FULL STACK DEV

Installing / Configuring React

Presented by:

Rajeev Khoodeeram

OCTOBER 2025

REACT (REACT.JS OR REACTJS)

- It has 3 basic principles:
  - Nature: A JavaScript library for building user interfaces. It's often described as "just the V (view) in MVC," meaning it's highly focused on UI components.
  - Best For: Single-page applications (SPAs), complex UIs, large-scale applications
  - Component-Based: Encourages building UIs from small, isolated, reusable components.
- We will TypeScript to write our code (to maintain consistency with Angular and also it is currently an excellent skill required, for Full Stack development)

STEP 1: CREATE A NEW REACT PROJECT WITH VITE

- Open your terminal or command prompt ; create a directory called **ClinicApp**
- Create backend with JavaSpring boot using SpringInitializr, etc
- Navigate to the directory (ClinicApp) where you want to create your project.

- Run the Vite command to create a new React project
- >>**npm create vite@latest** frontend -- --template react-ts

- npm create vite@latest: Invokes the Vite project scaffolder.
  - frontend : The name of your new project directory.
  - -- --template react-ts: Specifies that you want a React project with TypeScript.

```
(base) rajeev@Rajeev-Khoodeeram git % npm create vite@latest session4-react -- -]
-template react-ts
Need to install the following packages:
create-vite@7.0.3
Ok to proceed? (y) y


> npx
> "create-vite" session4-react --template react-ts

|
◇  Scaffolding project in /Users/rajeev/git/session4-react...
|
└  Done. Now run:

  cd session4-react
  npm install
  npm run dev

(base) rajeev@Rajeev-Khoodeeram git % cd
```

INSTALL DEPENDENCIES

- Navigate into your new project directory and install dependencies:

- >>cd frontend
- >>npm install
- >>npm run dev

```
(base) rajeev@Rajeev-Khoodeeram session4-react % npm run dev

> session4-react@0.0.0 dev
> vite


  VITE v7.0.6  ready in 245 ms

  ➜  Local:   http://localhost:5173/
  ➜  Network: use --host to expose
  ➜  press h + enter to show help
```

- This will usually start the server on http://localhost:5173/ (Vite's default port). Open that URL in your browser, and you should see the default Vite + React welcome page.

RUNNING YOUR ANGULAR APP

- Once the folder is created, open it in VS Code.
- >> npm install (in VS Code inside the folder)

- Let us check if our angular front end is working by opening a terminal inside VS code :
  - >> **ng serve**

- Normally, it will be using : http://localhost:4200.

- Add this line in RestController to allow access to backend
  - @CrossOrigin(origins = "http://localhost:4200")

CROSS DOMAIN

- Remember to add **CORS** for each frontend to allow cross domain access to backend (as each serves on a different port)

```
//@Controller  // Use @Controller if you want to return views (HTML pages)
@RestController // Use @RestController if you want to return JSON responses
@RequestMapping("/api/students")
//@CrossOrigin(origins = "http://localhost:4200") // Allow CORS for frontend
//@CrossOrigin(origins = "http://localhost:5175") // Allow CORS for frontend
@CrossOrigin(origins = "http://localhost:5176") // Allow CORS for all origin
public class StudentRestController {
```

FULL STACK DEV

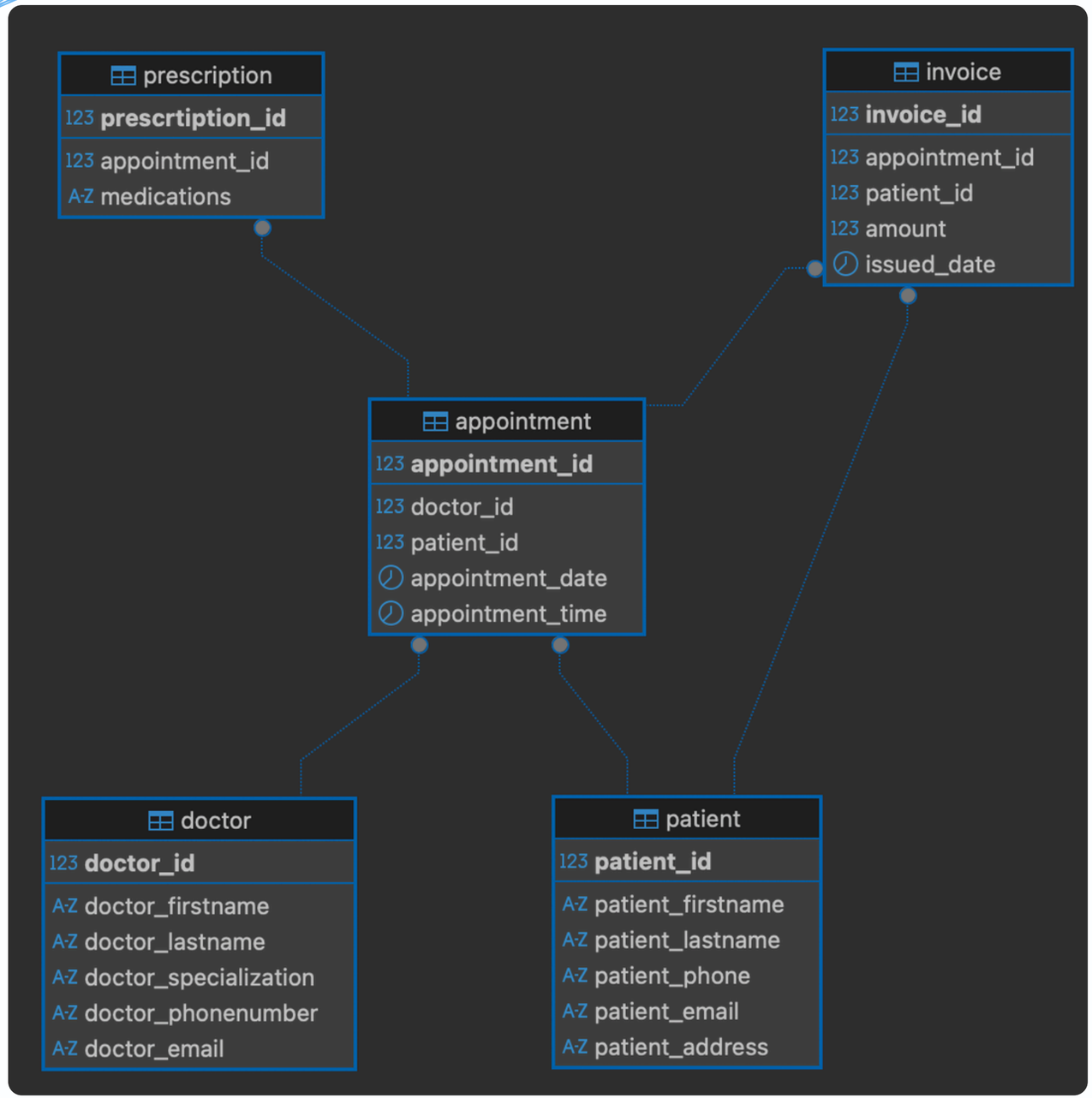
Our ClinicApp - Doctor Entity

Presented by:

Rajeev Khoodeeram

OCTOBER 2025

A BRIEF OF THE CLINICAPP



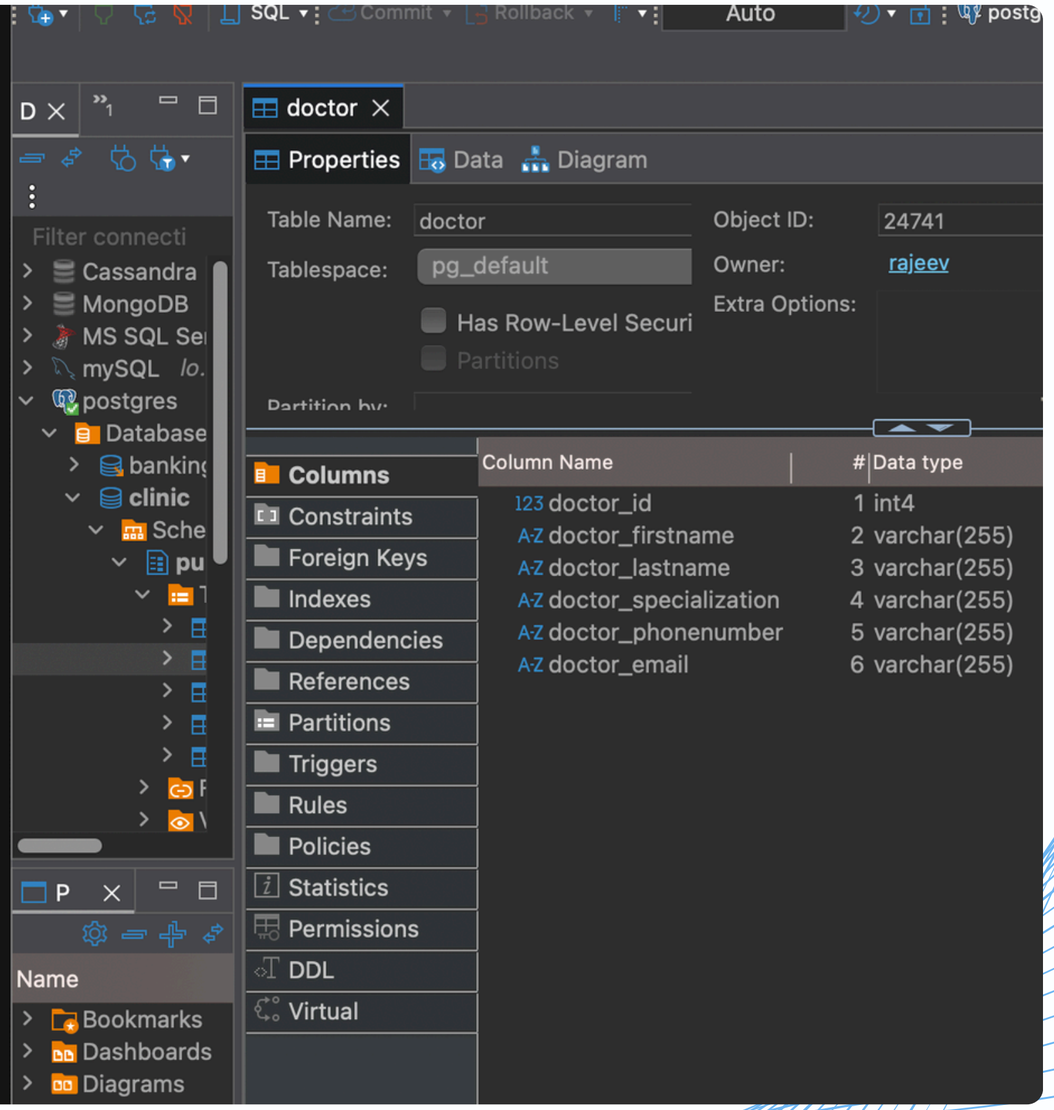
DOCTOR ENTITY

- We first test all the endpoints for the Doctor entity. Follow these steps :

- Create the following :
  - Doctor model
  - DoctorController
  - DoctorService
  - DoctorRepository
  - Doctor endpoint testing with postman

DOCTOR MODEL

```java
@Entity
@Table(name = "doctor")
public class Doctor {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "doctor_id")
    private int doctorId;

    @Column(name = "doctor_firstname")
    private String doctorFirstName;

    @Column(name = "doctor_lastname")
    private String doctorLastName;

    @Column(name = "doctor_specialization")
    private String doctorSpecialization;

    @Column(name = "doctor_phonenumber")
    private String doctorPhoneNumber;

    @Column(name = "doctor_email")
    private String doctorEmail;
```



DOCTORCONTROLLER

```java
@RestController
@RequestMapping("/api/doctors")
@CrossOrigin(origins = "http://localhost:5173")
public class DoctorController {
        private final DoctorService doctorService;

        public DoctorController(DoctorService doctorService) {
            this.doctorService = doctorService;
        }

        @GetMapping
        public ResponseEntity<List<Doctor>> getAllDoctors() {
            List<Doctor> doctors = doctorService.findAllDoctors();
            return new ResponseEntity<>(doctors, HttpStatus.OK);
        }

        @GetMapping("{id}")
        public Doctor getDoctorById(@PathVariable int id) {
            return doctorService.findDoctorById(id);
        }

        @PostMapping
        public Doctor createDoctor(@RequestBody Doctor doctor) {
            return doctorService.saveDoctor(doctor);
        }

        @PutMapping("{id}")
        public Doctor updateDoctor(@PathVariable int id, @RequestBody Doctor doctor) {
            return doctorService.updateDoctor(id, doctor);
        }
```

DOCTORSERVICE

```java
@Service
public class DoctorService {

    private final DoctorRepository doctorRepository;

    public DoctorService(DoctorRepository doctorRepository) {
        this.doctorRepository = doctorRepository;
    }

    public List<Doctor> findAllDoctors() {
        return doctorRepository.findAll();
    }

    public Doctor findDoctorById(int id) {
        return doctorRepository.findById(id).orElse(other:null);
    }

    public Doctor saveDoctor(Doctor doctor) {
        return doctorRepository.save(doctor);
    }

    public Doctor deleteDoctor(int id) {
        Doctor doctor = findDoctorById(id);
        if (doctor != null) {
            doctorRepository.deleteById(id);
        }
        return doctor;
    }
```

DOCTORREPOSITORY

```java
@Repository
public interface DoctorRepository extends JpaRepository<Doctor, Integer> {
    // Custom query methods (if needed) can be defined here
}
```

TESTING USING POSTMAN

- See demo.

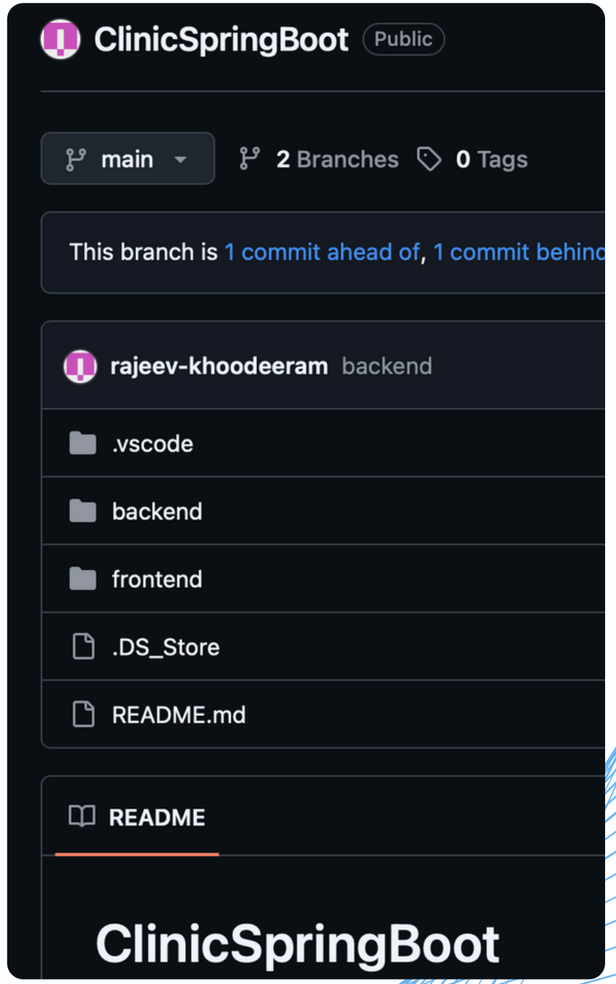
GITHUB

- echo "# ClinicApp" >> README.md
- git init
- git add README.md

- git commit -m "Committing Doctor endpoints"
- git branch -M main

- git remote add origin https://github.com/rajeev-khoodeeram/ClinicSpringBoot.git

- git push -u origin main



FULL STACK DEV

Creating your React frontend

Presented by:

Rajeev Khoodeeram

OCTOBER 2025

WRITING THE DOCTOR MODEL (TYPESCRIPT INTERFACE)

- Just like in Angular, it's good practice to define the shape of your data using TypeScript interfaces for type safety.

- Right click on src and create a new folder types (models in Angular) and add Doctor.ts (Use .ts for non-UI related logic).

```
export interface Doctor {
  doctorId: string;
  doctorFirstName: string;
  doctorLastName: string;
  doctorSpecialization: string;
  doctorPhoneNumber: string;
  doctorEmail: string;
}
```

API URL

- In React, if multiple services need the same API base URL, the clean approach is to store it in one place and import it everywhere.

- Use .env.development → for local dev.
- Use .env.production → for deployed app.
- Git should not commit secrets → add .env to .gitignore.

- We will use a **.env** file in the root folder (because we are using VITE !!) :

- VITE_API_URL=http://localhost:8080/api

SET UP AN API UTILITY

- React's equivalent of a "Service"

- In React, you often create simple utility files to encapsulate API calls.

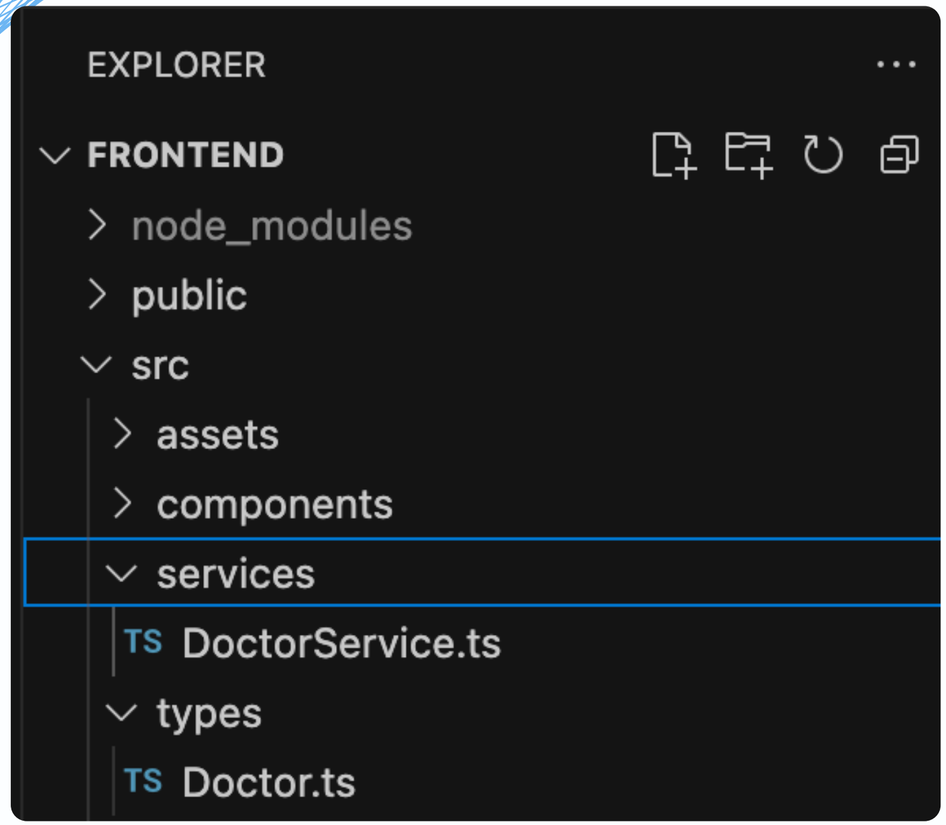
- IMPORTANT: Remember to adjust your Spring Boot **@CrossOrigin** annotation to include http://localhost:5173 (or whatever port Vite uses) if you don't have a global CORS configuration that handles all local development ports.

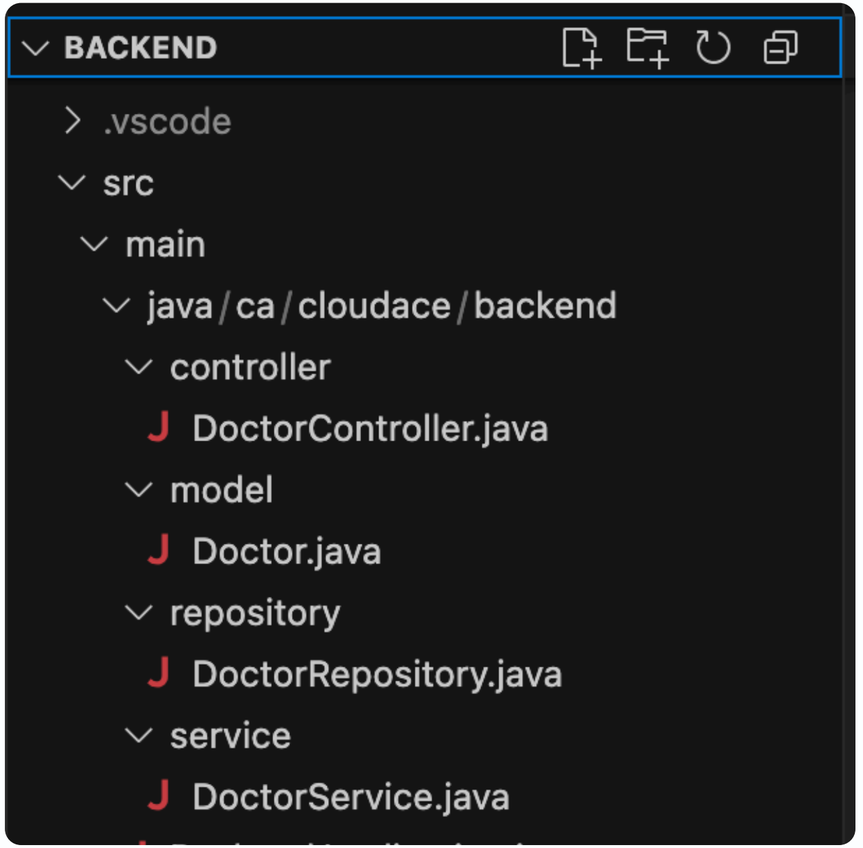
- Use .ts for non-UI related logic; so this file will be saved as ***DoctorService.ts*** in services folder

DOCTORSERVICE.TS

```
import { type Doctor } from "../types/Doctor";
const API_URL = import.meta.env.VITE_API_URL;

export const getDoctors = async (): Promise<Doctor[]>
=> {
  const response = await fetch(`${API_URL}/doctors`);
  console.log('API_URL:', API_URL);

  if (!response.ok) {
    throw new Error('Failed to fetch doctors');
  }
  return response.json();
};
```

WHERE ARE WE ?


EXPLORER
FRONTEND
node_modules
public
src
assets
components
services
DoctorService.ts
types
Doctor.ts


BACKEND
.vscode
src
main
java / ca / cloudace / backend
controller
DoctorController.java
model
Doctor.java
repository
DoctorRepository.java
service
DoctorService.java

FULL STACK DEV

spring boot

Listing doctors

Presented by:

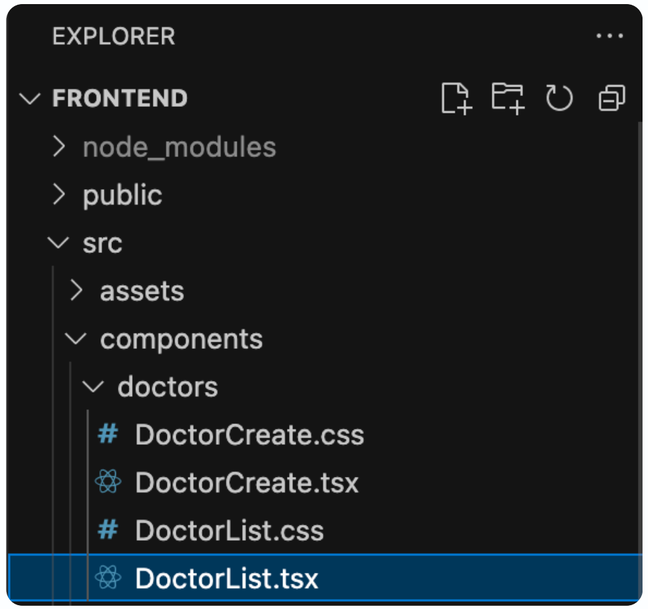
Rajeev Khoodeeram

OCTOBER 2025

CREATING THE DOCTORLIST COMPONENT

- Create components folder (and a subfolder for each entity as each will contain two files : a UI and a css)

- Purpose: Reusable UI building blocks (React components).
- Components import models for typing and services for data.

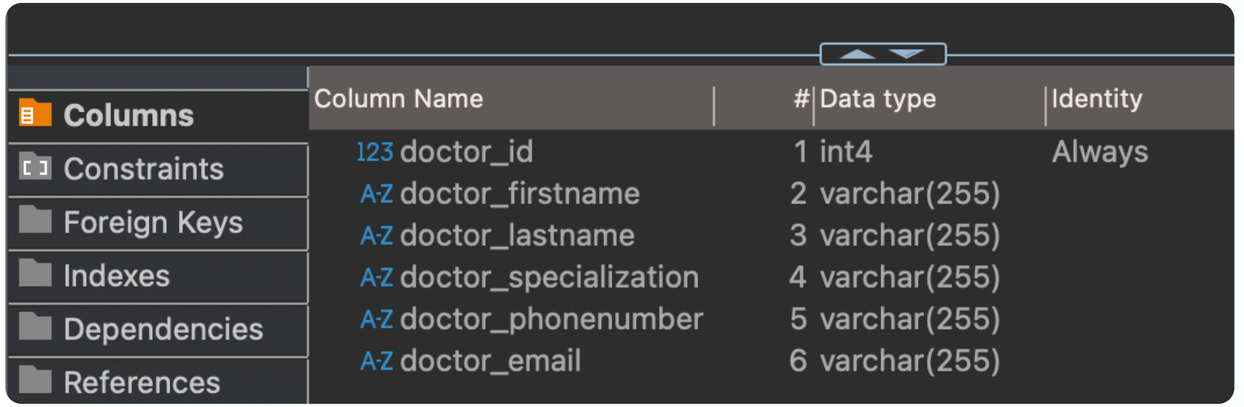


DOCTORLIST.TSX

```tsx
import {type  Doctor } from "../../types/Doctor";
import { useEffect, useState } from "react";
import { getDoctors } from "../../services/DoctorService";
import './DoctorList.css';

const DoctorList = () => {
  const [doctors, setDoctors] = useState<Doctor[]>([]);

  useEffect(() => {
    const fetchDoctors = async () => {
      try {
        const doctorsData = await getDoctors();
        setDoctors(doctorsData);
      } catch (error) {
        console.error('Error fetching doctors:', error);
      }
    };
```

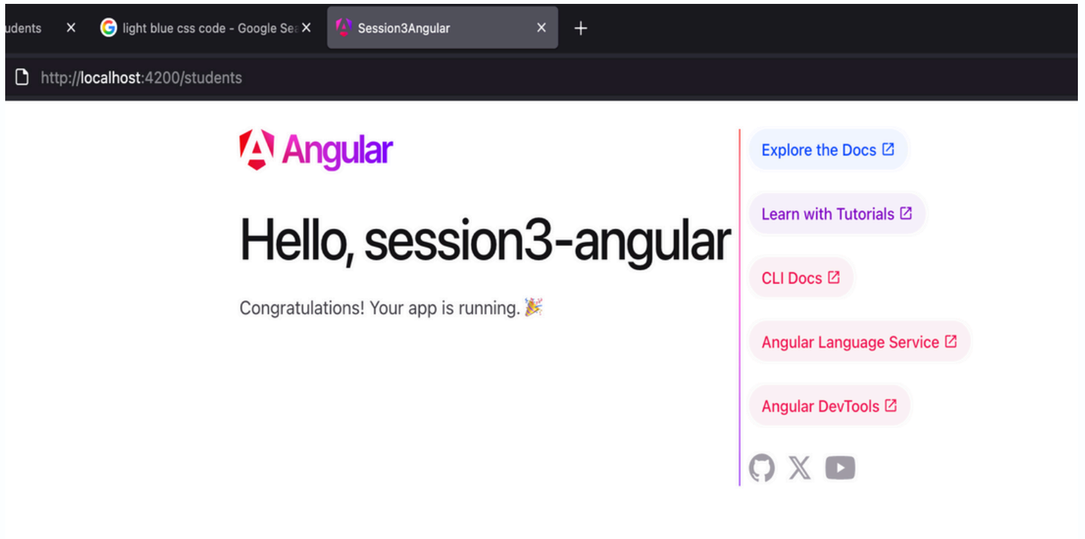
LISTING DOCTORS

```
  return (
    <div>
      <h2>Doctor List</h2>
      <table>
        <thead>
          <tr>
            <th>ID</th>
            <th>First Name</th>
            <th>Last Name</th>
            <th>Specialization</th>
            <th>Phone Number</th>
            <th>Email</th>
          </tr>
        </thead>
        <tbody>
          {doctors.map((doctor) => (
            <tr key={doctor.doctorId}>
              <td>{doctor.doctorId}</td>
              <td>{doctor.doctorFirstName}</td>
              <td>{doctor.doctorLastName}</td>
              <td>{doctor.doctorSpecialization}</td>
              <td>{doctor.doctorPhoneNumber}</td>
              <td>{doctor.doctorEmail}</td>
            </tr>
          ))}
        </tbody>
      </table>
    </div>
  );
};

export default DoctorList;
```



HTTP SPECIFIC : GETMAPPING

- With SSR:
  - User requests /home.
  - Angular runs on the server (Node.js) and generates HTML for /home.
  - Browser gets ready-to-render HTML instantly.
  - Angular JavaScript loads and "hydrates" (adds interactivity).
-

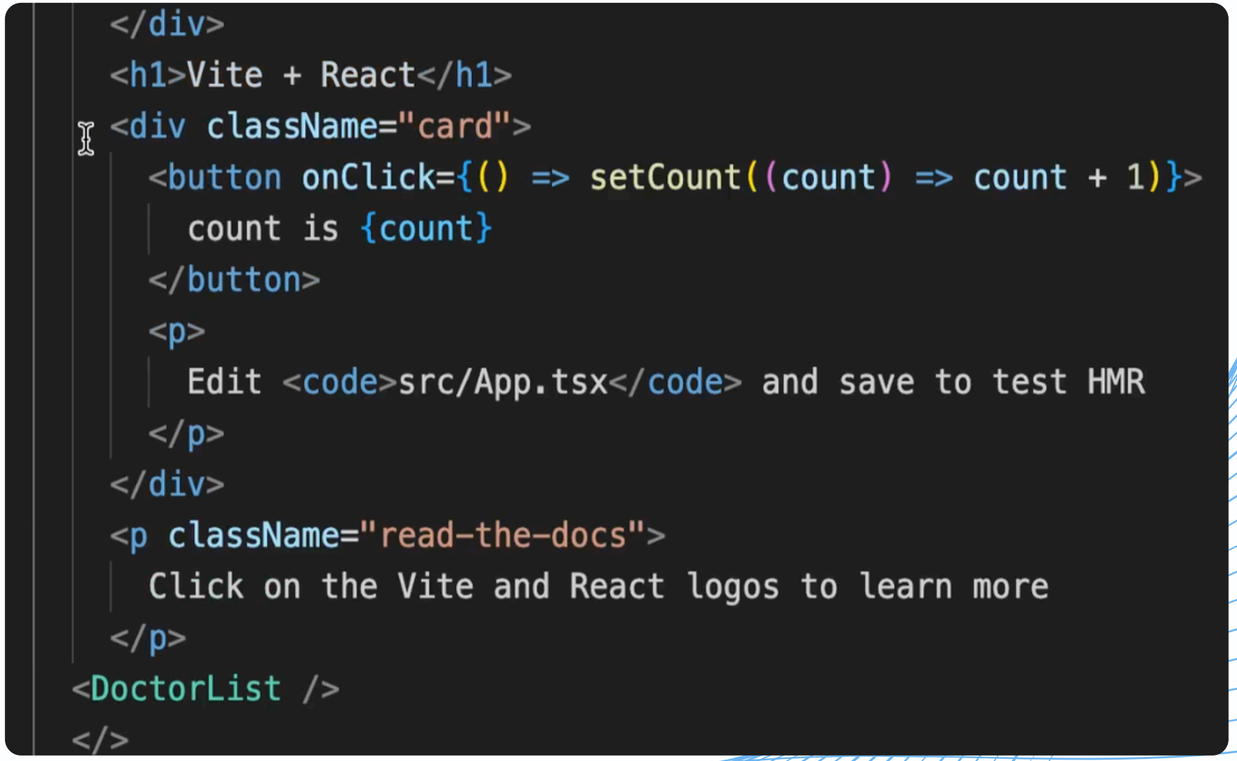


DON'T FORGET THE CSS FILE

```css
table {
  width: 100%;
  border-collapse: collapse;
}

th, td {
  padding: 12px 15px;
  border: 1px solid #4d0a0a;
}

th {
  background-color: #6e0808;
  text-align: left;
}

tr:hover {
  background-color: #502828;
}

.add-doctor {
  margin: 20px 0;
  text-align: left;
}
```

EDIT APP.TSX

```
<h1>Vite + React</h1>
 <DoctorList />
</>
)
}
export default App
```



FULL STACK DEV

ClinicApp : Adding a doctor

Presented by:

Rajeev Khoodeeram

OCTOBER 2025

ADDING A NEW DOCTOR

- Step 1:Add createDoctor function in DoctorService.ts
- Step 2: Create the DoctorCreate.tsx in folder components with DoctorCreate.css
- Step 3: Add a link that will allow users to navigate to the form for adding a new doctor
- Step 4: Add route in App.tsx
- Step 5: Test the application

CREATEDOCTOR

```
/**
 * Create a doctor
 * @param doctor
 * @return the created doctor in json format
 */
export const createDoctor = async (doctor: Doctor):
Promise<Doctor> => {
  const response = await fetch(`${API_URL}/doctors`, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify(doctor),
  });

  if (!response.ok) {
    throw new Error('Failed to create doctor');
  }
  return response.json();
};
```

CREATING THE FORM

- Things to remember :
  - We have 5 fields with Id being auto increment
  - Specify error and success messages

- Please note that each input field must include value and onChange:

<input type="text" id="doctorFirstName" value={doctorFirstName} onChange={(e) => setDoctorFirstName(e.target.value)} />

THE FORM

```
import React, { type FormEvent } from "react";
import { createDoctor } from "../../services/DoctorService";
import { useNavigate } from "react-router-dom";
import  "./DoctorCreate.css"
import doctor from '../../assets/doctor.png';


const DoctorForm : React.FC = () => {

    const [doctorId, setDoctorId] = React.useState<string>('');
    const [doctorFirstName, setDoctorFirstName] =
React.useState<string>('');
    const [doctorLastName, setDoctorLastName] =
React.useState<string>('');
    const [doctorSpecialization, setDoctorSpecialization] =
React.useState<string>('');
    const [doctorPhoneNumber, setDoctorPhoneNumber] =
React.useState<string>('');
    const [doctorEmail, setDoctorEmail] =
React.useState<string>('');
```

HOW TO HANDLE FORM SUBMISSION ?

```
const handleSubmit = (event: FormEvent<HTMLFormElement>) => {
    event.preventDefault();
    let doctor = null;
//check if all fields have values
if (
    doctorFirstName &&
    doctorLastName &&
    doctorSpecialization &&
    doctorPhoneNumber &&
    doctorEmail
) {
    // All fields are filled, proceed with form submission
    console.log("All fields are filled. Submitting form...");
    // Here you can add your form submission logic, e.g., API call

    doctor = {
        doctorId,
        doctorFirstName,
        doctorLastName,
        doctorSpecialization,
        doctorPhoneNumber,
        doctorEmail
    };
```

CALLING CREATEDOCTOR

```
createDoctor(doctor)
    .then((response) => {
        console.log("Doctor created successfully:", response);

        // Optionally, reset the form fields here
        setDoctorId(0);
        setDoctorFirstName('');
        setDoctorLastName('');
        setDoctorSpecialization('');
        setDoctorPhoneNumber('');
        setDoctorEmail('');

        // Navigate to the doctors list page
        navigate('/doctors');

    })
    .catch((error) => {
        console.error("Error creating doctor:", error);
    });
```

THE HTML FORM

```
return (
    <div className="add-doctor-form">
        <img src={doctorphoto} alt="Doctor" className="doctor-image"  width="20%" />
        <h2>{isEditMode ? 'Edit Doctor' : 'Create Doctor'}</h2>
        <form onSubmit={handleSubmit}>
            <div className="form-group">
                <label htmlFor="doctorFirstName">First Name</label>
                <input type="text" id="doctorFirstName"  value={doctorFirstName}
                onChange={(e) => setDoctorFirstName(e.target.value)} />
            </div>
        <div className="form-group">
            <label htmlFor="doctorLastName">Last Name</label>
            <input type="text" id="doctorLastName" value={doctorLastName}
            onChange={(e) => setDoctorLastName(e.target.value)} />
        </div>
```

FINAL STEPS

Step 3: Add a link on DoctorList that will allow users to navigate to the form for adding a new doctor

```
<div className="add-doctor">
      <a href="/doctors/create">Create New Doctor</a>
</div>
```

Step 4: Add route in App.tsx

```
<Route path="/doctors/create" element={<DoctorForm />} />
```

Let us add an image

Download image
Place in assets

Modify DoctorCreate.tsx

```
import doctor from '../../assets/doctor.png';
<img src={doctor} alt="Doctor" className="doctor-image"
width={"20%"} />
```

FULL STACK DEV

Editing a doctor

Presented by:

Rajeev Khoodeeram

OCTOBER 2025

WHERE TO CREATE THE ANGULAR FRONTEND ?

Step 1 : Add link on the list as Action (th also !)

```
<button onClick={() => navigate(`/doctors/edit/$
{doctor.doctorId}`)}>Edit</button>
```

Step 2 : Add route in App.tsx

```
<Route path="/doctors/edit/:id" element={<DoctorForm />} />
```

```
    <th>Actions</th>
  </tr>
</thead>
<tbody>
  {doctors.map((doctor) => (
    <tr key={doctor.doctorId}>
      <td>{doctor.doctorId}</td>
      <td>{doctor.doctorFirstName}</td>
      <td>{doctor.doctorLastName}</td>
      <td>{doctor.doctorSpecialization}</td>
      <td>{doctor.doctorPhoneNumber}</td>
      <td>{doctor.doctorEmail}</td>
      <td>
        <button onClick={() => navigate(`/doctors/edit/${doctor.doctorId}`)}>Edit</button>
```

COMMAND LINE

Step 3 : Modify DoctorService.ts to update a doctor

```
/**
 * Update a doctor
 * @param doctorId
 * @param updatedData
 * @returns
 */
export const updateDoctor = async (doctorId: number,
updatedData: Partial<Doctor>): Promise<Doctor> => {
  const response = await fetch(`${API_URL}/doctors/$
{doctorId}`, {
    method: 'PUT',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify(updatedData),
  });

  if (!response.ok) {
    throw new Error('Failed to update doctor');
  }
  return response.json();
};
```

TYPES OF ANGULAR APP

Step 4 : Modify DoctorCreate.tsx

Add outside handleSubmit

```
 const { id } = useParams();

 const API_URL = import.meta.env.VITE_API_URL + '/doctors';
    const isEdit = Boolean(id);

    // Fetch user data if editing
  useEffect(() => {
    if (id) {

      fetch(`${API_URL}/${id}`)
        .then((res) => res.json())
        .then((data) => {
          setDoctorId(data.doctorId);
          setDoctorFirstName(data.doctorFirstName);
          setDoctorLastName(data.doctorLastName);

setDoctorSpecialization(data.doctorSpecialization);
          setDoctorPhoneNumber(data.doctorPhoneNumber);
          setDoctorEmail(data.doctorEmail);
        });
    }
  }, [id]);
```

HTTP SPECIFIC : GETMAPPING

Add inside handleSubmit (when update button is clicked)

```
if (id) {
// Call the updateDoctor function with the doctor ID and updated
data
        updateDoctor(Number(id), doctor)
         .then((response) => {
          console.log("Doctor updated successfully:", response);
                        })
         .catch((error) => {
          console.error("Error updating doctor:", error);
                        });
```

RUNNING YOUR ANGULAR APP

In the form, change the following :

```
<h2>{isEdit ? "Edit Doctor" : "Create Doctor"}</h2>
```

```
<button type="submit">{isEdit ? "Update Doctor" : "Create Doctor"}</button>
```

TEST !!

FULL STACK DEV

Deleting a doctor

Presented by:

Rajeev Khoodeeram

OCTOBER 2025

MODIFY THE LIST

Step 1 : Add link on the list as Action

```
<button onClick={() => handleDelete(doctor.doctorId)}>Delete</
button>
```

This button will invoke a handleDelete inside the DoctorList component and as such there is no need to add a route !!

```
<td>
  <button onClick={() => navigate(`/doctors/edit/${doctor.doctorId}`)}>Edit</button>
  <button onClick={() => handleDelete(doctor.doctorId)}>Delete</button>
</td>
```

UPDATE DOCTORSERVICE

Step 2 : Modify DoctorService.ts

```
/**
 * Delete a doctor
 * @param doctorId
 * @returns
 */
export const deleteDoctor = async (doctorId: number):
Promise<Doctor | null> => {

  const confirmed = confirm("Are you sure you want to
delete this doctor?");
  if (!confirmed) {
    return null;
  }

  const response = await fetch(`${API_URL}/doctors/$
{doctorId}`, {
    method: 'DELETE',
  });

  if (!response.ok) {
    throw new Error('Failed to delete doctor');
  }

  const data = await response.json(); // I have modified
the backend delete to return a json object
  return data;
};
```

HANDLE THE DELETE ACTION

Step 3 : Modify DoctorList.tsx

```
const handleDelete = async (doctorId: number) => {
    try {
      const code = await deleteDoctor(doctorId);
      if (code !== null) {
       setDoctors((prevDoctors) =>
       prevDoctors.filter((doctor) => doctor.doctorId !==
doctorId)
        );
      }
    } catch (error) {
      console.error('Error deleting doctor:', error);
    }
  };
```