Hospital Management Web Application Development

PrakharAtishayRudrakAkashSaras HOSPITAL

Database Management Systems Laboratory
CS39202

Group name: PARAS

Group members

Prakhar Singh	20CS10045
Atishay Jain	20CS30008
Rudrak Patra	20CS30044
Akash Das	20CS10006
Saras Umakant Pantulwar	20CS30046

Objective

To design a web application for a Hospital Management System.

Task done by the system

- Registering patient
- Scheduling appointment with doctors
- Maintaining patient information about diagnostics tests and treatments administered
- Maintaining information about doctors/healthcare professionals
- Storing admit/discharge information about the patients.

Intended users of this web application are:

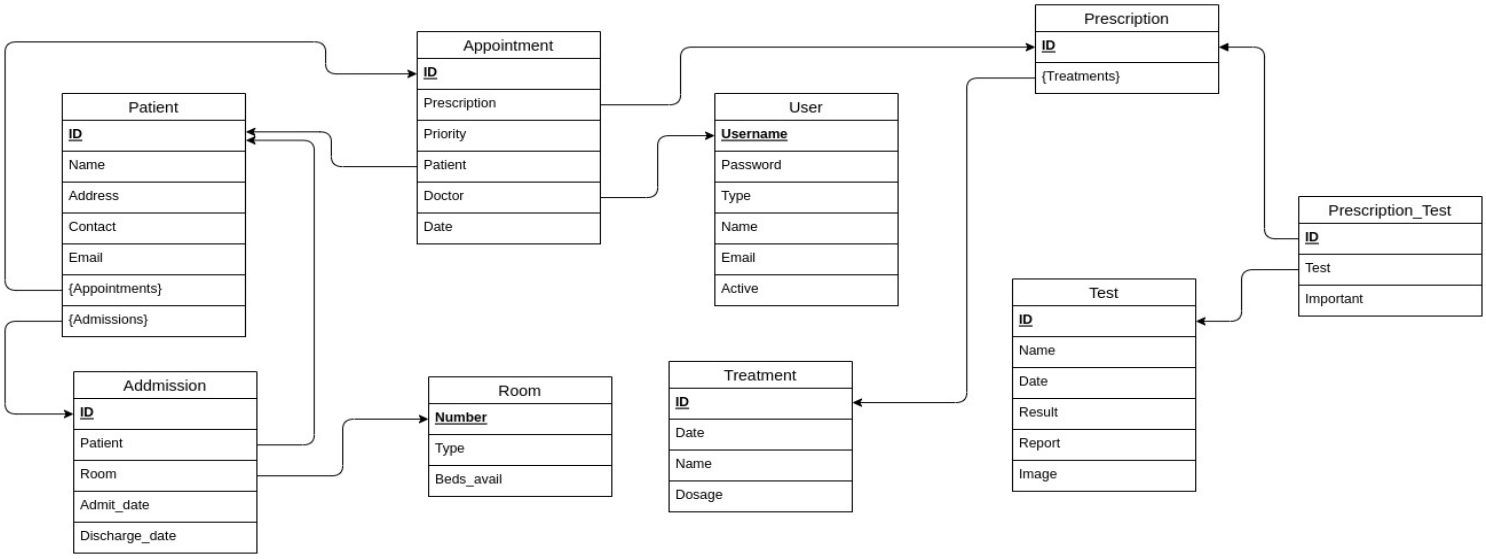
- Front Desk Operators: registers/admits/discharges patients
- Data Entry Operators: enters patient data about tests and treatments
- Doctors: query patient information
- Database Administrators: add/delete users

Functionalities implemented are:

- Patient registration/discharge and doctor appointment/test scheduling based on availability and priority.
- Doctor is notified about the appointments in a dashboard.
- For admitted patients, room is assigned based on available room capacity.
- For discharged patients, information is preserved and room occupancy is updated.
- The workflow also supports scheduling tests and treatments prescribed by doctors. Calendar is used for scheduling.
- Patient data entry – All the health information of a patient including test results and treatments administered are recorded. Also supports storage and display of images e.g., x-ray.
- Doctor dashboard – all the records of the patients treated by a doctor are displayed to them on the dashboard. Doctors can also query for any patient information and can also record drugs/treatments prescribed to a patient.
- Automated email reports are sent to a doctor about the health information of patients treated by them on a weekly basis, high priority events are emailed in an emergency manner.
- Database administrator is able to add/delete new users.
- Data security policy is implemented with suitable access control.

1. Schema of the underlying Database

Below is the E-R diagram for the underlying database.




```
USE Hospital;

create table User (
    `Username` varchar(256) primary key,
    `Password` text not null,
    `Type` varchar(16) not null,
    `Name` varchar(256) not null,
    `Email` varchar(256),
    `Active` boolean default true not null,
   CHECK (`Type` IN ('frontdesk','admin','dataentry', 'doctor'))
);

create table Room (
    `Number` int primary key,
    `Type` text not null,
    `Beds_avail` int not null
);

create table Test (
    `ID` int primary key auto_increment,
    `Name` text not null,
    `Date` datetime,
    `Result` text,
    `Report` MEDIUMBLOB,
```

```
    `Image` MEDIUMBLOB
);

create table Treatment (
    `ID` int primary key auto_increment,
    `Date` datetime not null,
    `Name` text not null,
    `Dosage` text not null
);

create table Patient (
    `ID` int primary key auto_increment,
    `Name` text not null,
    `Address` text not null,
    `Contact` text,
    `Email` text
);

create table Prescription (
    `ID` int primary key auto_increment
);

create table Prescription_Treatment (
    `ID` int,
    `Treatment` int,
    foreign key (`Treatment`) REFERENCES `Treatment`(`ID`),
    foreign key (`ID`) REFERENCES `Prescription` (`ID`),
    primary key (`Treatment`)
);

create table Prescription_Test (
    `ID` int,
    `Test` int,
    `Important` boolean default false not null,
    foreign key (`Test`) REFERENCES `Test`(`ID`),
    foreign key (`ID`) REFERENCES `Prescription`(`ID`),
    primary key (`Test`)
);

create table Admission (
    `ID` int primary key auto_increment,
    `Patient` int,
    `Room` int,
```

```
    `Admit_date` datetime not null,
    `Discharge_date` datetime,
    foreign key (`Patient`) REFERENCES `Patient`(`ID`),
    foreign key (`Room`) REFERENCES `Room`(`Number`)
);

create table Appointment (
    `ID` int primary key auto_increment,
    `Prescription` int,
    `Priority` int not null,
    `Patient` int,
    `Doctor` varchar(256),
    `Date` date not null,
    foreign key (`Prescription`) REFERENCES `Prescription`(`ID`),
    foreign key (`Patient`) REFERENCES `Patient`(`ID`),
    foreign key (`Doctor`) REFERENCES `User`(`Username`)
);

create table Patient_Appointment (
    `ID` int,
    `Appointment` int,
    foreign key (`Appointment`) REFERENCES `Appointment`(`ID`),
    primary key (`Appointment`)
);

create table Patient_Admission (
    `ID` int,
    `Admission` int,
    foreign key (`Admission`) REFERENCES `Admission`(`ID`),
    primary key (`Admission`)
);
```

2. Languages and Tools used

For our web development project, we used a combination of front-end and back-end technologies to create a dynamic and responsive user interface, handle server-side logic, and store and manage data.

The primary technologies and tools we used are

- **Html**
- **CSS**
- **JS/TS**
- **React.js,**
- **Node.js, and**
- **MySQL**

We also used

- Express.js
- TailwindCSS
- vite
- npm
- Yarn
- Git and Github

React

React.js is a popular front-end JavaScript library that we used to build the user interface of our web application. React's component-based architecture made it easy to create reusable UI components, which helped us to reduce the amount of code we needed to write and made our code more organised and modular. Additionally, React's virtual DOM and efficient rendering algorithm helped us to optimise the performance of our application. “ReactRouter” is being used to route to different pages while still maintaining a single page app, which improves the user experience.

Node.js

Node.js is a server-side JavaScript runtime environment that we used to build the back-end of our web application. Node.js allowed us to write server-side logic using JavaScript, which made it easier to maintain consistency between the front-end and back-end of our application. We used Node.js to handle requests from the client and communicate with the database.

MySQL

MySQL is a widely-used relational database management system that we used to store and manage data for our web application. MySQL provided us with a robust and scalable platform for storing data, and we used it to store user information, application data, and other important information. We used Node.js and the MySQL driver to connect to the database and perform CRUD operations (create, read, update, delete) on our data.

In addition to these core technologies, we also used a number of other tools and libraries to aid in development, testing, and deployment. These includes:

NPM (Node Package Manager)

To manage dependencies and packages.

Express.js

Express.js a web application framework for Node.js, to simplify the process of building APIs.

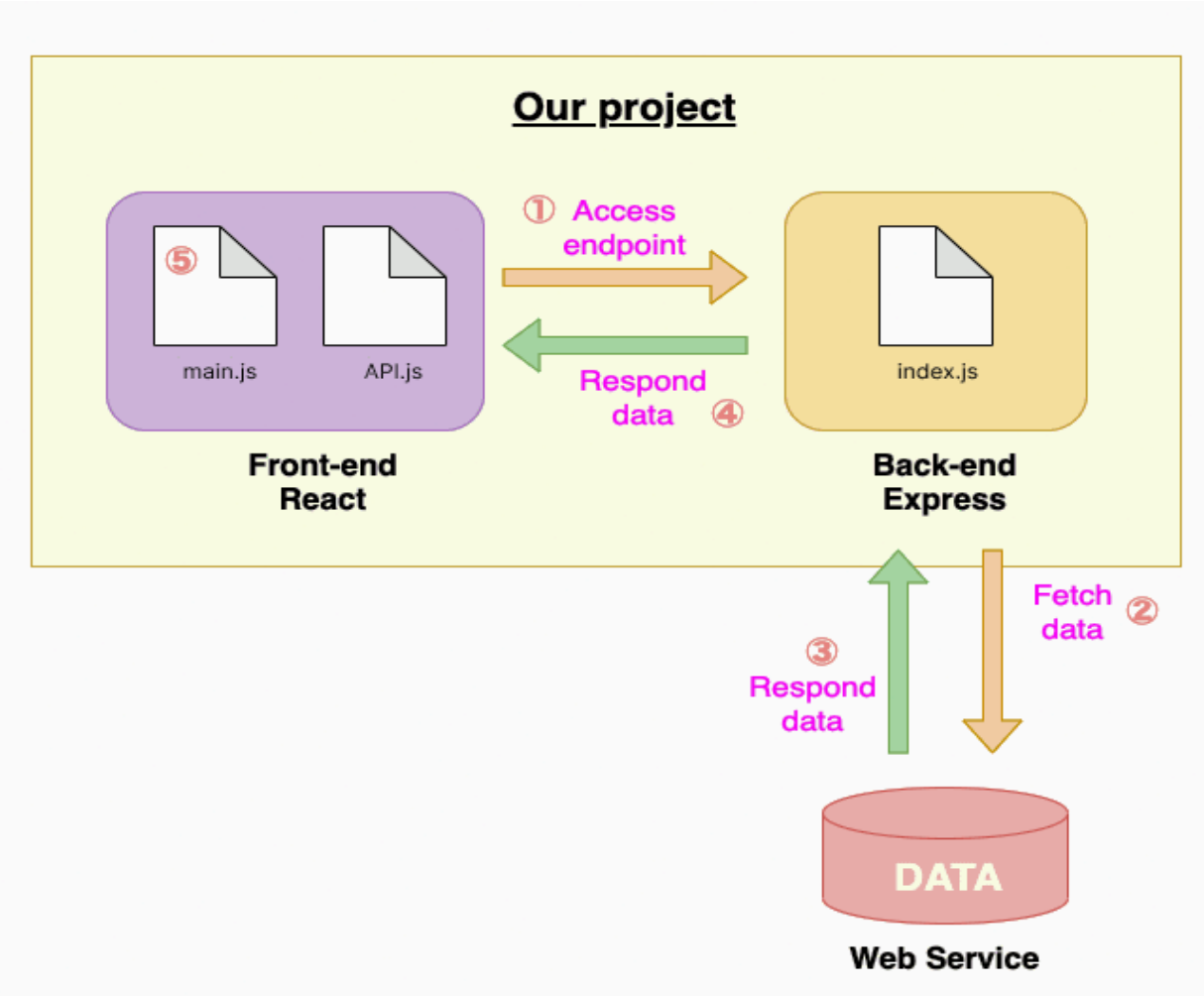
Git and GitHub

We used Git for version control and collaboration. We used Github to save the project on the cloud.

Overall, the combination of React.js, Node.js, and MySQL provided us with a powerful and flexible set of tools to create a high-quality web application with a great user experience.

We can have the database ,the backend, the frontend running on different machines.
Currently we have uploaded the database on the cloud.

3. Triggers and workflows implemented



Backend

Index.js

1. In this file we start by importing all the required libraries.
2. Then we **connect** to the database (we are using **Mysql**)
3. **Express**.js provides us an express app containing two handy functions app.**get**(path,callback) and app.**post**(path,callback).The callback function takes two parameters: **req** (containing info on the actual req received) and **res** (containing methods to send a response back to the client).
    a) So, in this first we run the **isAuth** function imported from auth.js from the same directory, in which we check the

authentication of the user who has requested the **get**/**post**. If the authentication is complete then we move forward.

b) As many of the features are accessible to only some types of users (like only registration of patients, appointment, admission are done by front desk, treatment and test entry is done by dataentry) so we then check the **type** of the user and if it **satisfies** the condition then we proceed further.

c) **sql** is a variable that stores the query required for processing the request. We pass this to the connection.query method and according to the result (error or correct result to the query) of this method we return the response in json format.

Following is one **example** in which we want to get the admission history of a patient. So if the user who made this request is frontdesk then only this process the query. So on running the query sql in the following code we get the history of the patient (whose id we get from the request req).

```
app.get("/getAdmissionHistory", (req, res) => {
 console.log({ body: req.headers });
 isAuth(connection, req, res, (user) => {
   console.log({ user });
   if (user.Type == "frontdesk") {
     let sql = `SELECT Admission.ID AS appID, Admission.Room AS
Room,Admission.Admit_date As Admit_date, Admission.Discharge_date AS
Discharge_Date,
             Patient.Name AS Name, Patient.ID AS Patient
             FROM Admission
             JOIN Patient ON Admission.Patient = Patient.ID
             ORDER BY Admission.ID DESC LIMIT 100;`;
     console.log(sql);
     connection.query(sql, function (err, result) {
       if (err) {
         res.json({ status: "error" });
       } else {
         console.log(result);
         res.json(result);
       }
     });
   }
 });
});
```

Auth.js

1. This file **contains** the definition of **authentication** function. It takes **connection** (to the mysql server), **req**, **res** (HTTP request and response) and **onSuccess** (a callback function that will run on successful authentication) as parameters.
2. In this function we first **decode** the **username** and **password** from the req and then using the following query we check if the user is valid or not and accordingly give the response. That is on **successful verification** we run **onSuccess** function else if we encounter any **error** then we **return** the **error** in json format.

For authentication we have stored the username,password and type for each user.

Any fetch call to the server requires username and password. When we login on the client side we temporarily store the username and password while he/she is logged in.

Middleware (Queries and Fetch Calls)

We have listed all the queries that are used in the server to read and write in the database along with their corresponding frontend **fetch** call that uses that.

(you can skip this part if you want to)

function isAuth

```
`select * from User where Username="${username}" and
Password="${password}" and Active=1;`
```

```
Is used in every call
```

```
app.get("/users"

`Select * from User WHERE Active;`


const getUsers = async ({ username, password })
```

```
app.post("/users",

`INSERT INTO User (Username, Password, Type, Name, Email) VALUES
('${req.body.username}', '${req.body.password}', '${req.body.type}',
'${req.body.name}', '${req.body.email}');`



const addUser = async (
   adminUsername,
   adminPassword,
   username,
   password,
   name,
   email,
   type
)
```

app.post("/users/delete"

```
`SELECT * FROM User WHERE Username = '${req.body.username}';`

`UPDATE User SET Active=0 where Username="${userToDelete.Username}"`

`SELECT Appointment.ID AS appID, Patient.ID AS pID, Patient.Name AS
pName, Appointment.Date AS date, Appointment.Priority AS priority,
Patient.Contact AS Contact, Patient.Email AS Email FROM Appointment,
Patient WHERE Appointment.Doctor = '${userToDelete.Username}' AND
Appointment.Patient = Patient.ID AND Appointment.Prescription is NULL
AND Appointment.Date >= '${date}';`

`SELECT Username FROM User
                  LEFT JOIN Appointment ON User.Username =
Appointment.Doctor AND Appointment.Prescription IS NULL AND
Appointment.Date >= CURDATE()
                  WHERE User.Type = 'doctor' AND Active
                  GROUP BY Username
                  ORDER BY COUNT(Username);`

 `UPDATE User SET Active=1 where Username="${userToDelete.Username}"`

`UPDATE Appointment SET Doctor='${doctorApp}' WHERE
ID='${appointment.appID}';`



const deleteUser = async (adminUsername, adminPassword, username)
```

```
app.get("/doctor/appointments"

`SELECT Appointment.ID AS appID, Patient.ID AS pID, Patient.Name AS
pName, Appointment.Date AS date, Appointment.Priority AS priority,
Patient.Contact AS Contact, Patient.Email AS Email
              FROM Appointment, Patient WHERE Appointment.Doctor =
'${user.Username}' AND Appointment.Patient = Patient.ID AND
Appointment.Prescription is NULL AND Appointment.Date >= CURDATE();`



const getAppointments = async (username, password)
```

```
app.get("/frontdesk/patients"

`SELECT Patient.*, Admission.Room AS Room, Room.Type AS Type,
         CASE WHEN Admission.ID IS NOT NULL AND
Admission.Discharge_date IS NULL
           THEN true
           ELSE false
         END AS admitted
       FROM Patient
       LEFT JOIN Admission ON Patient.ID = Admission.Patient AND
Admission.Discharge_date IS NULL
       LEFT JOIN Room ON Room.Number = Admission.Room
       ORDER BY Patient.ID DESC;
     `;




const getAllPatientsForFrontDesk = async ({ username, password })
```

app.get("/dataentry/appointments/noprescription"

```
`SELECT Appointment.ID as appID, Patient.ID as pID, Patient.Name as
pName, User.Name as dName, Date as date FROM Appointment, Patient,User
WHERE Appointment.Doctor=User.Username AND Prescription IS NULL AND
Patient=Patient.ID AND Appointment.Date <= CURDATE();`
```

```
const getAppointmentsWithNoPrescription = async ({
 username,
 password,
})
```

```
app.get("/dataentry/test/update"

`SELECT Test.*,Patient.ID as pID,Patient.Name as pName FROM
Test,Prescription_Test,Appointment,Patient WHERE
     Test.ID=Prescription_Test.Test AND
Prescription_Test.ID=Appointment.Prescription AND
Appointment.Patient=Patient.ID
     AND
     Result IS NULL AND Test.Date < LOCALTIME();`



const fetchTestUpdate = async ({ username, password })
```

```
app.get("/doctor/patients

`SELECT DISTINCT Patient.ID as ID, Patient.Name AS Name,
Patient.Address AS Address,Patient.Contact AS Contact, Patient.Email AS
Email
              FROM Appointment, Patient WHERE Appointment.Doctor =
'${user.Username}' AND Appointment.Patient = Patient.ID`




const getAllPatientsForDoctor = async ({ username, password })
```

```
app.get("/getAdmissionHistory"

`SELECT Admission.ID AS appID, Admission.Room AS
Room,Admission.Admit_date As Admit_date, Admission.Discharge_date AS
Discharge_Date,
             Patient.Name AS Name, Patient.ID AS Patient
             FROM Admission
             JOIN Patient ON Admission.Patient = Patient.ID
             ORDER BY Admission.ID DESC LIMIT 100;`



const fetchAdmissionHistory = async ({ username, password })
```

```
app.get("/getRescheduling"

`SELECT Patient.*, Appointment.Date AS appDate, Appointment.ID AS appID
     FROM Patient, Appointment
     WHERE Patient.ID = Appointment.Patient AND CURDATE() >=
Appointment.Date AND Appointment.Prescription IS NULL;
     `



const fetchRescheduling = async ({ username, password })
```

```
app.get("/getScheduleTest"
      `SELECT Patient.*, Test.ID AS testID, Test.Name AS testName
           FROM Patient, Appointment, Prescription AS P,
      Prescription_Test AS T, Test
           WHERE Patient.ID = Appointment.Patient AND
      Appointment.Prescription = P.ID AND P.ID = T.ID AND T.Test =
      Test.ID AND Test.Date IS NULL;`
```

```
const fetchScheduleTest = async ({ username, password })
```

app.get("/getRescheduleTest"

```
`SELECT Patient.*, Test.ID AS testID, Test.Name AS testName, Test.Date
AS Date
     FROM Patient, Appointment, Prescription AS P, Prescription_Test AS
T, Test
     WHERE Patient.ID = Appointment.Patient AND
Appointment.Prescription = P.ID AND P.ID = T.ID AND T.Test = Test.ID
AND CURDATE() > Test.Date AND Test.Result IS NULL;
     `;
```

```
const fetchRescheduleTest = async ({ username, password })
```

```
app.post("/discharge"

`Select Admission.ID, Admission.Room AS Room from Admission WHERE
Admission.Patient = ${req.body.patientId} AND Admission.Discharge_date
IS NULL;`

`UPDATE Admission SET Discharge_date = '${date.slice(
            0,
            10
          )}' WHERE ID = ${result[0].ID};`


`UPDATE Room SET Beds_avail = Beds_avail + 1 WHERE Number =
${roomNumber};`




const dischargePatient = async (username, password, patientId)
```

```
app.post("/register"

`INSERT INTO Patient (Name, Address, Contact, Email) VALUES
('${req.body.name}', '${req.body.Address}', '${req.body.contact}',
'${req.body.email}');`



const registerPatient = async (
 username,
 password,
 name,
 Address,
 contact,
 email
)
```

```
app.post("/dataentry/prescription"

`INSERT INTO Test (Name) VALUES `
+= `('${test.name}'), `
`SELECT 0;`

`INSERT INTO Treatment (Date, Name, Dosage) VALUES `
+= `('${treatment.date}', '${treatment.name}', '${treatment.dosage}'),
`
`SELECT 0;`

`INSERT INTO Prescription VALUES ();`

`INSERT INTO Prescription_Test VALUES `
`(${prescriptionId}, ${testIds + i}, ${imp}), `
`SELECT 0;`

`INSERT INTO Prescription_Treatment VALUES `
`(${prescriptionId}, ${treatmentIds + i}), `
`SELECT 0;`

`UPDATE Appointment SET Prescription = ${prescriptionId} WHERE ID =
${req.body.appID};`



const addPrescription = async (props)
```

```
app.post("/dataentry/testresult"

`UPDATE Test SET Result = '${req.body.result}', Report =
x'${req.body.report}', Image = x'${req.body.image}' WHERE ID =
${req.body.ID};`
SELECT ID,Important FROM Prescription_Test WHERE Test =
${req.body.ID};`

`SELECT User.Name as dName,Patient.Name as pName, Appointment.Patient
as pID, User.Email from Appointment,Patient,User WHERE Prescription =
${Prescription} AND Doctor=Username AND
Patient.ID=Appointment.Patient;`

`select * from Test where ID=${req.body.ID}`



const dataentryAddResult = async ({ username, password }, props)
```

```
app.post("/admit"

`SELECT Number FROM Room WHERE Type = '${req.body.type}' ORDER BY
Beds_avail LIMIT 1;`
`INSERT INTO Admission (Patient, Room, Admit_date) VALUES
('${req.body.patientId}', ${roomNumber}, '${date}');`

`UPDATE Room SET Beds_avail = Beds_avail - 1 WHERE Number =
${roomNumber};`

`INSERT INTO Patient_Admission (ID, Admission) VALUES
(${req.body.patientId}, ${AdmitId});`



const admitPatient = async (username, password, patientId, roomType)
```

app.post("/appointment/schedule"

```
`SELECT Username FROM User
                    LEFT JOIN Appointment ON User.Username =
Appointment.Doctor AND Appointment.Prescription IS NULL AND
Appointment.Date >= CURDATE()
                    WHERE User.Type = 'doctor' AND Active
                    GROUP BY Username
                    ORDER BY COUNT(Username);`

`INSERT INTO Appointment (Patient, Doctor, Date, Priority) VALUES
('${req.body.patientId}', '${doctorApp}', '${req.body.date}',
'${req.body.priority}');`

`INSERT INTO Patient_Appointment (ID, Appointment) VALUES
(${req.body.patientId}, ${AppId});`
```

```
const scheduleAppointment = async (
   username,
   password,
   patientId,
   date,
   priority
)
```

```
app.post("/getTreatment"

`select Appointment.ID AS appID,
                        Treatment.ID AS treatmentID,
                        Treatment.Name AS treatmentName,
                        Treatment.Dosage AS Dosage,
                        Treatment.Date AS Date
                from Treatment, Prescription_Treatment,Appointment
                where Appointment.Patient = '${req.body.patientId}' and
Appointment.Prescription = Prescription_Treatment.ID and
Prescription_Treatment.Treatment = Treatment.ID;`



const getTreatments = async (username, password, patientId)
```

```
app.post("/getTest"

`select Appointment.ID AS appID, Test.ID AS ID , Test.Name AS Name,
Test.Date AS Date, Test.Result AS Result, Test.Report AS Report,
Test.Image AS Image from Test, Prescription_Test, Appointment where
Appointment.Patient = "${req.body.patientId}" and
Appointment.Prescription = Prescription_Test.ID and
Prescription_Test.Test = Test.ID;`


const getTests = async (username, password, patientId)
```

```
app.post("/appointment/updateSchedule"

`UPDATE Appointment SET Date='${req.body.date}',
Priority=${req.body.priority} WHERE ID = ${req.body.appID};`

const updateAppointment = async (
 username,
 password,
 appID,
 date,
 priority
)
```

```
app.post("/test/schedule"

`UPDATE Test SET Date='${req.body.date}' WHERE ID =
${req.body.testID};`



const scheduleTest = async (username, password, testID, date)
```

Client

Main.tsx

The main starter file for react

Index.css

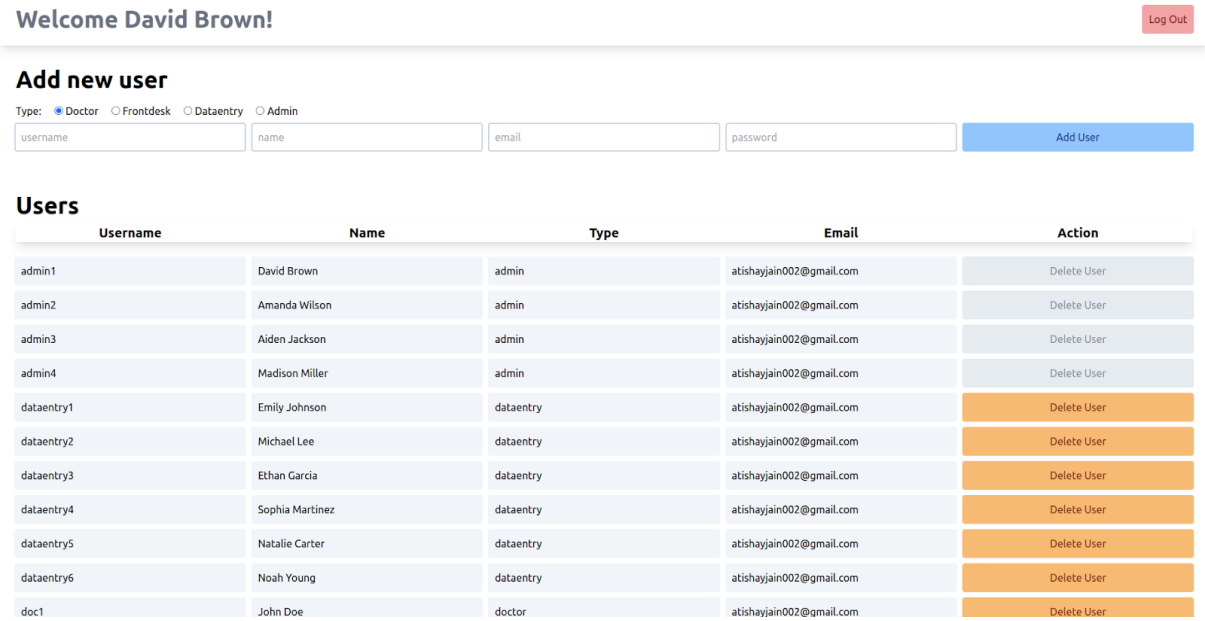
The main css stylesheet that contains generic styling for entire web application

Admin.tsx

PATH: /admin

- Add new user
  Security rules
  Usernames must be at least > 6 characters
  Passwords must be 8 characters long…
  (refer to backend/index.js `app.post("/users"...) for full info`

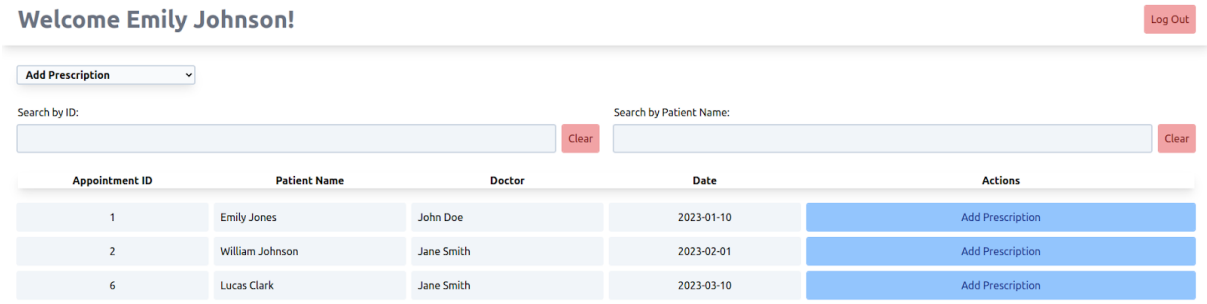
- Delete an existing user (records of the user still remains)
- **Deleting** a **Doctor** who has **pending** appointments will **reassign** his/her pending appointments **uniformly** to other doctors.

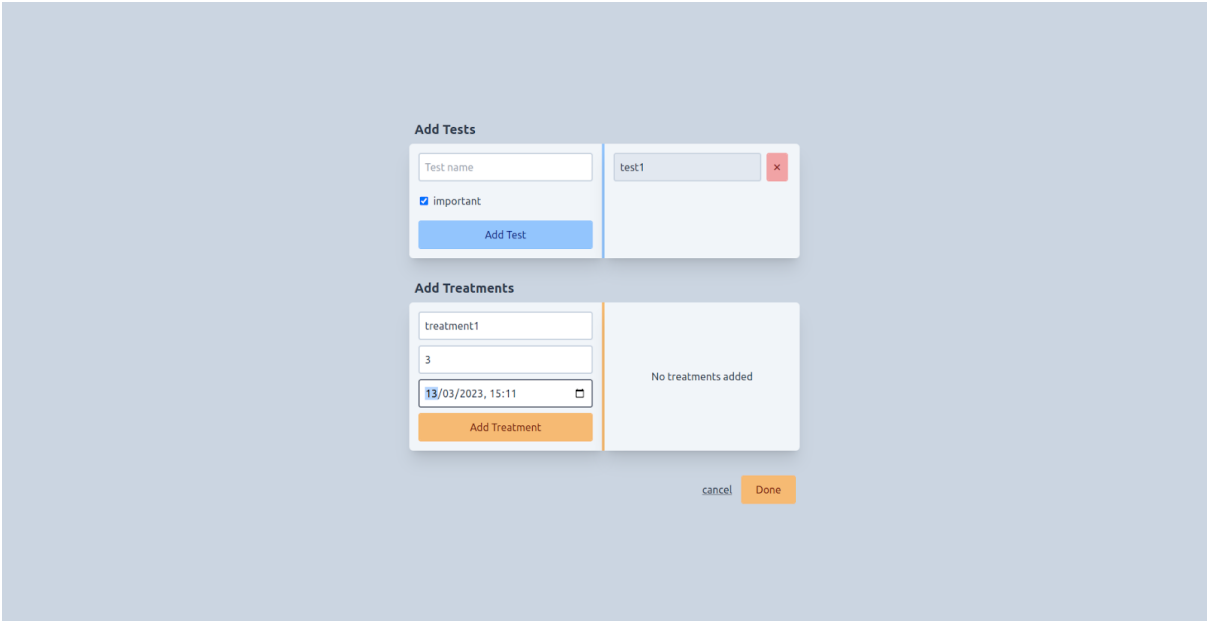


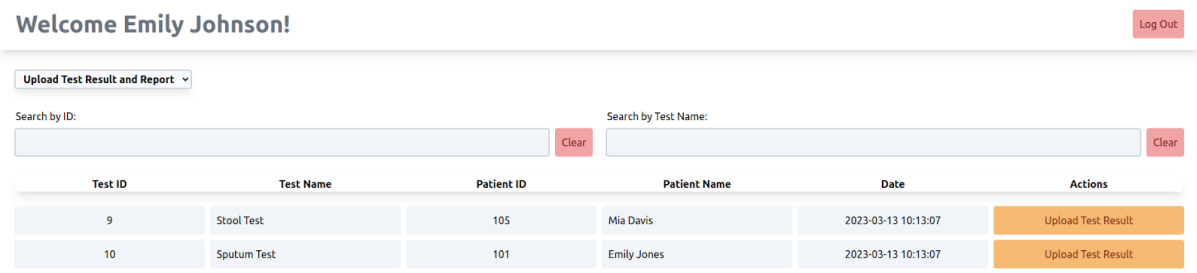
DataEntry.tsx

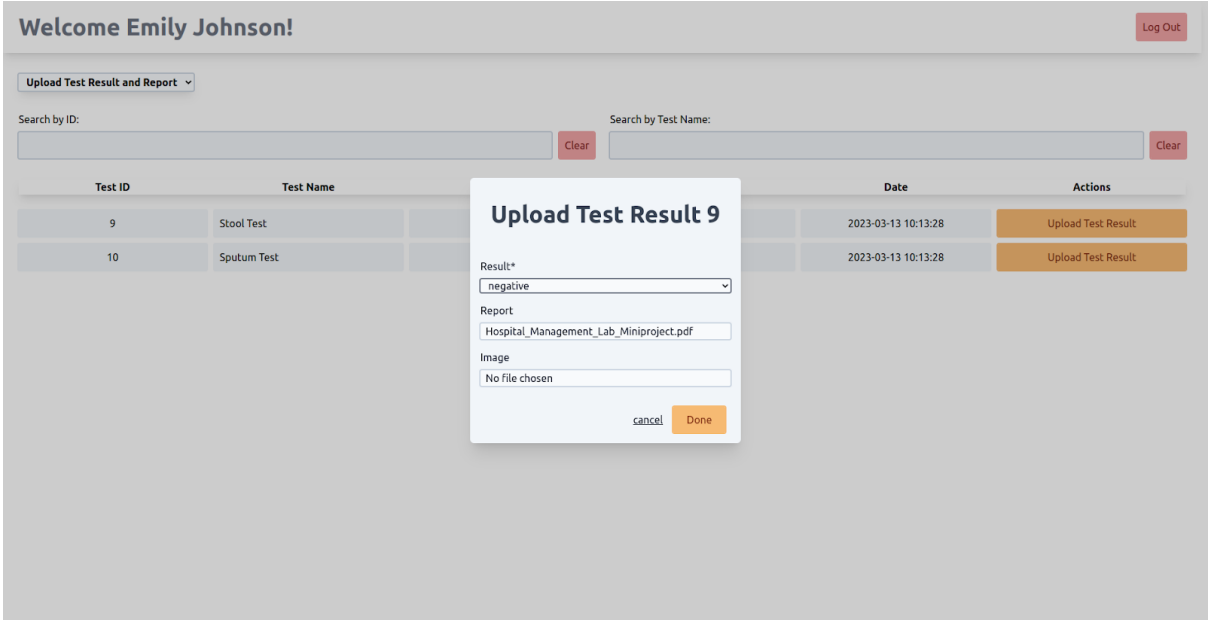
PATH: /dataentry

- Add prescriptions (tests and treatments) to appointments that are supposedly done
- Can mark a test to be **important** which will **emailed to the doctor** as soon as the results are **entered in the system**
- Add test results and reports to scheduled tests





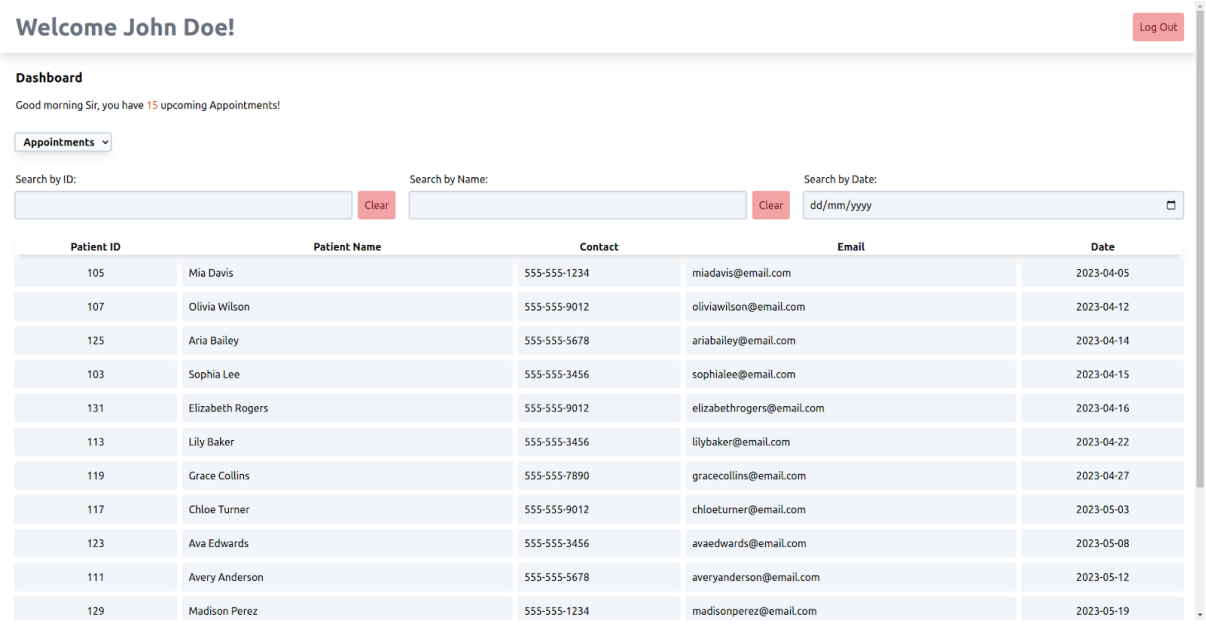

Welcome Emily Johnson!
Log Out
Upload Test Result and Report
Search by ID:
Clear
Search by Test Name:
Clear
Test ID
Test Name
Patient ID
Patient Name
Date
Actions
9
Stool Test
105
Mia Davis
2023-03-13 10:13:07
Upload Test Result
10
Sputum Test
101
Emily Jones
2023-03-13 10:13:07
Upload Test Result

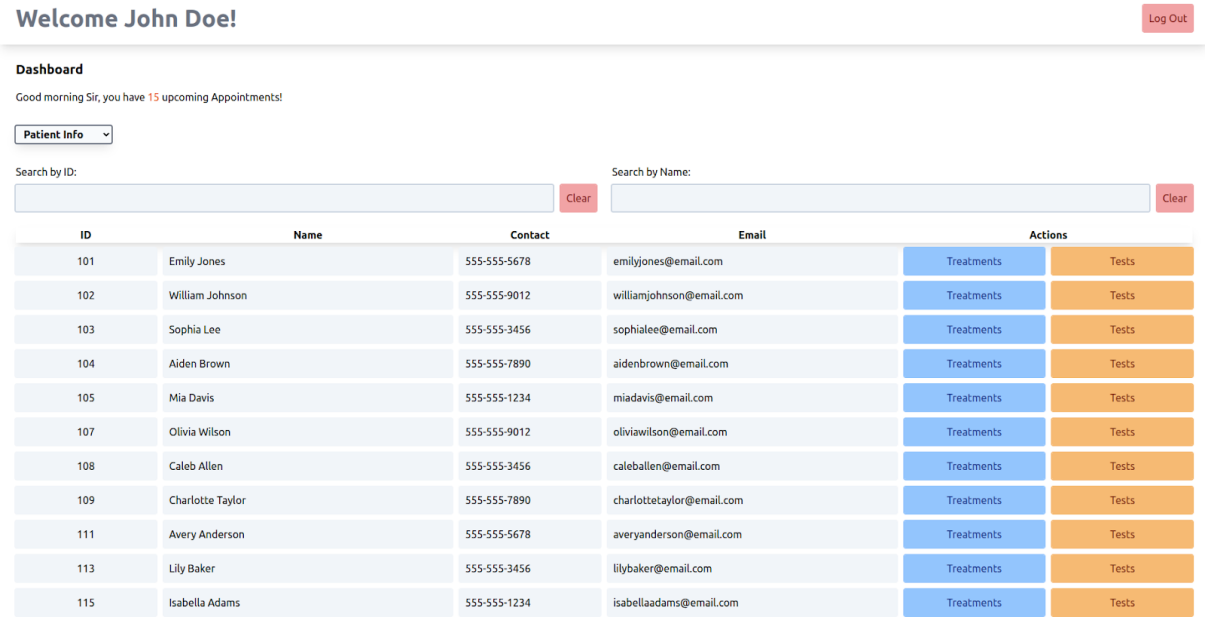

Welcome Emily Johnson!
Log Out
Upload Test Result and Report
Search by ID:
Clear
Search by Test Name:
Clear
Test ID
Test Name
Date
Actions
9
Stool Test
2023-03-13 10:13:28
Upload Test Result
10
Sputum Test
2023-03-13 10:13:28
Upload Test Result
Upload Test Result 9
Result*
negative
Report
Hospital_Management_Lab_Miniproject.pdf
Image
No file chosen
cancel
Done

Doctor.tsx

PATH: /doctor

- Can see the upcoming appointments they are assigned to.
- Gets an **email report** of all his/her patients on a **weekly** basis
- Can query patient info (their name, tests**…**
  see report pdfs + images, treatments …)





Treatments of patient 101

Name	Date	Dosage
Painkiller	2022-01-02 06:30:00	100mg, as needed
Steroid	2022-01-04 10:30:00	10mg, once a day

Close

Tests of patient 101

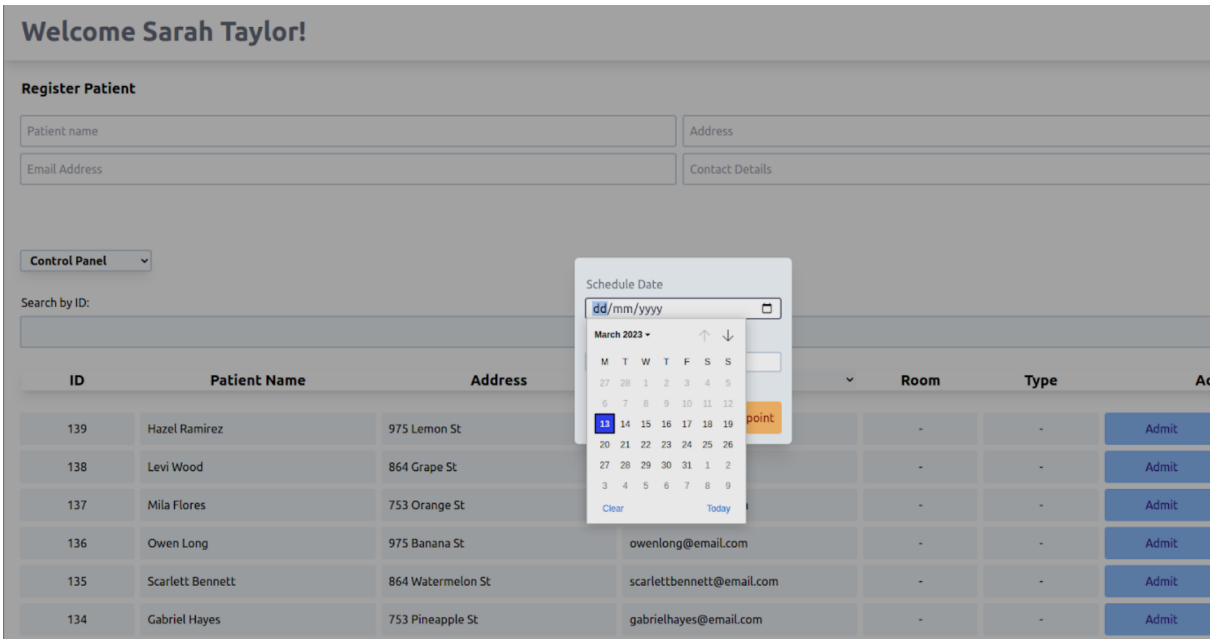
Name	Date	Result	Report	Image
Blood Test	2022-01-01 10:30:00	Positive	None	None
Sputum Test	2023-03-13 10:12:00	positive	Open	Open

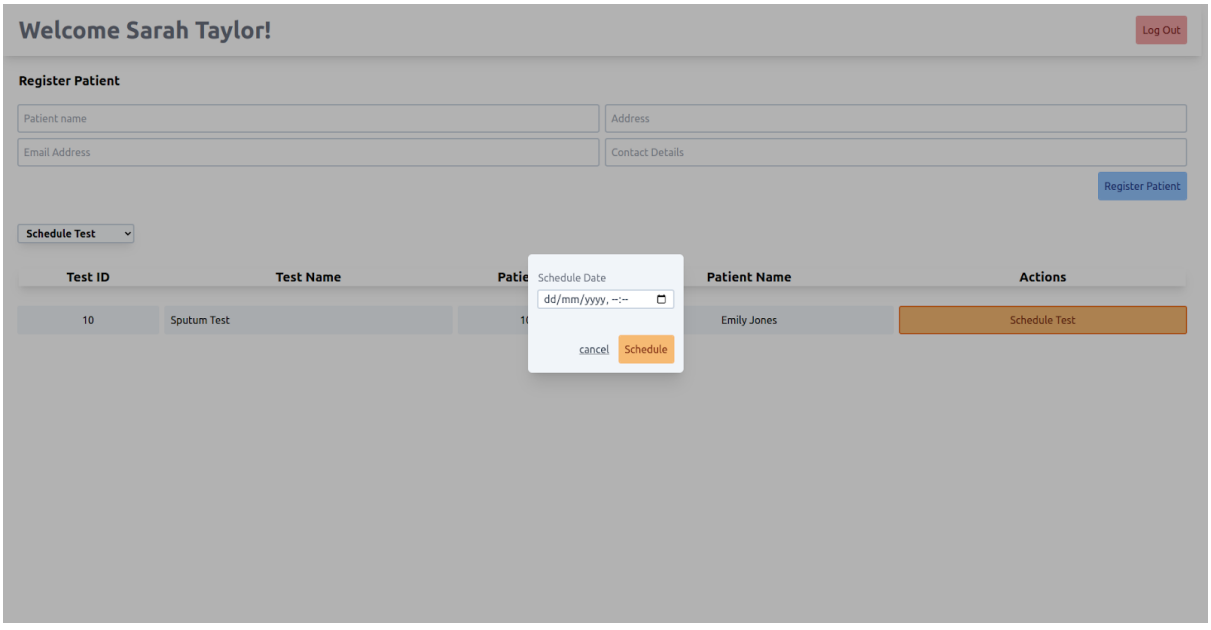
Close

FrontDesk.tsx

PATH: /frontdesk

- Can register for a patient
- Can admit / discharge the patient
- Schedule an appointment for a registered patient **using Calendar**
- Schedule Test for Patients who have tests in their prescription
- Reschedule Appointments and Tests if required
- Can see Admission History of Patients





Welcome Sarah Taylor!

Log Out

Register Patient

Patient name | Address

Email Address | Contact Details

Register Patient

Rescheduling

Appointment | Test

ID	Patient Name	Address	Email	Past Date	Actions
101	Emily Jones	456 Elm St	emilyjones@email.com	2023-03-13	Reschedule
102	William Johnson	789 Oak St	williamjohnson@email.com	2023-03-13	Reschedule
106	Lucas Clark	147 Cherry St	lucasclark@email.com	2023-03-13	Reschedule
139	Hazel Ramirez	975 Lemon St	hazelramirez@email.com	2023-03-13	Reschedule
137	Mila Flores	753 Orange St	milaflores@email.com	2023-03-13	Reschedule
134	Gabriel Hayes	753 Pineapple St	gabrielhayes@email.com	2023-03-13	Reschedule

Welcome Sarah Taylor!

Log Out

Register Patient

Patient name | Address

Email Address | Contact Details

Register Patient

Admission History

ID	Patient ID	Patient Name	Admit Date	Discharge Date	Room No.
20	119	Grace Collins	2023-03-13	-	209
19	118	Daniel Hall	2023-03-13	2023-02-09	208
18	117	Chloe Turner	2023-03-13	-	205
17	116	Benjamin Lewis	2023-03-13	2023-02-05	207
16	115	Isabella Adams	2023-03-13	-	206
15	114	Alexander Green	2023-03-13	2023-02-01	205
14	113	Lily Baker	2023-03-13	-	203
13	112	Mason Scott	2023-03-13	2023-01-28	202
12	111	Avery Anderson	2023-03-13	-	201
11	110	Ethan Martin	2023-03-13	-	201

Home.tsx

PATH: /

If user logged in

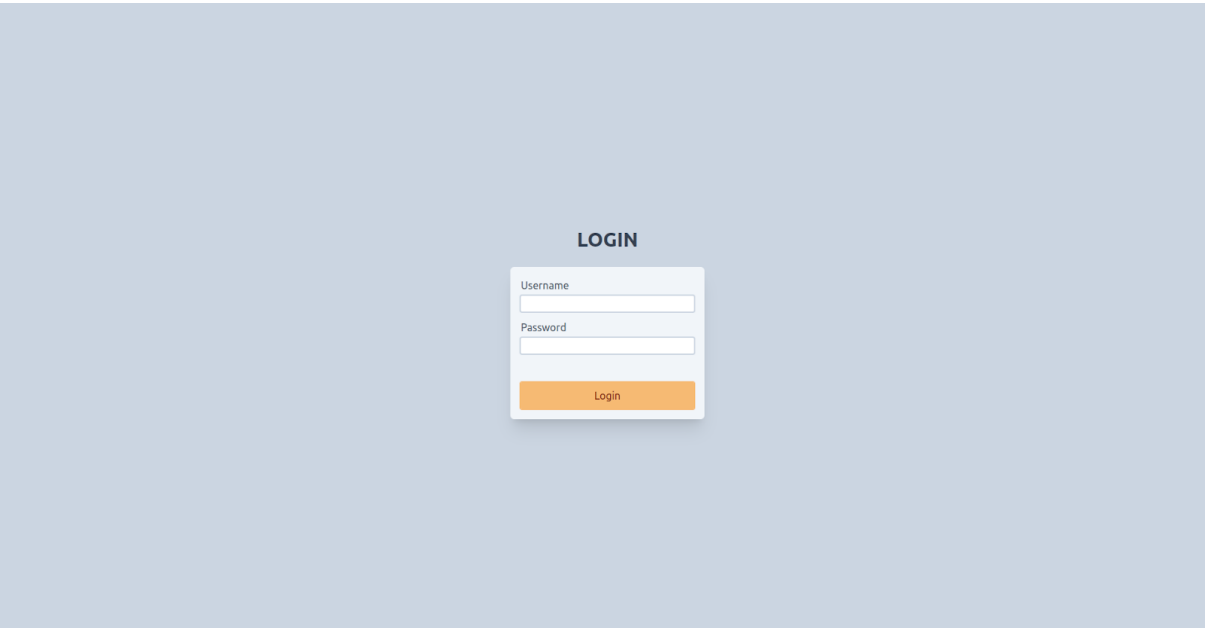
Redirect to the appropriate user page

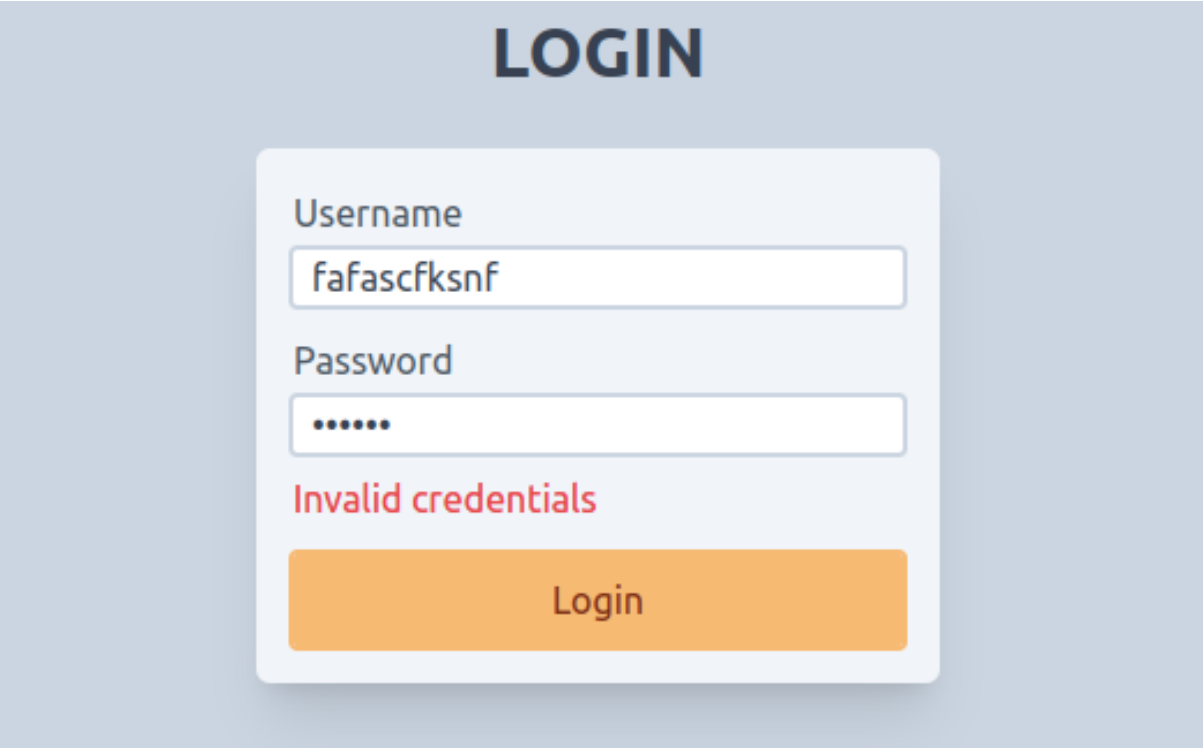
Else **redirect** to login page

Login.tsx

PATH: /login

User can login to their account
Every fetch call we do in the frontend contains a
Basic Authorization Header
For **enabling security**
Username and Password is checked in the **server side** before sending a response



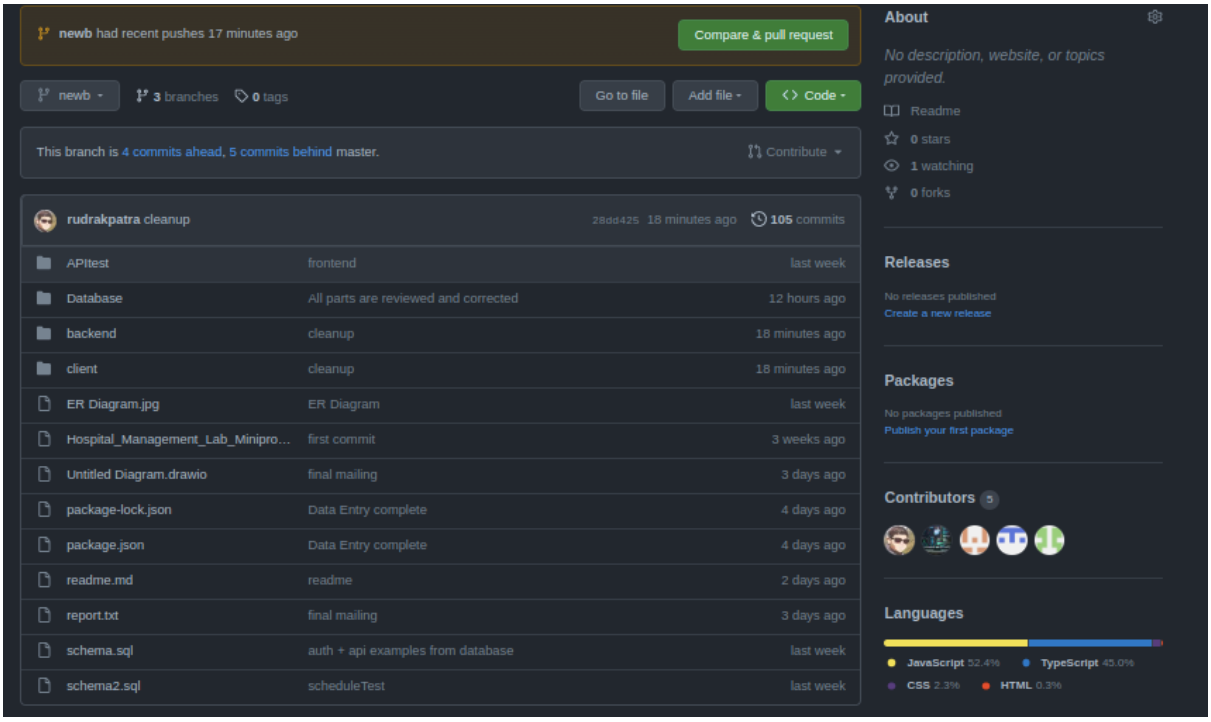


GITHUB SOURCE CODE

we have uploaded our source code on github (use the newb branch)

Instructions to run locally
You must have Git, Nodejs, NPM, Nodemon, MySQL installed in your machine

Next,
Clone git repository (use the newb branch)
git clone https://github.com/rudrakpatra/DBMS_MiniProject.git
Open mysql
Create database Hospital;
From Database run:
creation.sql
Insertion.sql
*(database is ready)
In backend:
npm i
npm run start
*(server starts running)
In client:
npm i
npm run dev
*(client opens in browser)



[Contributors]

[Code Frequency]