Understanding the React UI

Before building our backend with Spring Boot, let's understand how we'll test it using a React frontend application. Initially, we'll focus on fetching all job posts.

You can download this **React UI** project from: ***React UI***

👉 Setup Steps

Step 1: Install JSON Server (Fake Backend)

```
npm install -g json-server
```

- This gives us a quick way to create a dummy backend without writing any server code.

Step 2: Run the JSON Server

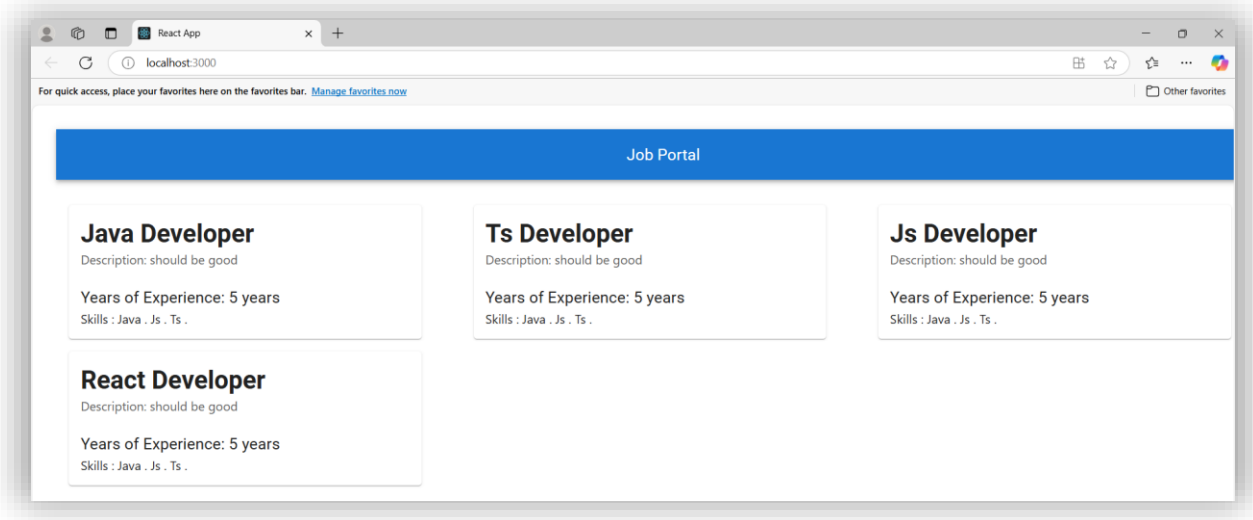
```
json-server --watch db.json --port 8000
```

- This starts our fake backend using the data in db.json. Our dummy job posts are now available at:

  http://localhost:8000/posts

Step 3: Run the React App

```
npm install    # Installs all required packages
npm start      # Starts the React application
```



👉 Understanding the React Code Structure

➢ React is a library that helps us build user interfaces with reusable components. Let's break down how our app works:

1. The HTML Entry Point (index.html)

In the public folder, we have index.html - this is the actual page displayed in the browser:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />

    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />

    <title>React App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>

  </body>
</html>
```

➢ Notice how the body only contains an empty . This is our mounting point where React will inject all our components.

2. The JavaScript Entry Point (index.js)

The src/index.js file connects React to our HTML:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
    <App />
);
```

- This code finds the "root" div in our HTML and renders our App component inside it.

3. The Main App Component (App.js)

App.js contains our main application component:

```
import './App.css';
import Search from './components/Search';

function App() {
  return (
    <div>
      <Search/>
    </div>
  );
}

export default App;
```

- It simply includes our Search component, which handles displaying the job posts.

4. The Search Component (AllPosts.jsx)

This component does the heavy lifting:

```jsx
import React from 'react'
import {
    AppBar,
    Toolbar,
    Box,
    Card,
    Grid,
    Typography,
  } from "@mui/material";
  import axios from "axios";
  import { useEffect, useState } from "react";

const Search = () => {
    const [post, setPost] = useState(null);

    useEffect(() => {
        const fetchInitialPosts = async () => {
            const response = await axios.get(`http://localhost:8000/posts`);
            console.log(response);
            setPost(response.data);
        }
         fetchInitialPosts();
      }, []);

  return (
        <Grid container spacing={2} sx={{ margin: "2%" }}>
                  <Box sx={{ flexGrow: 1 }}>
      <AppBar position="static">
        <Toolbar>

          <Typography variant="h6" align='center' component="div" sx={{ flexGrow: 1 }}>
            Job Portal
          </Typography>
        </Toolbar>
      </AppBar>
    </Box>
      <Grid item xs={12} sx={12} md={12} lg={12}>
      </Grid>
      {post &&
        post.map((p) => {
          return (
            <Grid key={p.id} item xs={12} md={6} lg={4}>
              <Card sx={{ padding: "3%", overflow: "hidden", width: "84%" }}>
                <Typography
                  variant="h5"
                  sx={{ fontSize: "2rem", fontWeight: "600" }}
                >
               {p.postProfile}
                </Typography>
                <Typography sx={{ color: "#585858", marginTop:"2%" }} variant="body" >
                  Description: {p.postDesc}
                </Typography>
                <br />
                <br />
                <Typography variant="h6">
                  Years of Experience: {p.reqExperience} years
                </Typography>

                <Typography gutterBottom  variant="body">Skills : </Typography>
                {p.postTechStack.map((s, i) => {
                  return (
                    <Typography variant="body" gutterBottom key={i}>
                      {s} .
                      {` `}
                    </Typography>
                  );
                })}

              </Card>
            </Grid>
          );
        })}
    </Grid>
  )
}

export default Search
```

👉 Key Parts of the Search Component

1. Fetching Data from the Backend

```
const [post, setPost] = useState(null);

useEffect(() => {
    const fetchInitialPosts = async () => {
        const response = await axios.get(`http://localhost:8000/posts`);
        console.log(response);
        setPost(response.data);
    }
    fetchInitialPosts();
}, []);
```

- useState(null) creates a state variable called post to store our job data
- useEffect(() => {...}, []) runs once when the component loads
- Inside we use axios.get() to fetch data from our fake backend at http://localhost:8000/posts
- When data arrives, we store it in the post state with setPost(response.data)

2. Displaying Job Cards Dynamically

```
{post &&
  post.map((p) => {
    return (
      <Grid key={p.id} item xs={12} md={6} lg={4}>
        <Card sx={{ padding: "3%", overflow: "hidden", width: "84%" }}>
          {/* Card content showing job details */}
        </Card>
      </Grid>
    );
})}
```

- post && post.map() first checks if we have post data, then maps through each item
- For each job post object in our array, we create a new Card component
- This creates multiple cards dynamically - one for each job in our data
- If we have 10 job posts in our JSON data, we'll get 10 cards displayed

3. Displaying Skills Dynamically

```
{p.postTechStack.map((s, i) => {
  return (
    <Typography variant="body" gutterBottom key={i}>
      {s} .{` `}
    </Typography>
  );
})}
```

- This is a nested map function that loops through the skills array (postTechStack) for each job
- For each skill in the array, it creates a Typography component showing the skill name

👉 The Flow of Data

- When the app starts, it contacts our JSON Server at http://localhost:8000/posts
- The server returns an array of job post objects
- React stores these objects in the post state variable
- For each item in this array, React renders a job card on the screen
- For each skill in a job's skill array, React renders a skill label

Eventually, we'll replace the JSON Server with our Spring Boot backend, but the React code will work the same way - it just needs to get an array of job posts in the same format.