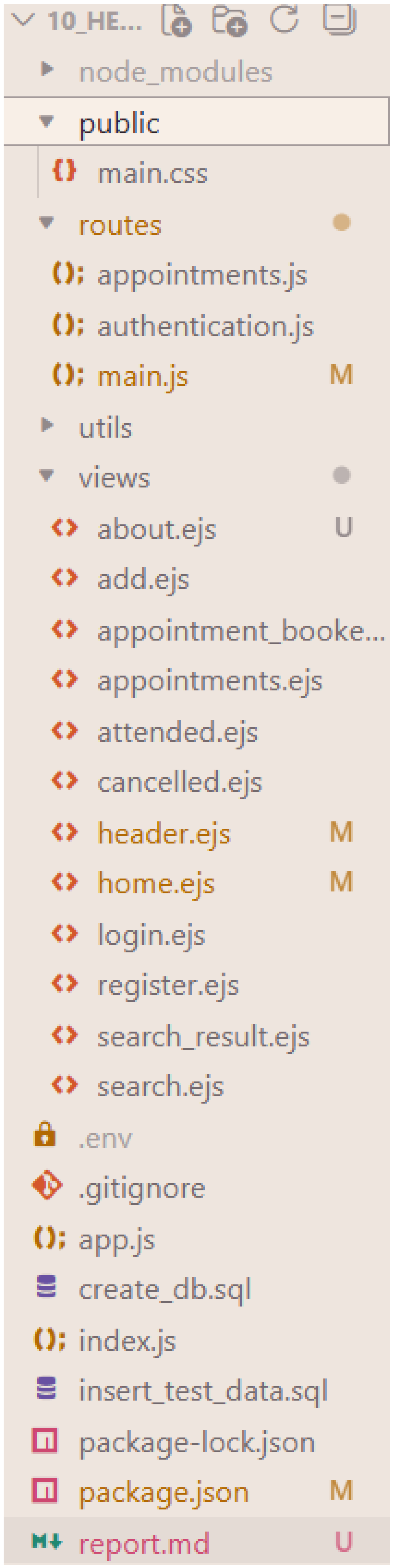
Outline

This application is an appointment system, both doctors and patients are able to log in, create appointments, view their appointments, and cancel appointments. Doctors are also able to mark an appointment as complete.

This application has two levels of authorisation, each user account is classed as either a doctor or a patient and the data they are able to view is different depending on which type of account they have.

All users are able to view all the appointments they are part of and no others, they can also look through a list of either doctors(if a patient) or patients(if a doctor) to create new appointments with a given date and time.

Architecture


```
10_HE...
  node_modules
  public
    main.css
  routes
    appointments.js
    authentication.js
    main.js                M
  utils
  views
    about.ejs              U
    add.ejs
    appointment_booke...
    appointments.ejs
    attended.ejs
    cancelled.ejs
    header.ejs             M
    home.ejs               M
    login.ejs
    register.ejs
    search_result.ejs
    search.ejs
  .env
  .gitignore
  app.js
  create_db.sql
  index.js
  insert_test_data.sql
  package-lock.json
  package.json             M
  report.md                U
```

One aim of this project was to ensure modularity, therefore I structured this project as follows:

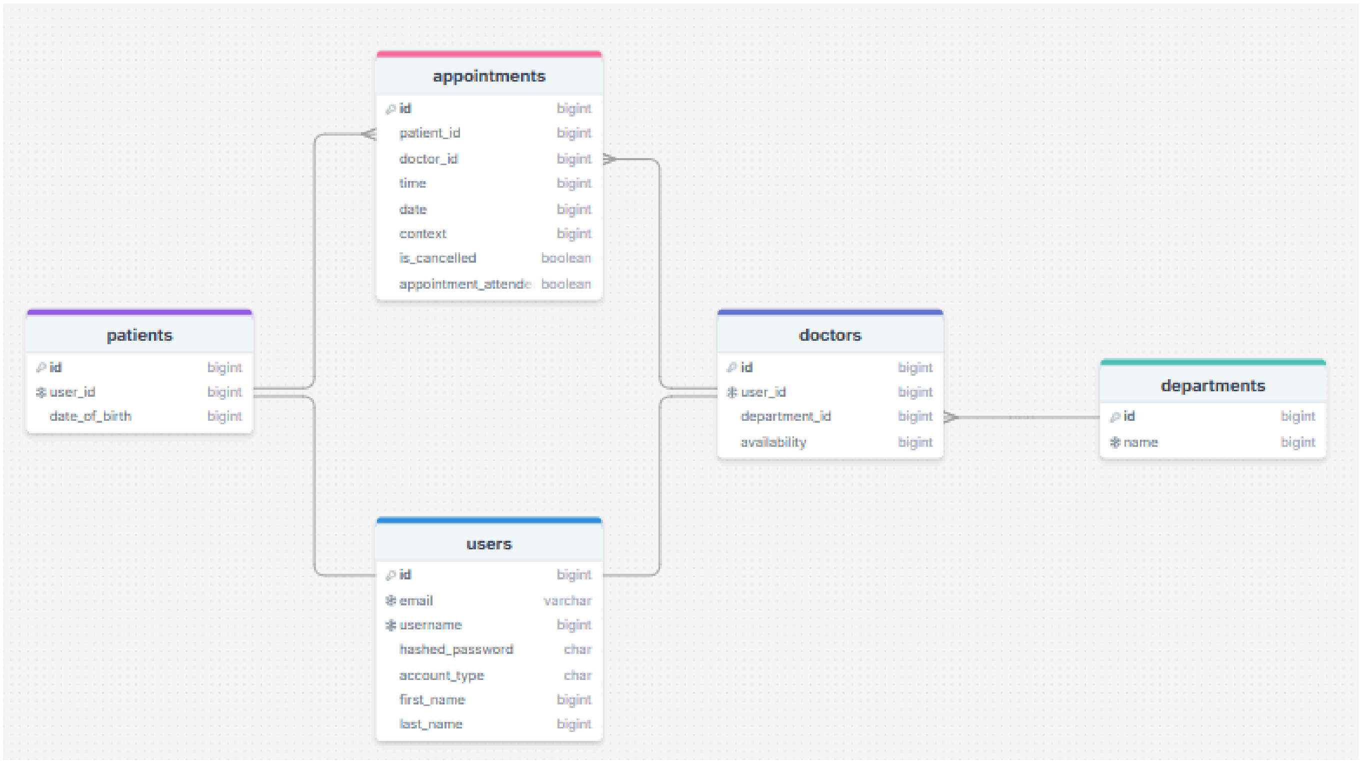
- A public folder, consisting of publicly available files, in this case it was just the CSS
- A routes folder, this folder solely consisted of router objects. To ensure separation of concerns main.js imports the other two routers. This way appointments logic and authentication logic can be kept separate.
- A utils folder, this folder consists of a config file which I used to set environment variables and two helper/utilities files. These files were used to separate application logic from their corresponding router files
- A views folder, consisting of ejs files
- An app.js file, this file contains all logic for initalising node and express
- And index.js, the applications entry point. This file is solely responsible importing from app.js and calling .listen()

This project used:

- Node, for running web server
- Express, for HTTP
- EJS, for serving custom HTML
- bcrypt, for encryption and hashing
- mysql2, for storing information in a database,
- dotenv, for storing secret information
- forever, for ensuring the webapp runs indefinitely
- express-session, used for storing session data

Data Model

The basic database model for this project was as follows



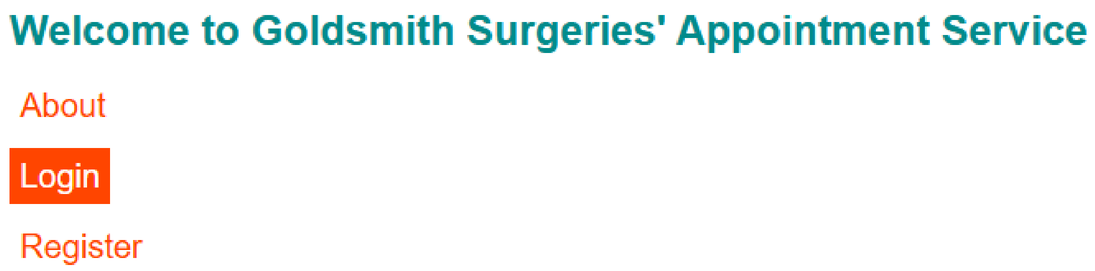
My aim for this model was to reduce data redundancy as much as possible. In order to meet this goal I created one table called users which consisted of all shared data between patients and doctors, I then created two new tables called patients and doctors. These held additional data which the opposing table would not need, such as department.

As one department may have several doctors as members I also created a department table with a one to many relationship with the doctors table, this again allowed me to reduce data redundancy as I was able to create one entry for each department which could then be referenced through the department_id entry on entries in doctors.

Finally I also created a table called appointments, this holds all necessary data for the bookings, such as who the patient is, who the doctor is, and at what time. When interacting with other tables it is done so through mySql joins.

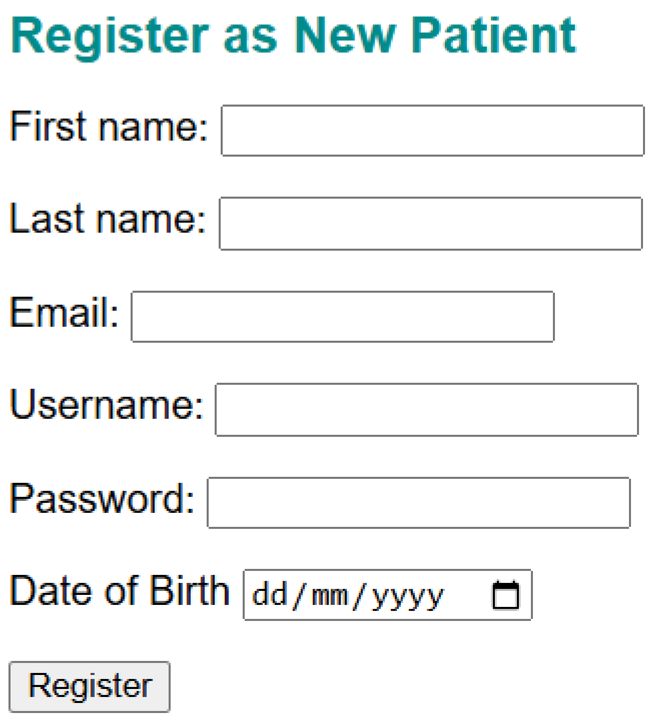
User Functionality

When users first visit the page all they are able to do is log in, register, or view the about page.



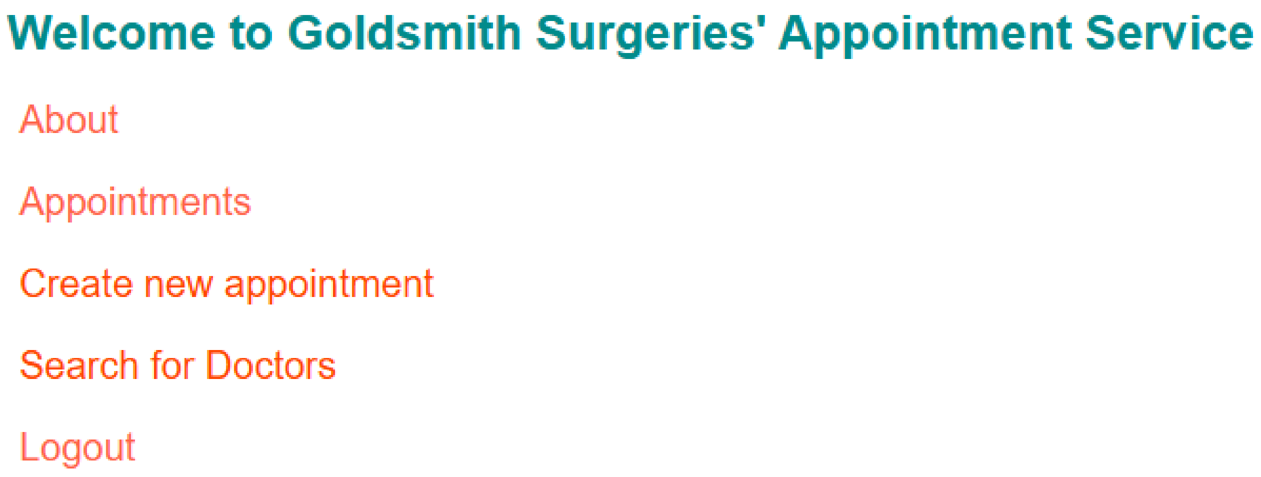
Welcome to Goldsmith Surgeries' Appointment Service

New patients can register as follows.



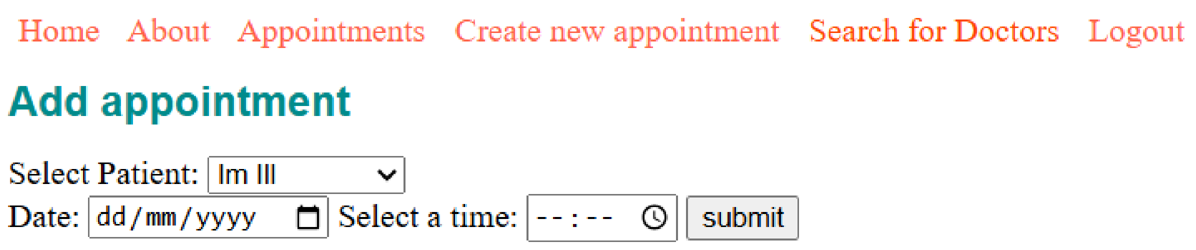
Register as New Patient

Once logged in there are new buttons available



Welcome to Goldsmith Surgeries' Appointment Service

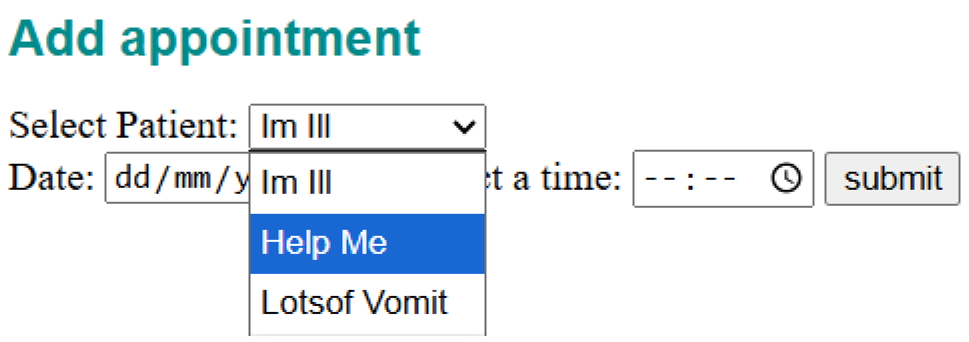
Add appointments takes a user to appointment creation screen,



Add appointment

Select Patient: Im Ill

Date: dd/mm/yyyy Select a time: --:-- submit



Add appointment

Select Patient: Im Ill

Date: dd/mm/y

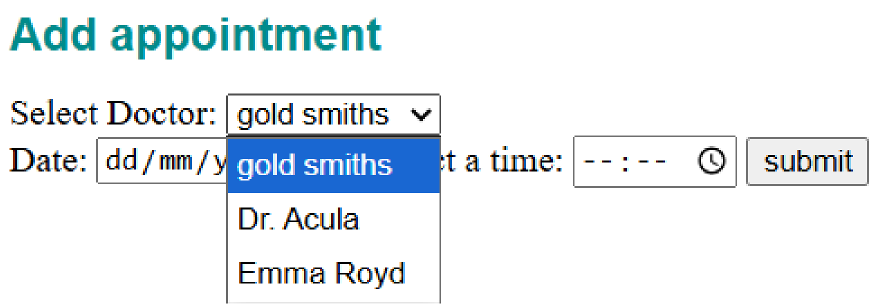
Im Ill

Help Me

Lotsof Vomit

t a time: --:-- submit

Note that the screen you see differs depending on if you are a patient or a doctor, the previous screen was for a doctor creating appointments and can therefore only see a list of patients, in contrast when logged in as a patient a user is only able to see a list of doctors.



Add appointment

Select Doctor: gold smiths

Date: dd/mm/y

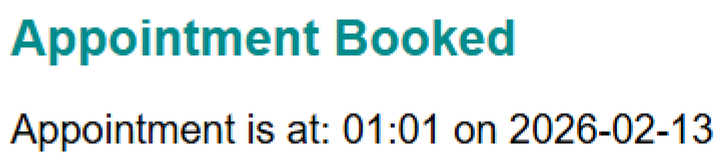
gold smiths

Dr. Acula

Emma Royd

t a time: --:-- submit

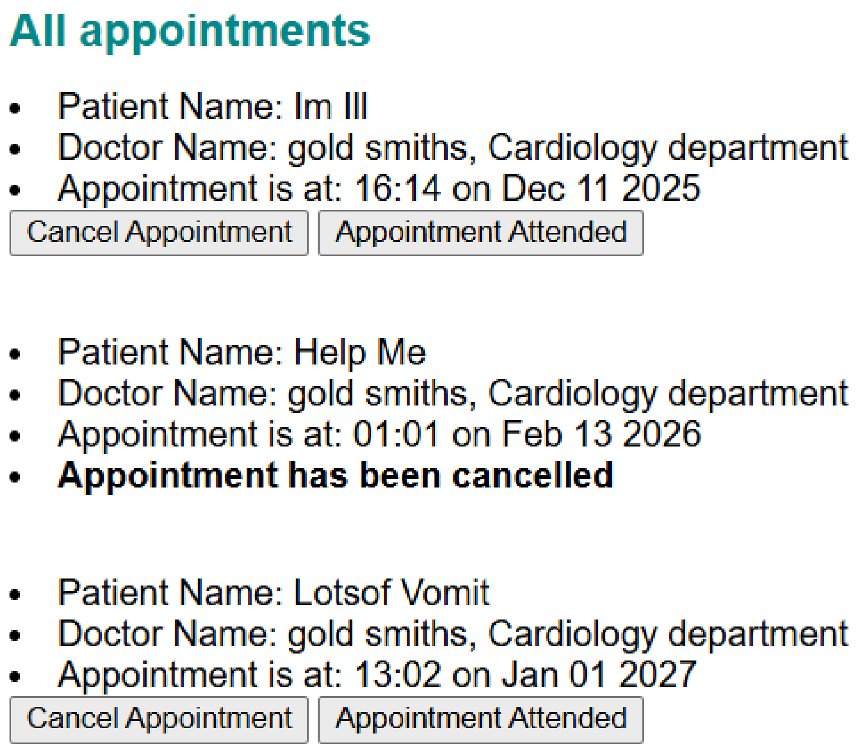
Once an appointment is booked then a confirmation screen appears showing the data and time of the appointment.



Appointment Booked

Appointment is at: 01:01 on 2026-02-13

When viewing all appointments the page is displayed as follows, note that users are only able to see their own appointments. In this case the doctor is the `gold smiths` account and as such can see all the appointments users have with them.

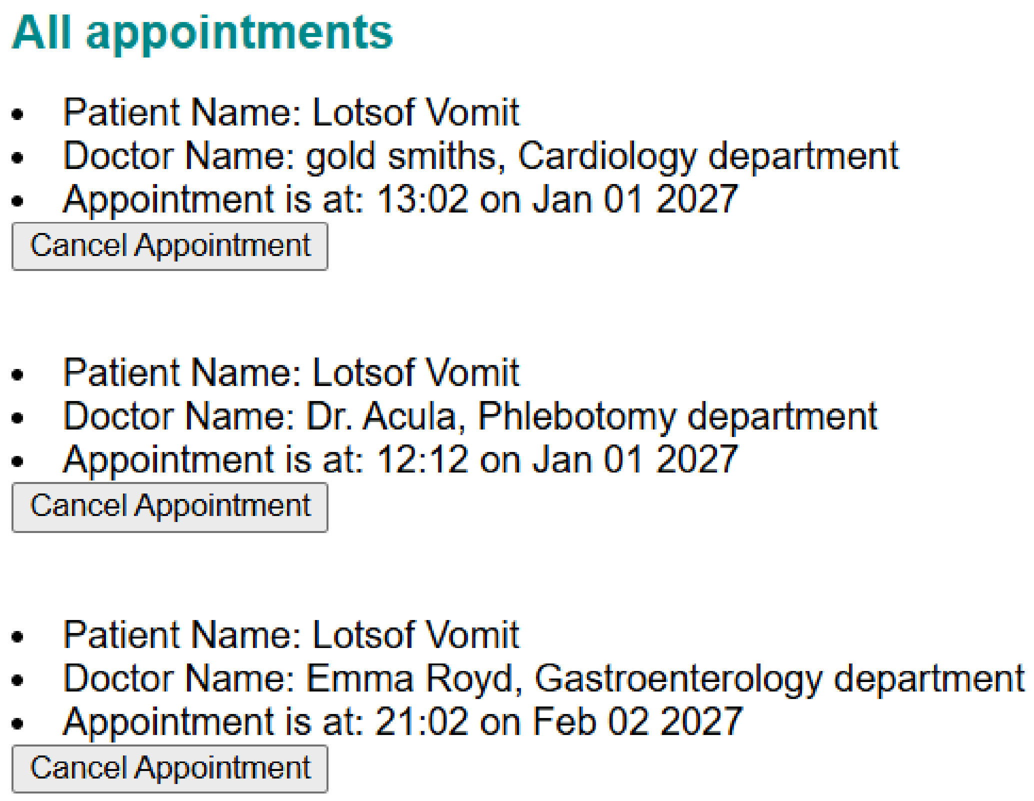

All appointments

- Patient Name: Im Ill
- Doctor Name: gold smiths, Cardiology department
- Appointment is at: 16:14 on Dec 11 2025

- Patient Name: Help Me
- Doctor Name: gold smiths, Cardiology department
- Appointment is at: 01:01 on Feb 13 2026
- **Appointment has been cancelled**

- Patient Name: Lotsof Vomit
- Doctor Name: gold smiths, Cardiology department
- Appointment is at: 13:02 on Jan 01 2027

Here is the same screen from the point of view of a patient, note that the only button available to them is to cancel the appointment, these leaves the power to mark an appointment as attended in the hands of the doctor.

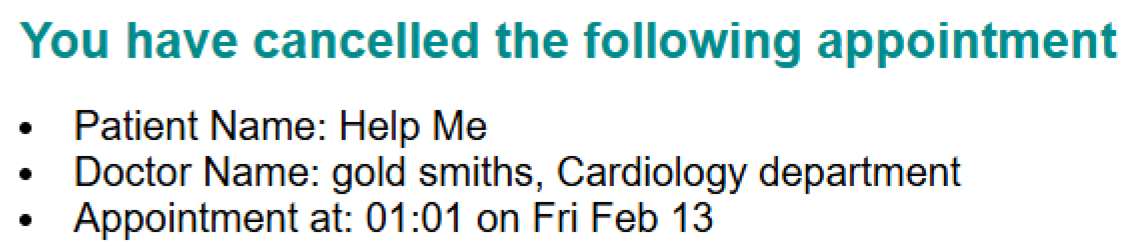

All appointments

- Patient Name: Lotsof Vomit
- Doctor Name: gold smiths, Cardiology department
- Appointment is at: 13:02 on Jan 01 2027

- Patient Name: Lotsof Vomit
- Doctor Name: Dr. Acula, Phlebotomy department
- Appointment is at: 12:12 on Jan 01 2027

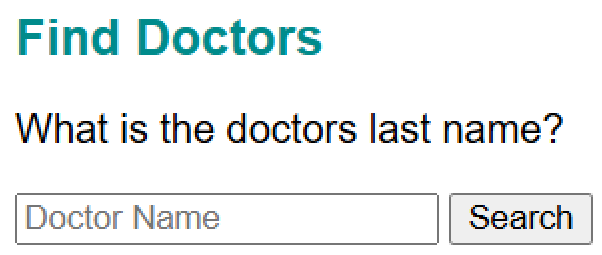
- Patient Name: Lotsof Vomit
- Doctor Name: Emma Royd, Gastroenterology department
- Appointment is at: 21:02 on Feb 02 2027

Confirmation screen for cancelling appointment

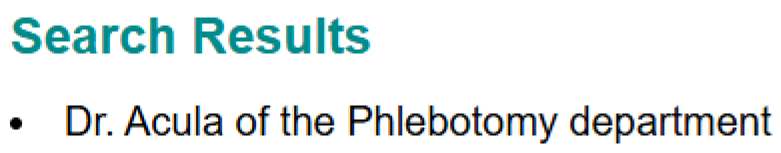

You have cancelled the following appointment

- Patient Name: Help Me
- Doctor Name: gold smiths, Cardiology department
- Appointment at: 01:01 on Fri Feb 13

Users are also able to search for a doctors based on their last name


Find Doctors

What is the doctors last name?


Search Results

- Dr. Acula of the Phlebotomy department

This uses `LIKE %last_name%` in the query, meaning if there were multiple results for Acula then they would also appear here.

Finally users are also able to logout, returning them to the original login screen.

If you would like to access the website from other accounts either register a new user or login with the following usernames. The default password for each user is apart from the user gold is `password`

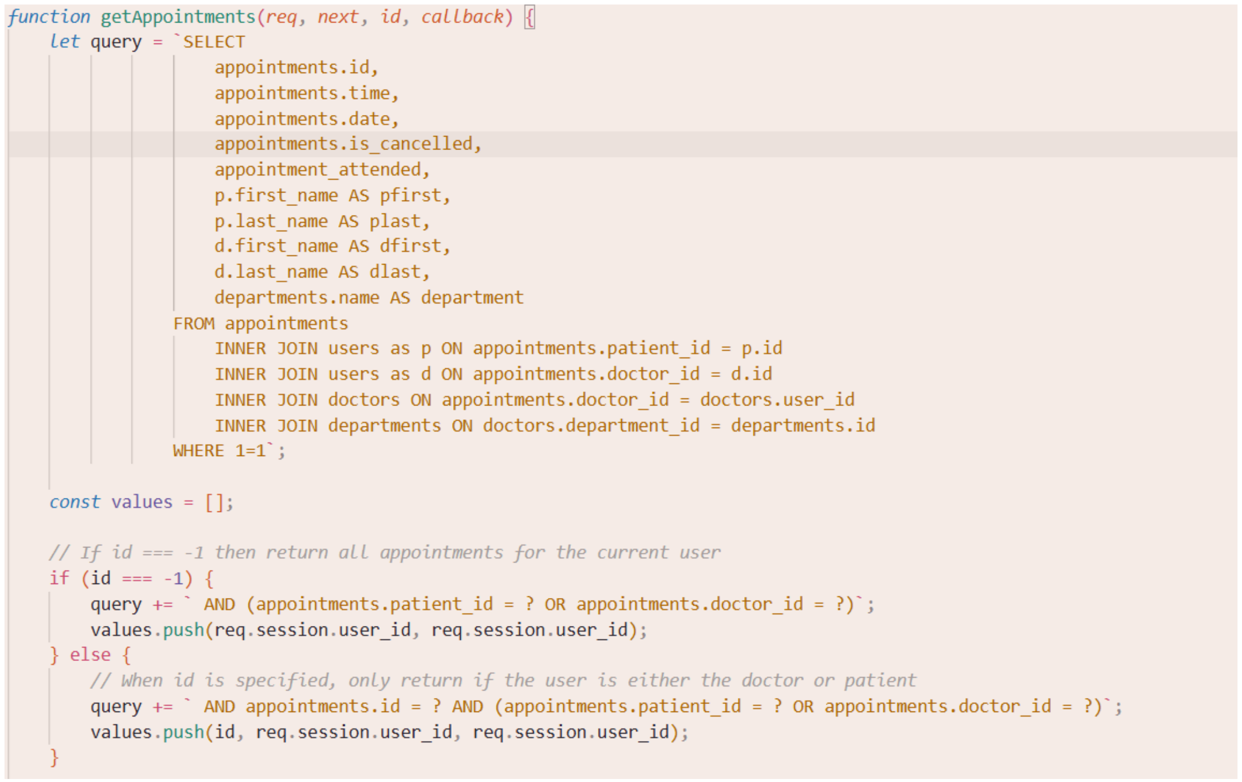
Patient usernames:

- illMan,
- badHeadache
- DyingMan
  Doctor usernames
- Second
- Third

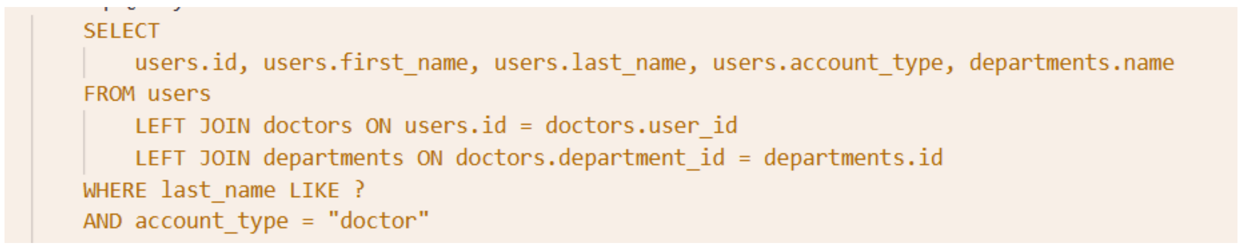
Advanced Techniques

- Database Normalisation
  - There is no data reuse in the database, each reference to another table is done strictly through IDs
  - Can be demonstrated in CREATE_DB.sql

    ```sql
    CREATE TABLE IF NOT EXISTS departments (
        id      INT AUTO_INCREMENT,
        name    VARCHAR(25) UNIQUE NOT NULL,
        PRIMARY KEY(id)
    );

    CREATE TABLE IF NOT EXISTS users (
        id                      INT AUTO_INCREMENT,
        email                   VARCHAR(255) UNIQUE NOT NULL,
        username                VARCHAR(255) UNIQUE NOT NULL,
        hashed_password         VARCHAR(255) NOT NULL,
        account_type            VARCHAR(6  ) NOT NULL,
        first_name              VARCHAR(255),
        last_name               VARCHAR(255),
        PRIMARY KEY(id)
    );

    CREATE TABLE IF NOT EXISTS patients (
        user_id         INT UNIQUE NOT NULL,
        date_of_birth   DATE NOT NULL,
        PRIMARY KEY (user_id),
        FOREIGN KEY (user_id) REFERENCES users(id)
    );

    CREATE TABLE IF NOT EXISTS doctors (
        user_id        INT UNIQUE NOT NULL,
        department_id  INT NOT NULL,
        availability   BIT(7) NOT NULL,
        PRIMARY KEY(user_id),
        FOREIGN KEY (user_id) REFERENCES users(id),
        FOREIGN KEY (department_id) REFERENCES departments(id)
    );
    ```
  -
- SQL Joins

    - In order to facilitate the previous point I had to make excessive use of SQL joins, often times stringing several tables together at once
    - Examples can be found throughout the project, such as utils/appointmentUtils.js or routes/authentication.js
    - Here is one example, in this one I needed to reference the user table twice through two different contexts, therefore I used inner join along with SQL variables(INNER JOIN users as p) so that the results from users could be returned twice.
    - 

```
function getAppointments(req, next, id, callback) {
    let query = `SELECT
                    appointments.id,
                    appointments.time,
                    appointments.date,
                    appointments.is_cancelled,
                    appointment_attended,
                    p.first_name AS pfirst,
                    p.last_name AS plast,
                    d.first_name AS dfirst,
                    d.last_name AS dlast,
                    departments.name AS department
                FROM appointments
                    INNER JOIN users as p ON appointments.patient_id = p.id
                    INNER JOIN users as d ON appointments.doctor_id = d.id
                    INNER JOIN doctors ON appointments.doctor_id = doctors.user_id
                    INNER JOIN departments ON doctors.department_id = departments.id
                WHERE 1=1`;

    const values = [];

    // If id === -1 then return all appointments for the current user
    if (id === -1) {
        query += ` AND (appointments.patient_id = ? OR appointments.doctor_id = ?)`;
        values.push(req.session.user_id, req.session.user_id);
    } else {
        // When id is specified, only return if the user is either the doctor or patient
        query += ` AND appointments.id = ? AND (appointments.patient_id = ? OR appointments.doctor_id = ?)`;
        values.push(id, req.session.user_id, req.session.user_id);
    }
```

    - And another, in this one I had to LEFT JOIN departments, this is because not all users have departments, therefore LEFT JOIN was done to ensure that ALL items are selected from the database and not just the ones with a department.
    - 

```
    SELECT
        users.id, users.first_name, users.last_name, users.account_type, departments.name
    FROM users
        LEFT JOIN doctors ON users.id = doctors.user_id
        LEFT JOIN departments ON doctors.department_id = departments.id
    WHERE last_name LIKE ?
    AND account_type = "doctor"
```

- Two level authorisation
    - This project has two levels of authorisation, patients and doctors. Each user is only allowed to see and interact with data related to themselves. Patients are unable to see the names or info of other patients, whereas doctors are able to see all patients when booking.
    - This is done throughout the project such as at views/add.ejs, views/appointments.ejs, utils/appointmentUtils.js, and route/authentication.ejs. here are some examples

- When users log in their account type is stored as a session variable, this is then checked at several points throughout to control what the user is able to see.

```
bcrypt.compare(req.body.password, result[0].hashed_password, (err, equal) => {
    if (err) return next(err);
    if (equal){
        req.session.user_id = result[0].id
        req.session.account_type = result[0].account_type
        // Makes account_type available in all ejs templates
        res.locals.account_type = req.session.account_type;
        res.send('Login Successful. Return  <a href='+'../'+'>Home</a>')
    } else {
        res.send('Login Unsuccesful Return  <a href='+'../'+'>Home</a>')
    }
})
```

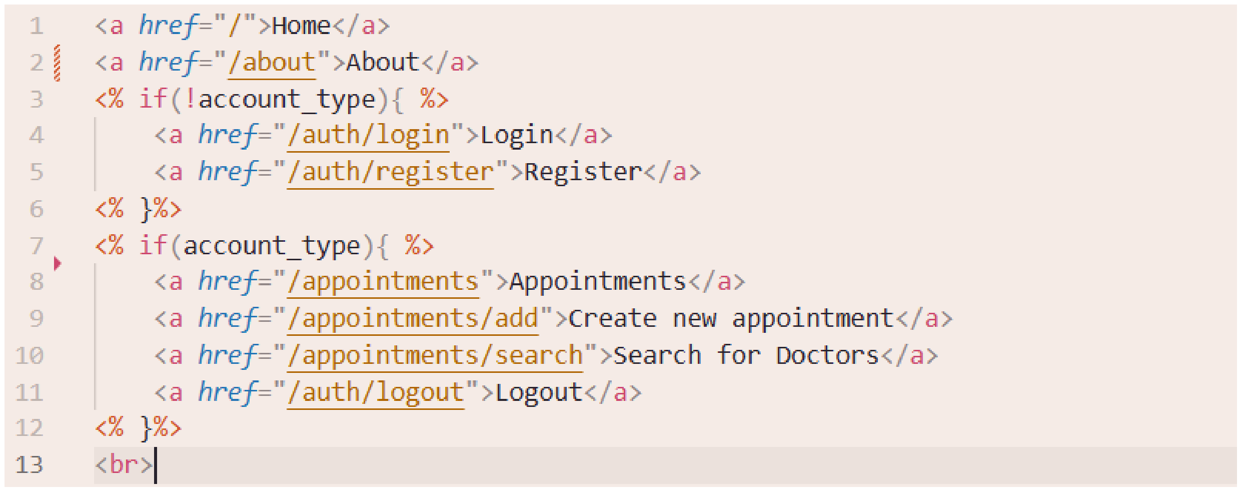
- Here is a section of EJS for adding appointment, this code checks the account type and displays data for the form accordingly. If a user is a patient then they can select from a list of doctors, whereas if they're a doctor they can select from a list of patients.

```
<form action="./add" method="post">
    <% if (account_type === "pat"){ %>
        <input type="hidden" name="patient"  value="<%=user_id %>">
        <label for="doctor">Select Doctor:</label>
        <select name="doctor" id="doctor">
            <% doctors.forEach(doctor => { %>
                <option value="<%= doctor.id %>"><%= doctor.first_name %> <%= doctor.last_name %></option>
            <% }) %>
        </select>
    <% }else if (account_type === "doctor"){ %>
        <input type="hidden" name="doctor"  value="<%=user_id %>">
        <label for="patient">Select Patient:</label>
        <select name="patient" id="patient">
        <% patients.forEach(patient => { %>
            <option value="<%= patient.id %>"><%= patient.first_name %> <%= patient.last_name %></option>
        <% }) %>
        </select>
    <% } %>
```

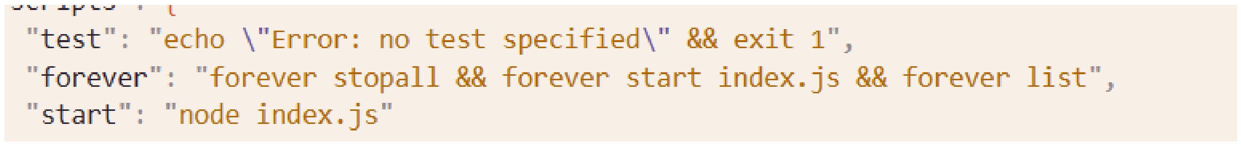
- The following code displays buttons for cancelling or marking an appointment as complete, each of the buttons are displayed dynamically depending on the is_cancelled and appointment_attended fields in the database. Patients are unable to view the mark attended button and can only cancel their appointments, whereas doctors can do both.

```
<% if(appointment.is_cancelled){ %>
   <li><b>Appointment has been cancelled</b></li>
<% } else if(appointment.appointment_attended){ %>
    <li><b>Appointment was attended</b></li>
<% } else { %>
    <input type="submit" value="Cancel Appointment" />
    <% if(account_type === "doctor"){ %>
    <input type="submit" formaction="/appointments/attended" value="Appointment Attended" />
    <% } %>
<% } %>
```

- The home page and header also dynamically display content based on if a user is logged in or not, if they are not logged in the only buttons they have available are login, register, or about. Whereas logged-in uses have the about button, the search button, the appointment buttons and the logout button.

- 

```
<a href="/">Home</a>
<a href="/about">About</a>
<% if(!account_type){ %>
    <a href="/auth/login">Login</a>
    <a href="/auth/register">Register</a>
<% }%>
<% if(account_type){ %>
    <a href="/appointments">Appointments</a>
    <a href="/appointments/add">Create new appointment</a>
    <a href="/appointments/search">Search for Doctors</a>
    <a href="/auth/logout">Logout</a>
<% }%>
<br>
```

- Advanced node scripts
  - I also added my own npm script `forever`, this allows me to quickly launch a new server on the virtual machine which will stop all previous forever process and launch a new one.
  - Can be found in package.json
  - 

```
  "test": "echo \"Error: no test specified\" && exit 1",
  "forever": "forever stopall && forever start index.js && forever list",
  "start": "node index.js"
```

AI Declaration

AI was not used in this project