponding login pages. To continue with the sign in process, the program requires account credentials those of which are acquired in the registration process. Thus, only authorized users and doctors may have account sessions. After entering credentials and pressing *Log In* button, program will redirect to profile page **(Figure 2.3.41,**

**Figure 2.3.42).**



**Figure 2.3.41**: User sign in page



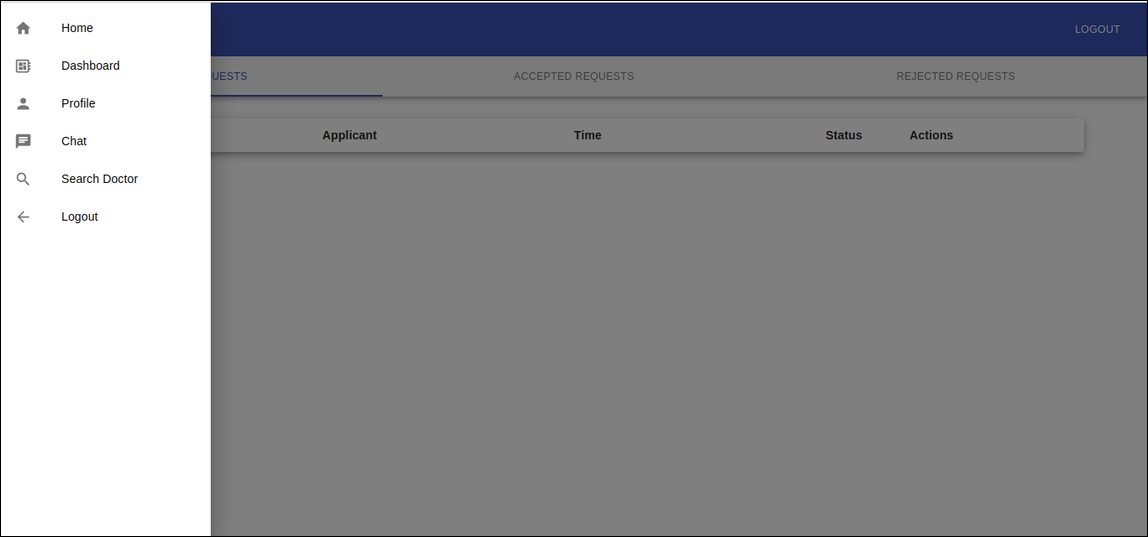
**Figure 2.3.42**: Doctor sign in page

## 2.4 User Interface Guidelines

In this chapter, user interface components will be introduced using sample figures.

### 2.4.1 User Navigation Sidebar

User dashboard is the first page after the user successfully logged in. Navigation bar will appear on every page covering the *Logout* button which is responsible to log the user out and terminate the session, and the *Menu* button which provides the sidebar options in the left side of screen **(Figure 2.4.1**). Sidebar gives important navigation facilities to lead to relevant pages, and provides better user experience on small screen devices. More detailed explanation about the dashboard will be presented in the following subchapters.

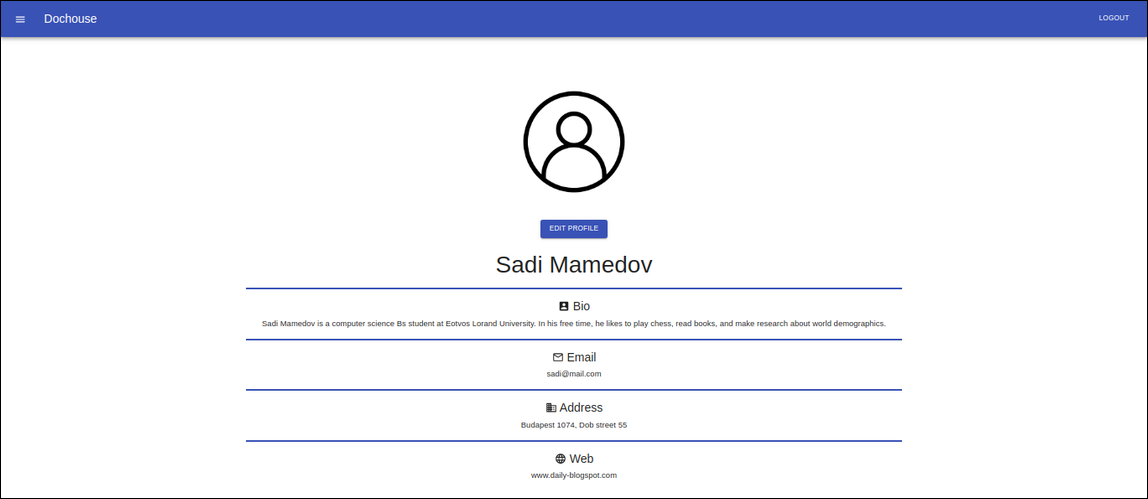


**Figure 2.4.1**: User dashboard with open sidebar

### 

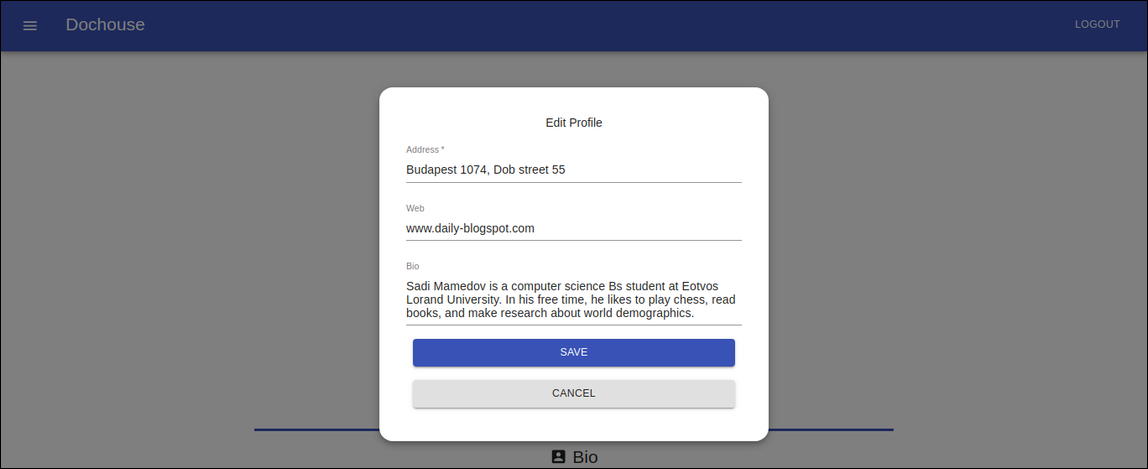
### 2.4.2 User Profile

Following sidebar navigation users may reach to the profile page. Profile page resembles a portfolio where users may review their personal information **(Figure 2.4.21)**.



**Figure 2.4.21**: Profile page of website

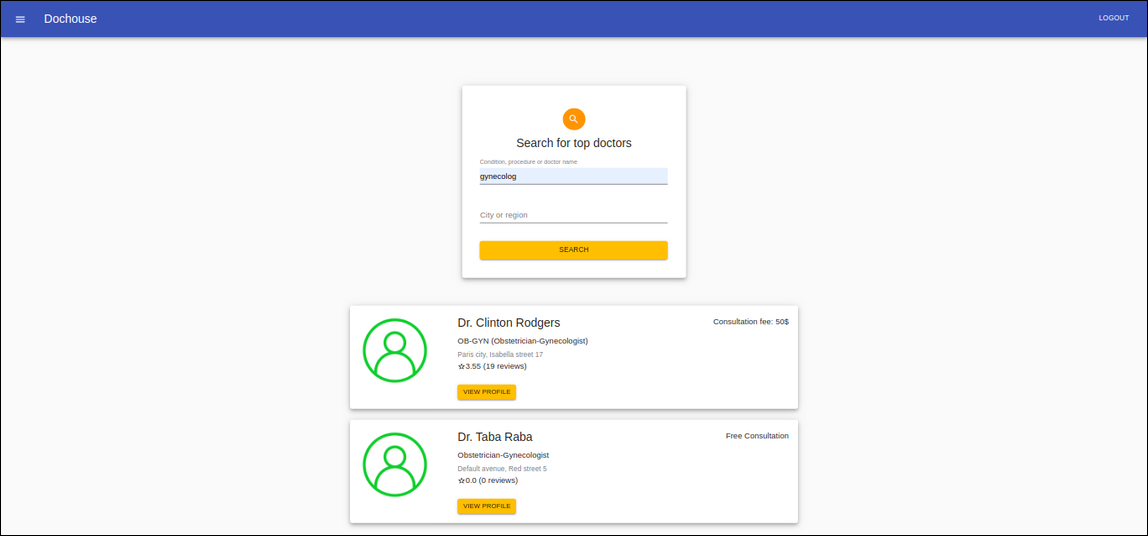
Users are given functionality to edit profile details by pressing *Edit Profile* button. After making changes to corresponding fields, one may save all the changes by clicking *Save* button. *Cancel* button will simply close the modal and ignore changes (**Figure 2.4.22**).



**Figure 2.4.22**: Edit Profile Modal of profile page

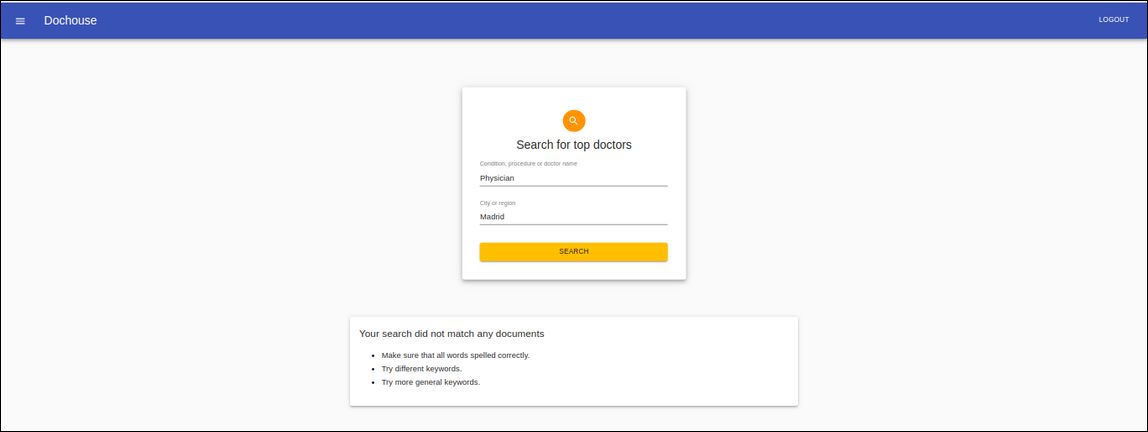
### 2.4.3 Search Doctor

Users may discover doctors who are members of the application network. Doctors may be searched based on *city or region*, and *condition or doctor name* criterias. After clicking on *search* button, a list of doctors will appear representing doctor description including their rating, work address and consultation fee. *View Profile* buttonwill forward the user to the selected doctor’s profile **(Figure 2.4.31)**.



**Figure 2.4.31**: Search Doctor page of website

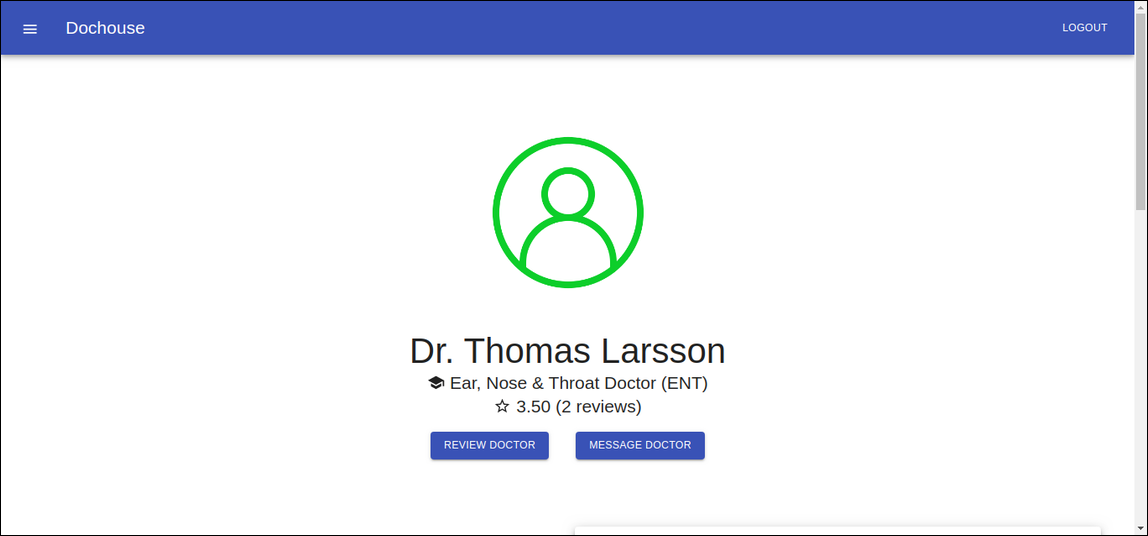
In case of no information found based on search query, relevant warning will pop up under the search component **(Figure 2.4.32)**.



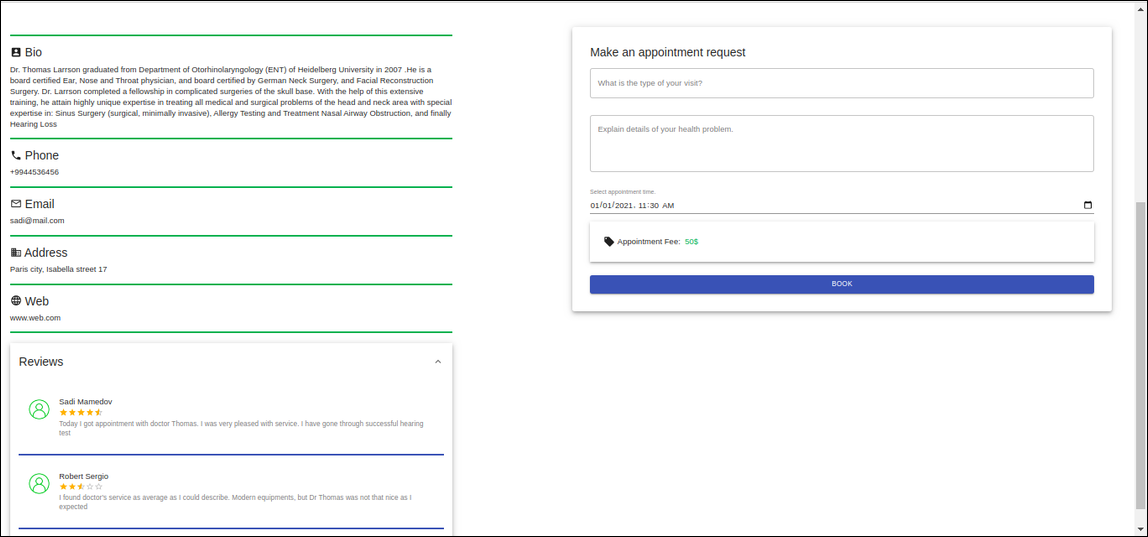
**Figure 2.4.32**: Not found warning of Search Doctor page

### 2.4.4 View Profile

One of the most necessary part of the application is to view the doctor's profile where users may decide whether to make an appointment request based on the doctor's qualifications. Aside from personal details, review scores conducted by other patients are illustrated. Considering the appointment fee users may submit a request by following *Book* button. By clicking *Review Doctor* button, the user may initiate interactive chat with the doctor. Subject of the appointment, detailed explanation of the health issue and appointment proposal time must be provided by the user **(Figure 2.4.41, Figure 2.4.42)**.

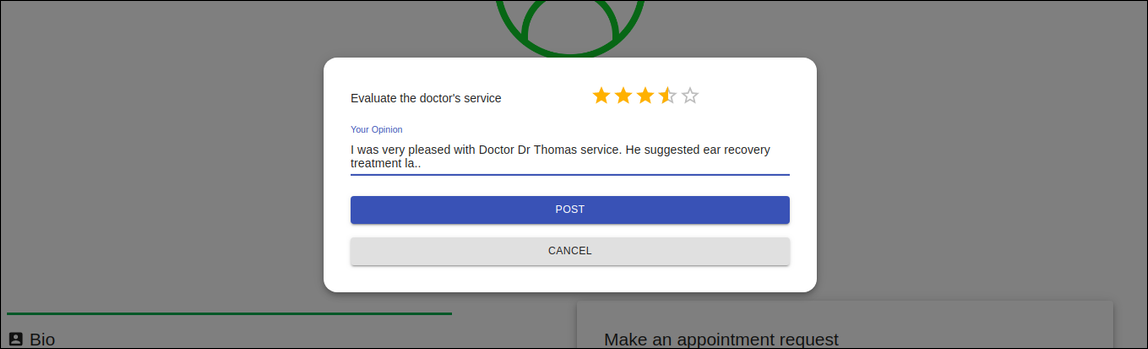


**Figure 2.4.41**: Upper image of View profile page



**Figure 2.4.42**: Lower image of View profile page

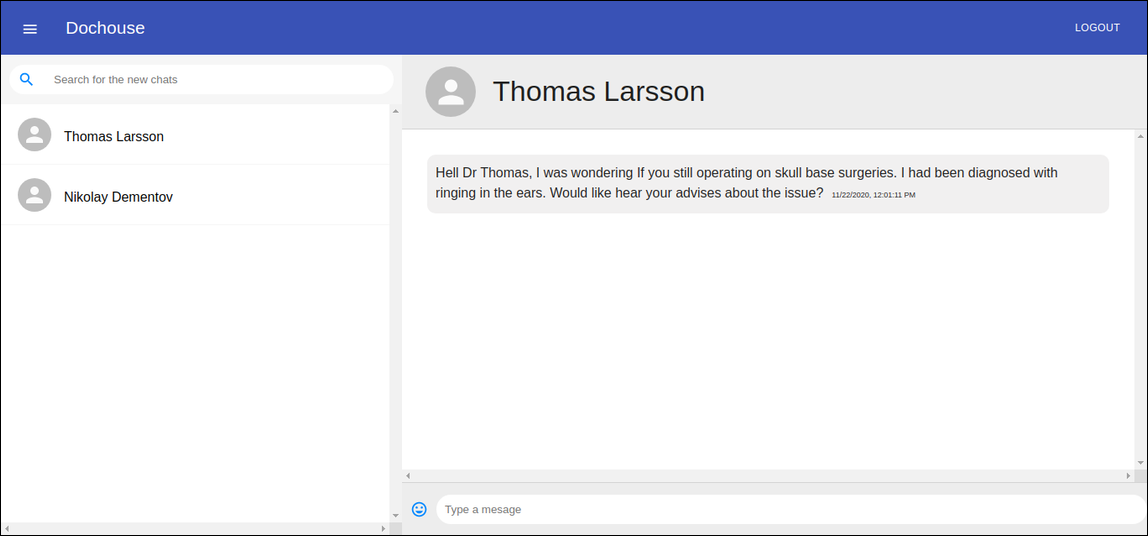
Users may share feedback regarding the doctor’s service by pressing on *Review Doctor* button. The *Post* button will publish user opinion and an evaluation score represented by stars, while the *Cancel* button will simply ignore the operation and close the review modal **(Figure 2.4.43)**.



**Figure 2.4.43**: Review modal of View profile page

### 2.4.5 User Chat

Users may either initiate interactive chat from doctors profile, or reach directly from sidebar navigation. The chat is implemented on a real time basis. The left side of the window shows a list of conversations, while the right depicts actual chat messages (**Figure 2.4.5**).

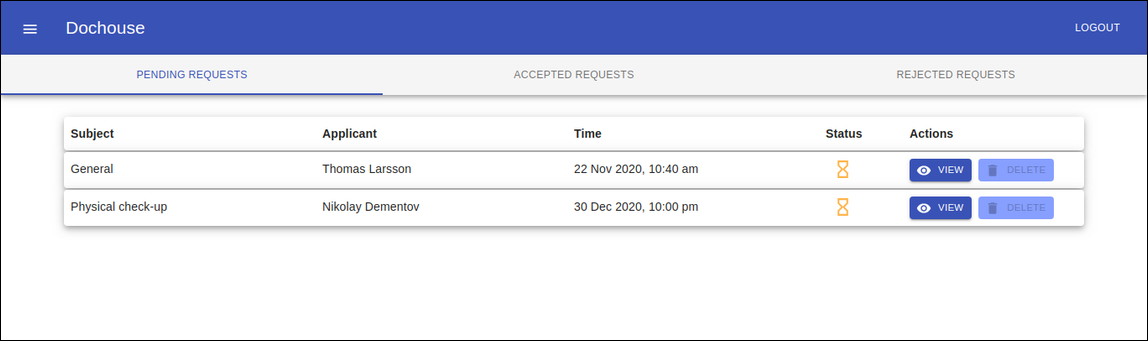


**Figure 2.4.5**: User mirror of the chat window

### 2.4.6 User Dashboard

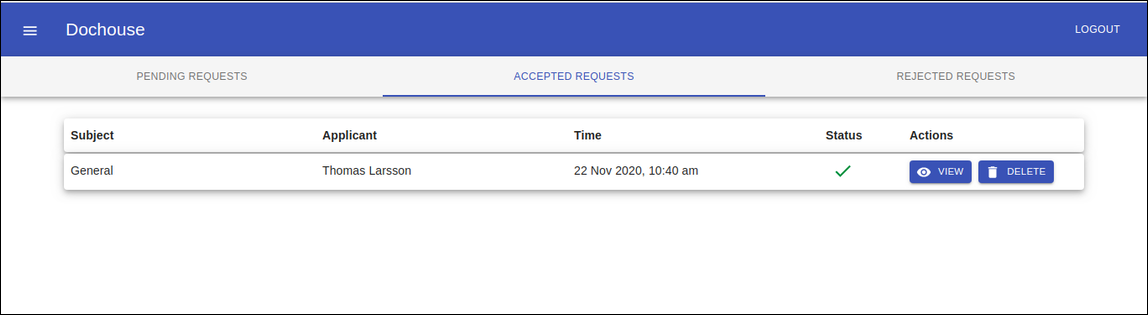
Using sidebar navigation, users may access the dashboard which is the first page that appears after the login process. Users may control all request activities by dashboard tools. The dashboard consists of three categories: *pending requests, accepted requests,* and *rejected requests.* The table contains *subject, applicant, time, status* fields regarding request, as well as, *view* and *delete* action tools, which of those are available to users depending on the type of subject.

Pending requests represent the appointment requests which have not been resolved by doctors **(Figure 2.4.61)**.



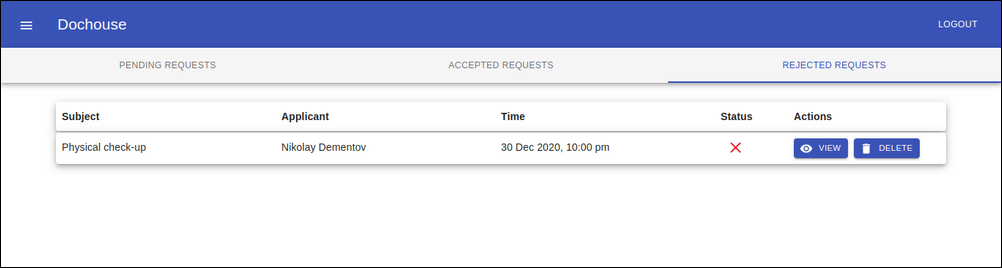
**Figure 2.4.61**: Pending requests tab of User dashboard

Accepted requests represent the appointment requests which have been accepted by doctors **(Figure 2.4.62)**.



**Figure 2.4.62**: Accepted requests tab of User dashboard

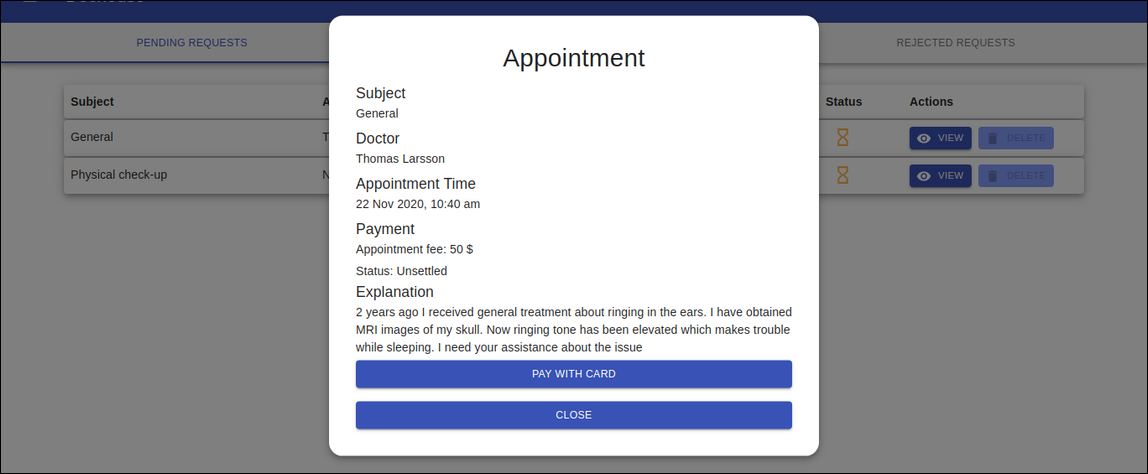
Rejected requests represent the appointment requests which have been refused by doctors **(Figure 2.4.63)**.



**Figure 2.4.63**: Rejected requests tab of User dashboard

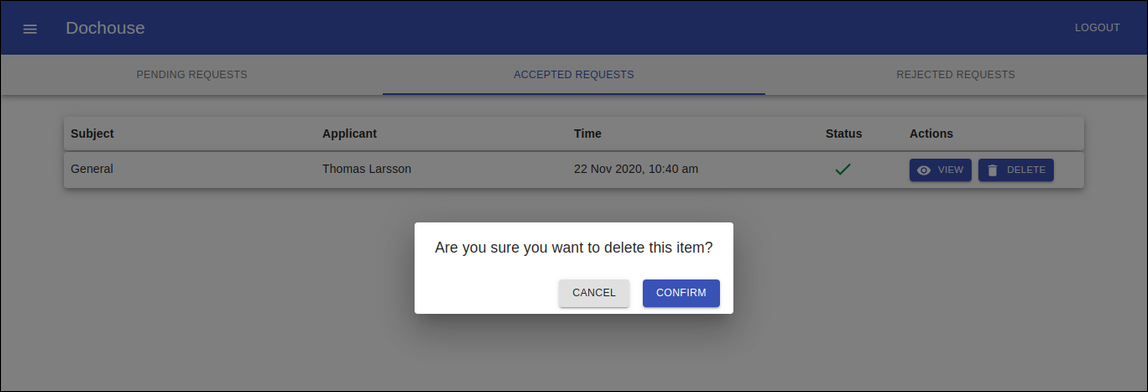
By applying *view* button users may see a detailed description of the appointment request. If the request has a predetermined appointment fee, users may proceed to the payment gate by following *pay with card* button. Use *close* button in order to close the *appointment* window

**(Figure 2.4.64)**.



**Figure 2.4.64**: Appointment window of pending request

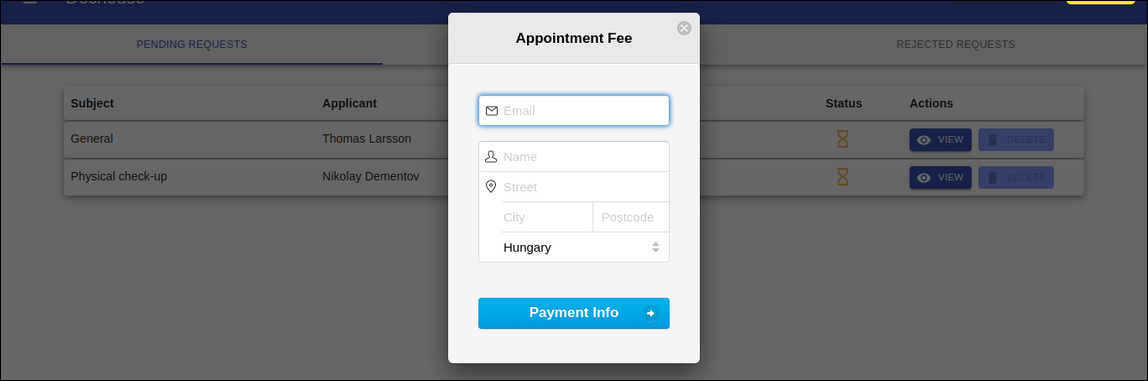
Except from the pending requests, users are allowed to delete accepted and rejected requests. After the dialog appears, suggested buttons of the dialogue will operate based on their names **(Figure 2.4.65)**.



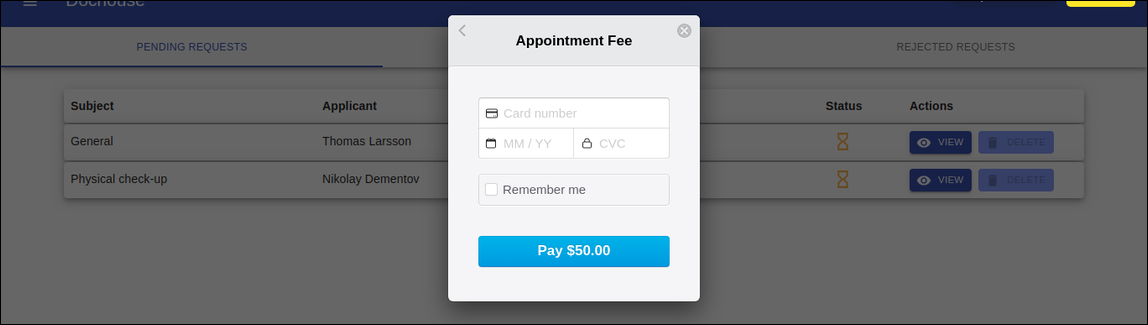
**Figure 2.4.65**: Delete dialogue of accepted request

### 2.4.7 Payment

Payment gate is responsible to make card payments according to the corresponding appointment fee. After inserting required personal datas on the first step **(Figure 2.4.71)**, the user may proceed to the last step where card details are required to make payment **(Figure 2.4.72)**. After the operation is executed, users will be notified about the payment status. On the other hand, doctors will be able to see the payment status of the appointment request.



**Figure 2.4.71**: First step of payment gate



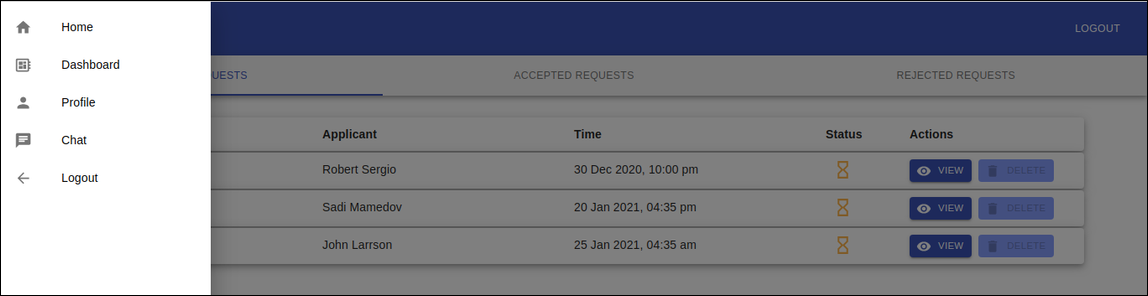
**Figure 2.4.72**: Last step of payment gate

## 2.5 Doctor Interface Guidelines

In this chapter, user interface components will be introduced using sample figures.

### 2.5.1 Doctor Navigation Sidebar

After successful sign in, doctors have access to the account canvas. Navigation sidebar is accessible on all pages from the top left corner, and provides the following options: *Home*, *Dashboard*, *Profile*, *Chat* and *Logout*. Selecting one of the options, doctors may navigate to other pages of canvas **(Figure 2.5.1)**.



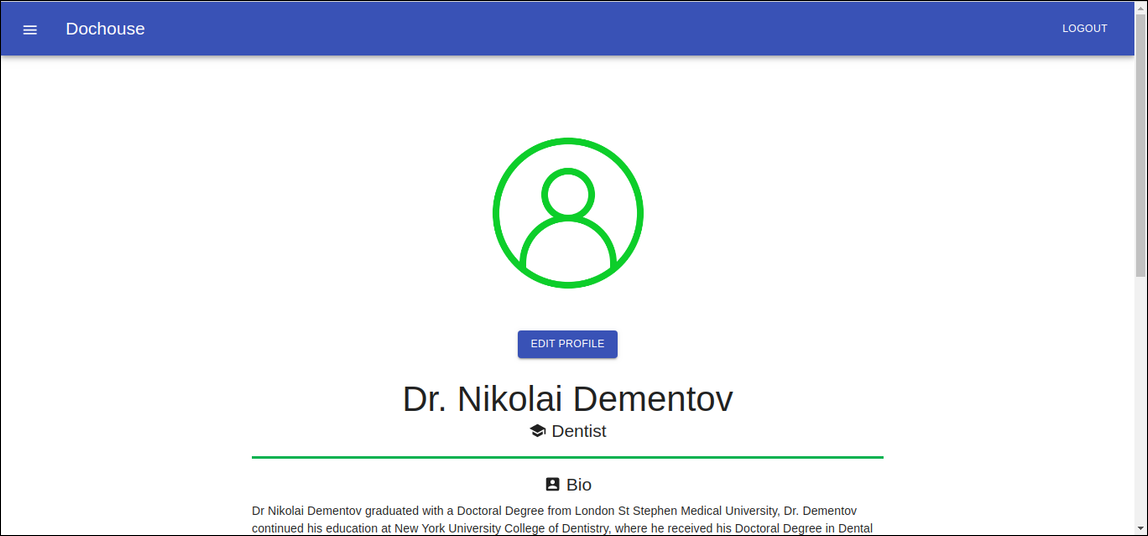
**Figure 2.5.1**: Doctor dashboard with open sidebar

### 

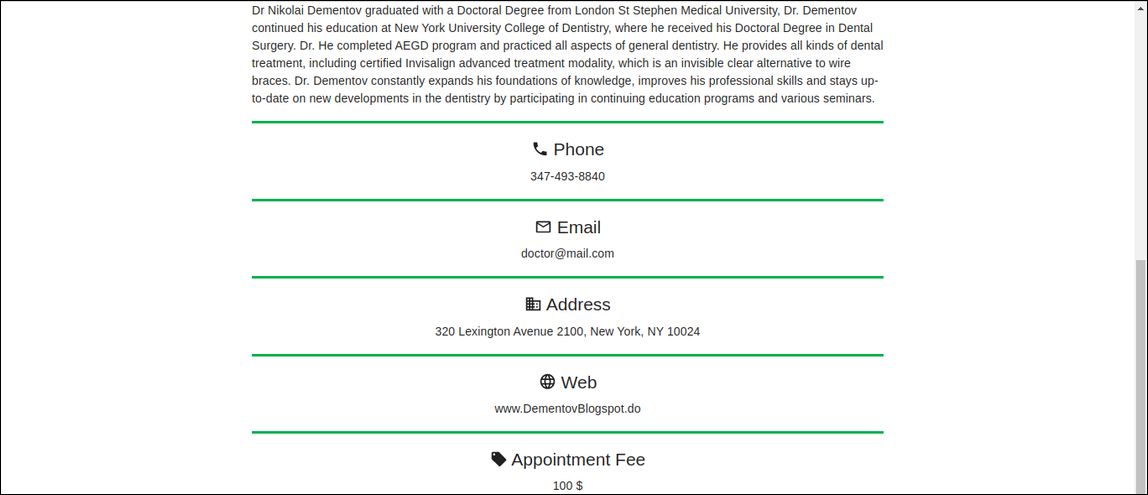
### 2.5.2 Doctor Profile

Doctor profile is one of the crucial interfaces that application provides. In this part of the website, important doctor details are illustrated which will be visible to the patients. Contact details such as *phone, email, speciality,* and *address* are saved after the registration process.

Apart from the concrete data, doctors should prepare substantial content for the *Bio* section such as their graduated institutions, work experiences, and some other field qualifications **(Figure 2.5.21 , Figure 2.5.22)**.

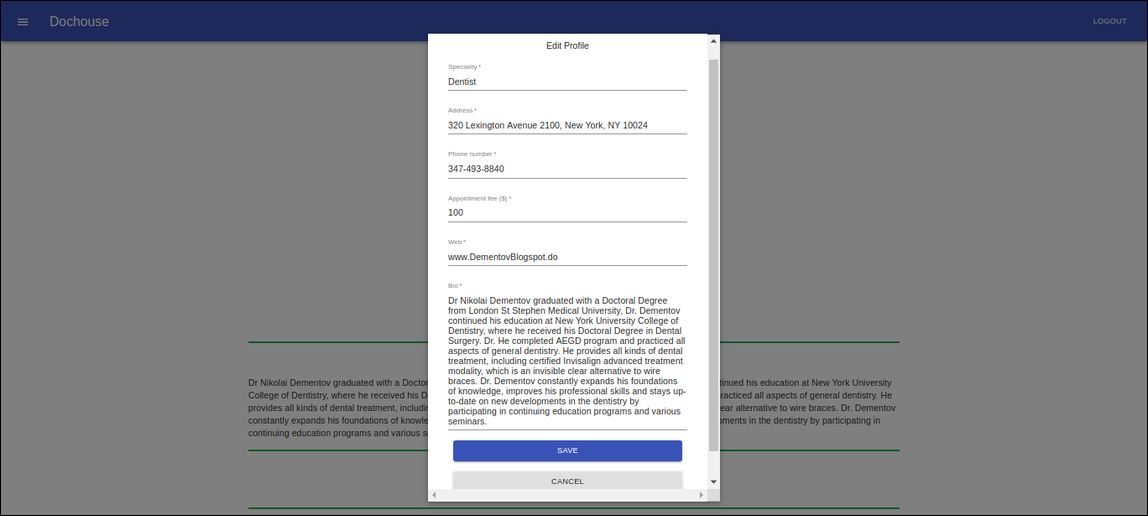


**Figure 2.5.21**: Upper view of the doctor profile



**Figure 2.5.22**: Lower view of the doctor profile

By pressing *Edit Profile* button, doctors can make some changes on displayed profile details. After making changes to proportional fields, clicking *Save* button will save the recent changes, while pressing *Cancel* button will simply close the window and ignore the changes. (**Figure 2.5.23**).



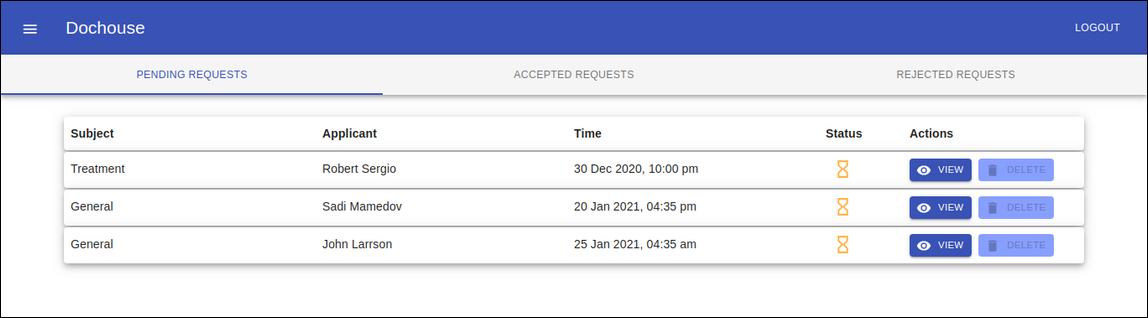
**Figure 2.5.23**: Edit profile window of the doctor profile

### 

### 2.5.3 Doctor Dashboard

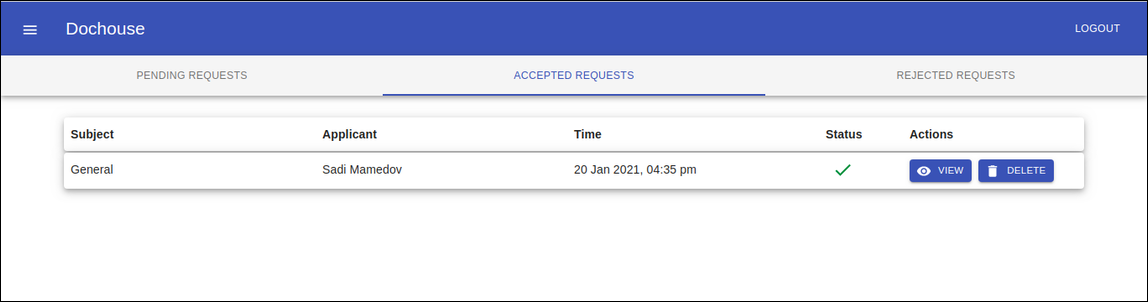
Doctors may manage appointment requests via dashboard tools. The received appointment requests will appear in *pending requests* tab. Depending on the doctor’s decision of the pending request, that will be transferred to *either accepted requests* or *rejected requests*. The table contains *subject, applicant, time, status* properties, and *action* toolset.

Pending requests represent the appointment requests that have not been responded by the doctor **(Figure 2.5.31)**.



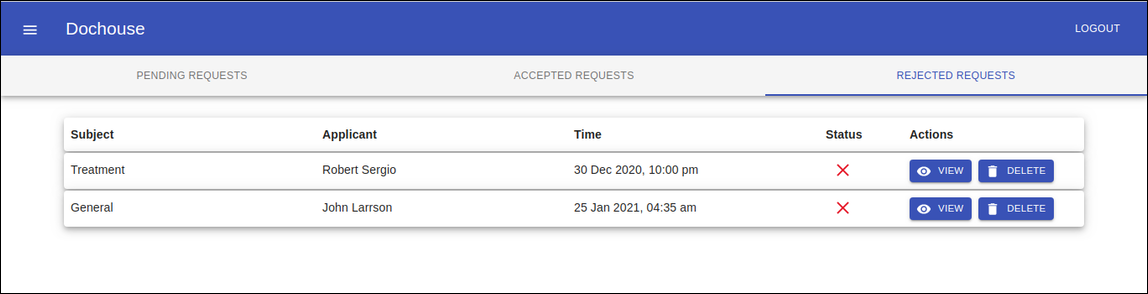
**Figure 2.5.31**: Pending requests tab of Doctor dashboard

Accepted requests represent the appointment requests that have been accepted by the doctor **(Figure 2.5.32)**.



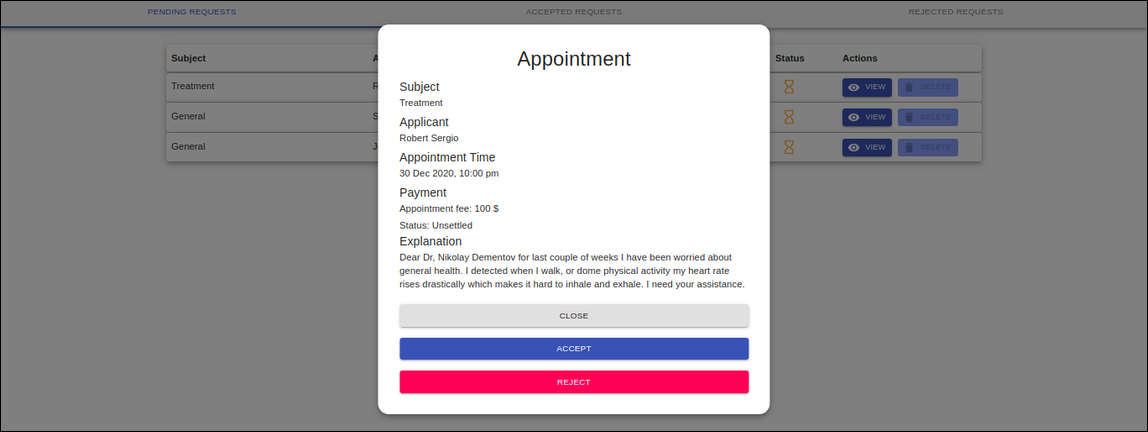
**Figure 2.5.32**: Accepted requests tab of Doctor dashboard

Rejected requests represent the appointment requests that have been rejected by the doctor **(Figure 2.5.33)**.



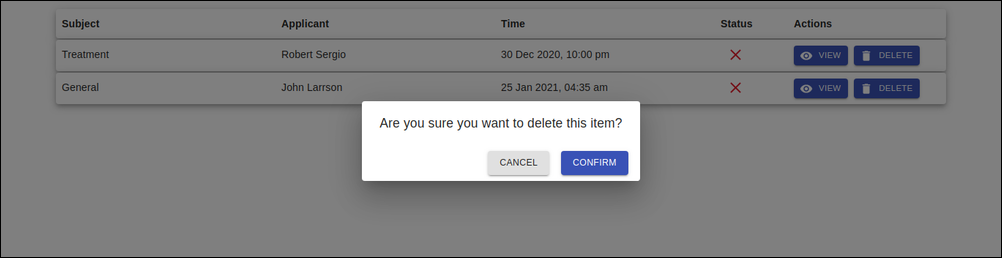
**Figure 2.5.33**: Rejected requests tab of Doctor dashboard

In order to see an appointment request, by clicking on *view* button doctors may check detailed description. Moreover, appointment modal depicts *payment status* and *explanation* of a health issue, Considering all the details, doctors may refuse the request by pressing on the red colored *reject* button, also in contrast, they may accept the request following *accept* button. *Close* button will close the *Appointment* modal **(Figure 2.5.34)**.



**Figure 2.5.34**: Appointment window of pending request from doctor canvas

Doctors have the option to delete accepted and rejected requests. After the dialog window appears, choosing *confirm* option will simply delete the selected request from the doctor dashboard **(Figure 2.5.35)**.



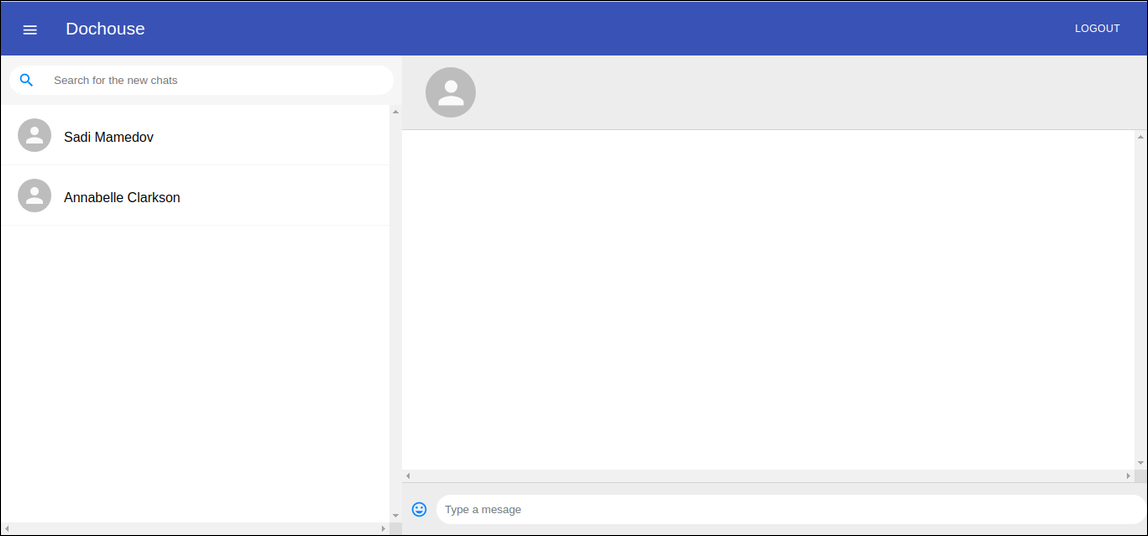
**Figure 2.5.35**: Delete dialogue of rejected request

### 

### 

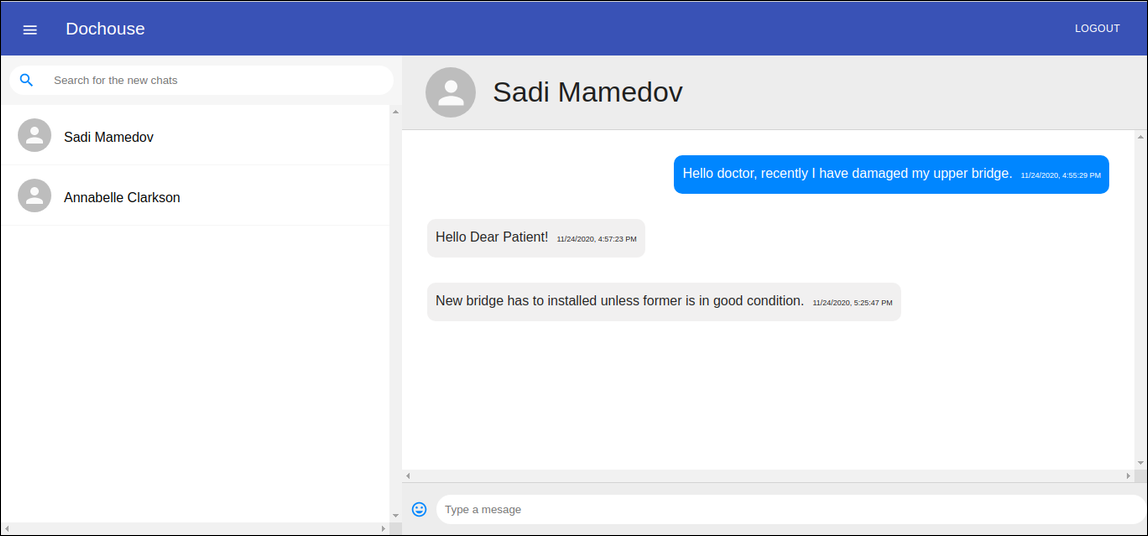
### 2.5.4 Doctor Chat

Doctors may enter interactive chat from sidebar navigation. Left side of the window represents the list of conversations (**Figure 2.5.41**).



**Figure 2.5.41**: Doctor mirror of the initial chat window

Pressing on any of the chat conversations, corresponding messages will appear at the right side of the window. Pressing *Enter* button from keyboard, inserted text message will be sent to the receiver. Received messages are indicated within blue layout (**Figure 2.5.42**).



**Figure 2.5.42**: Doctor mirror of the selected chat window

Chapter 3

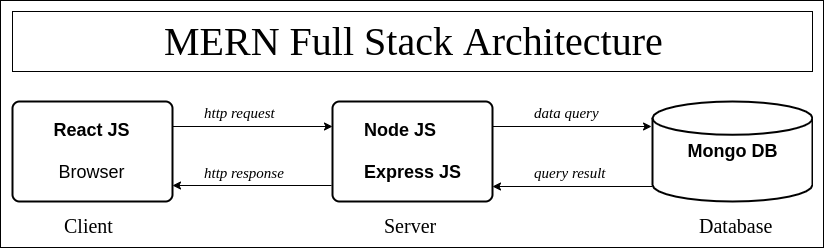
Developer Documentation

General directory structure, class and use-case diagrams will be illustrated in this chapter. Additionally, key functionalities of the application, relationship between client application and server application, database management will be analyzed elaborately, as well as testing plan and results will be discussed. For the configuration section of application, instructions in subchapters **2.1 System requirements** and **2.2 Installation Process** should be applied.

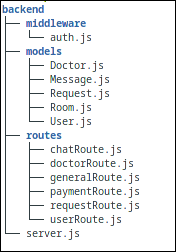
## 3.1 Program Structure

Full Stack application is built according to the MERN Structure. Client side application is developed by React JS framework [3], while server side is developed via Node JS web server [4] and Express JS [5] web framework. And for database management MongoDB database [6] is responsible. General page layout design is built on react library Material-UI [7], and some component designs are inspired from open-source project [8].

React is a declarative, maintainable, component-based JavaScript framework used to build dynamic client-side applications. Using react packages we may connect to the server via sending front-end data through *http requests*. Express JS is a fast, minimalist server-side web framework that runs inside Node JS which is an event-driven, asynchronous JavaScript server runtime environment to build scalable web applications. Using Node JS database drivers or callbacks, documents stored in MongoDB, a cross-platform document-oriented database, may be accessed, updated and deleted. After database operation is completed, a query result in the shape of JSON format is received from a server-side application. Subsequently, received data from the database is forwarded to client side through *http response***.** As a result, client-side UI elements are updated simultaneously **Figure (3.1.1)**.



**Figure 3.1.1**: Full Stack Architecture of the application



Project structure consists of two major subfolders: *backend* **(Figure 3.1.2)** (server) and *doc-house*(client).The main configuration of the server lies in *server.js* file where initially, node server is launched on any determined port, followed by connection established to MongoDB database. Furthermore, the server app listens to changes on the database to make it real-time for interactive chat. Next, a set of backend REST APIs is set up categorizing these into routers based on database models. We will use those routers to receive and handle http requests coming from the front-end.  **Figure 3.1.2**: Tree structure

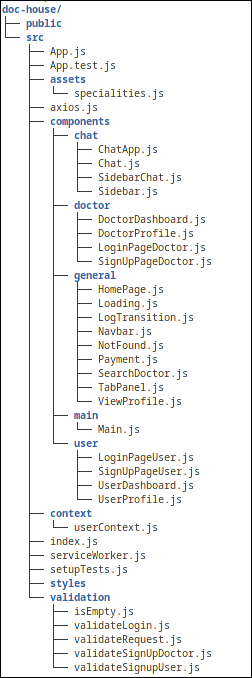
of server application

At the end of the file, the server app continuously listens to the port. Collection of API

routes lies under *routes* directory, while database schema models composed under *models.*

Most of the user requests will require authentication in *auth.js* which will be applied for data security purposes. Middleware will only accept authenticated user actions and block all the requests made from outside of the application.

## 



Client side application files lie under doc-house directory **(Figure 3.1.3).** Initially, React DOM locating at the top of client web application will render *App.js* component in *index.js* file. Later, it returns the Main component in *Main.js* which wraps all the other components with Router to render them based on url path. Also browser history is created at the root, and passed down to have access from children components. Source folder *src* consists of *assets*, *components*, *context*, *styles*, and *validation* subdirectories. Rendering elements locate under *components*, and designs will apply from JavaScript and CSS files located under *styles* subdirectory. In order to store user details Context API along with useContext hook in file *context/userContext.js* will be utilized. Rendering components are stateful class components where rendering is controlled by lifecycle methods, except *chat* components which are stateless, and to control lifecycle methods powerful feature react hooks will be used. Client validation rules under *validation* will apply to form inputs inserted by users and doctors.

**Figure 3.1.3**: Tree structure

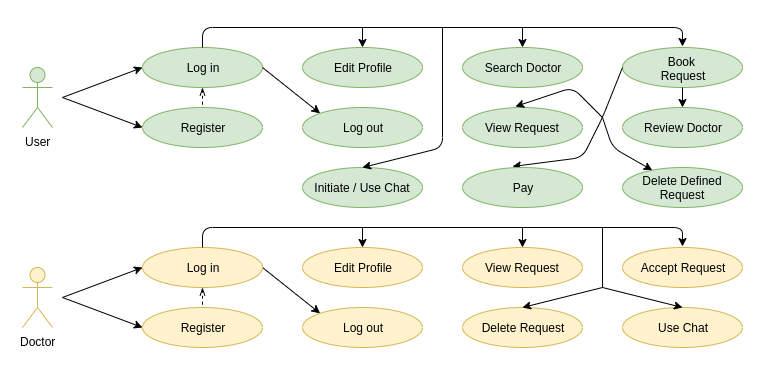
of client application

## 

## 

## 3.2 Use Case diagram

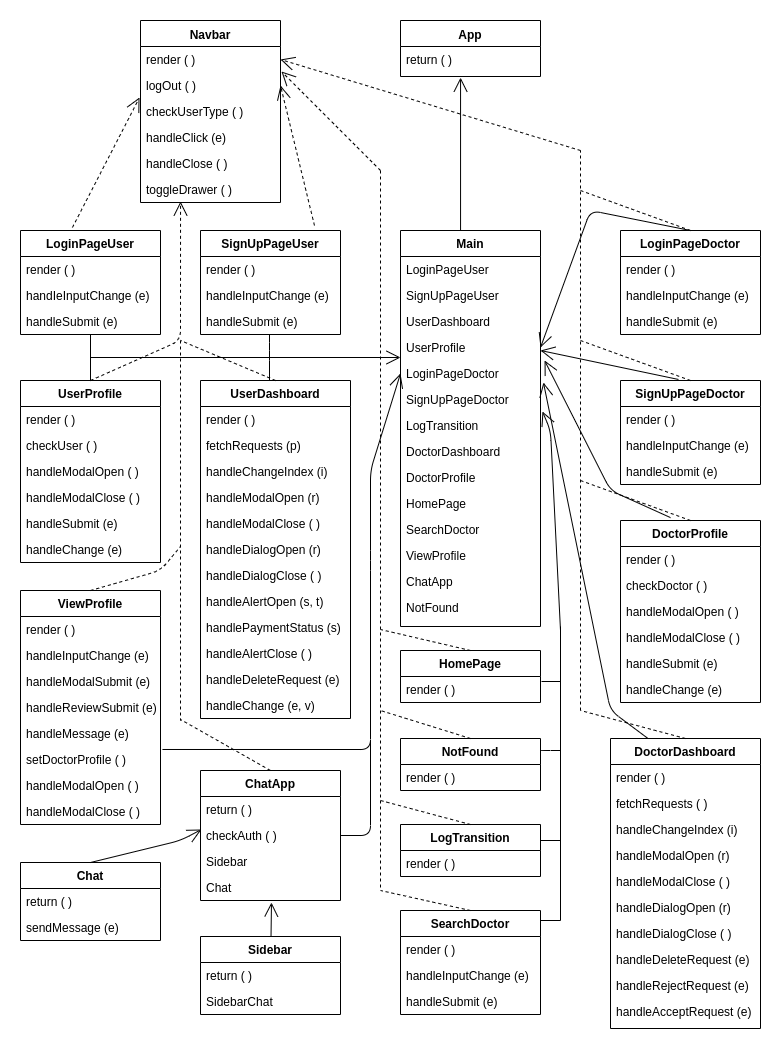
One of the objectives of developing this application is to accomplish high quality user experience and interface design. Straightforward instructions and overall rudimentary design enable users to navigate through and use functionalities at ease. Both users and doctors are required to register in order to log into the account canvas, and to close the session the both may log out. Functions such as editing profile, using chat, viewing and deleting requests are available to both end-users. Users may search the doctor, make an appointment request and initiate the chat conversation. Moreover, the user may opt to pay the appointment fee, and after the service may even review the doctor. On the other hand, doctors react to the request either accepting or deleting it. Use cases of the sequence are connected with directed arrows where the source case is considered as prerequisite, and the pointed one as dependent case (**Figure 3.2)** .



**Figure 3.2.1**: Use case diagram of application

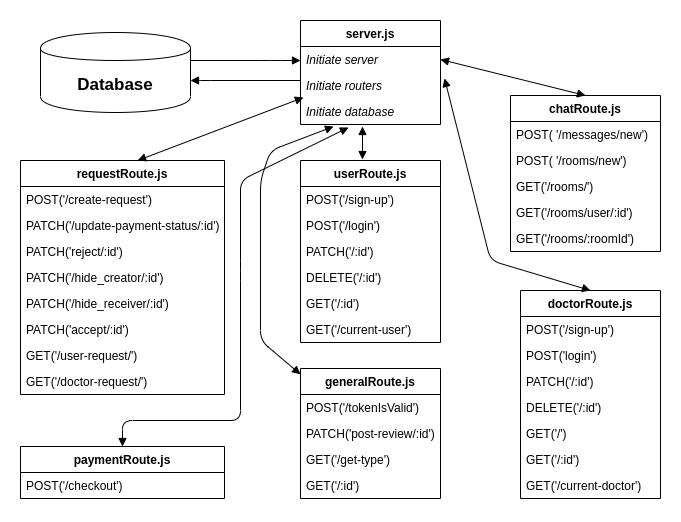
## 3.3 Class diagrams

Apart from database schema, Dochouse web application consists of two significant parts: client-side application and server-side application. Client-side class diagram represents the React classes including member methods and relations with other components which of those all together constitute the client API built into web browser (**Figure 3.3.1)** .



**Figure 3.3.1**: Class diagram of client application

Server-side class diagram (**Figure 3.3.2)** represents mutual use relationship between server and database, and connection between routes and main server thread. Class diagrams depict determined routes where application’s endpoint URLs respond to requests coming from the client-side application. Technically, these routes are http-based RESTful APIs which are mounted on the server app instance. Using Express routers, dedicated URLs can be structured into separate javascript files under different directories. Routes from the figure below contains http request methods like *POST, PATCH, GET, DELETE* and each of them is mounted on following resource locators preserving relative naming conventions : *'/general'*, *'/users', /doctors', '/requests', '/chat'*, and  *'/payment'*.



**Figure 3.3.2**: Class diagram of server application

## 3.4 Authorization Processes

## 3.5 Profile Processes

## 3.6 Chat Process

## 3.7 Request Processes

## 3.8 Payment Process

Chapter 4

Testing

Chapter 5

Conclusion

# Bibliography

[1] Somebody, something.