Arpit Singh

19BCG10069

Kaiburr – Task 4

Technical Task

(Placement)

GitHub Link: https://github.com/TSM-ArpitSG/Kaiburr/tree/main/Kaibur_Tasks/Task_4

Task 4:

WEB UI Forms:

Create a basic WEB UI frontend for an application that you created for ***Task #1*** or #2 using any UI framework of your choice. You should be able to create, show and delete records from your UI.

Steps:

1. **Choose a UI framework**:
    a. You can choose any ***UI framework*** of your choice like React, Angular, Vue.js, etc. For this task, we will use ***ReactJS***.

2. **Set up a new React project**:
    a. ***Installing Node.js:***
        i. Go to the official ***Node.js website at https://nodejs.org/***

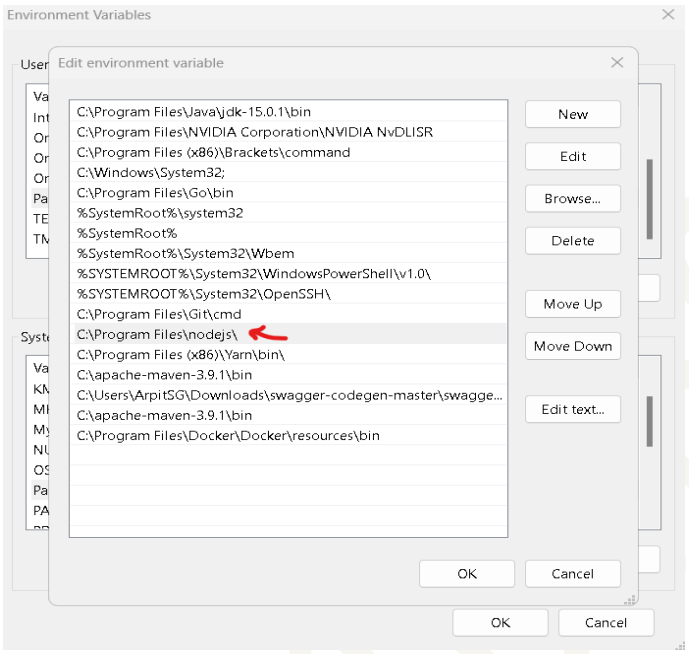
        ii. Click on the "***Downloads***" button on the home page.

        iii. Select the '***version of Node.js'*** that you want to download. If you're not sure which version to download, choose the latest version.

        iv. Choose the appropriate installer based on your '***operating system'*** (e.g., ***Windows***, macOS, Linux, etc.).

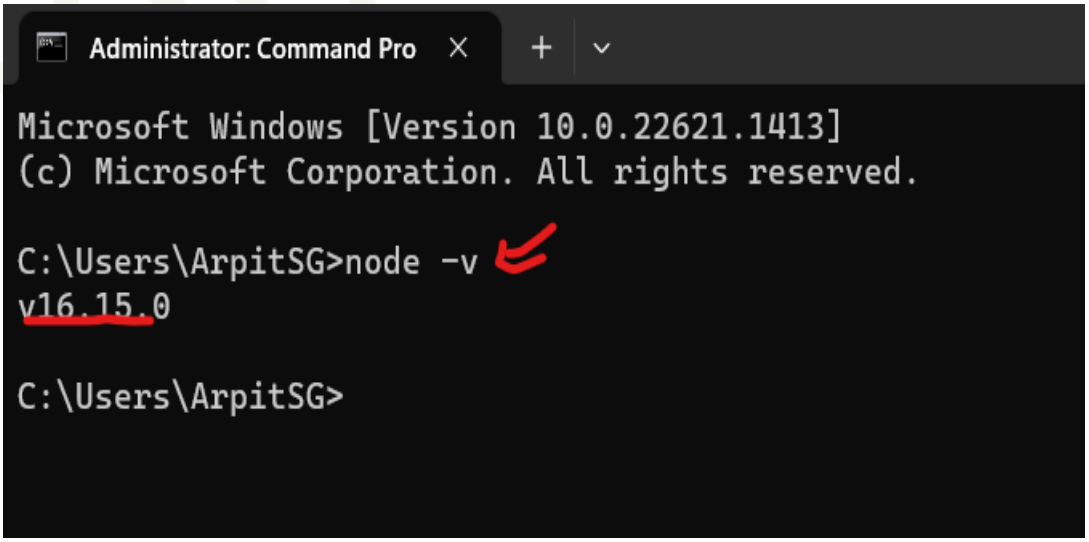
        v. ***"Run the installer"*** and follow the installation wizard's prompts to complete the installation.

vi. Setup the ***System (Environment) variable*** as per the desired directory.
For Example: "***C:\Program Files\nodejs***".



vii. Once the installation is complete, open a terminal or command prompt and ***type "node -v" to verify that Node.js is installed*** correctly. You should see the version number of Node.js printed in the terminal.

❖ Successful Installation can be checked as follows:



b. ***Creating a new REACT APP:***

i. ***'Navigate'*** to the Directory where you want to create the React Application. Ex: "***E:\Kaibur_Tasks\Task_4***"

ii. Run the following ***command to install create-react-app:***

```
npm install -g create-react-app.
```

iii. Run the following ***command to create a new React project:***

```
npx create-react-app my-app
```

iv. This will create a new React project in a folder named "***my-app***".

v. ***Open*** this React Project with an IDE of your choice (***IntelliJ***).

c. ***Install necessary packages***:

i. You will need to "***install some additional packages***" to use ***React with your REST API***. Open the terminal and ***navigate to the project directory*** using the following command:

```
cd E:\Kaibur_Tasks\Task_4\my-app
```

ii. Run the following command to install the required packages:

```
npm install axios react-router-dom
```

iii. The "***axios***" package is used to make ***HTTP requests to your REST API***, and the "***react-router-dom***" package is used for ***client-side routing.***

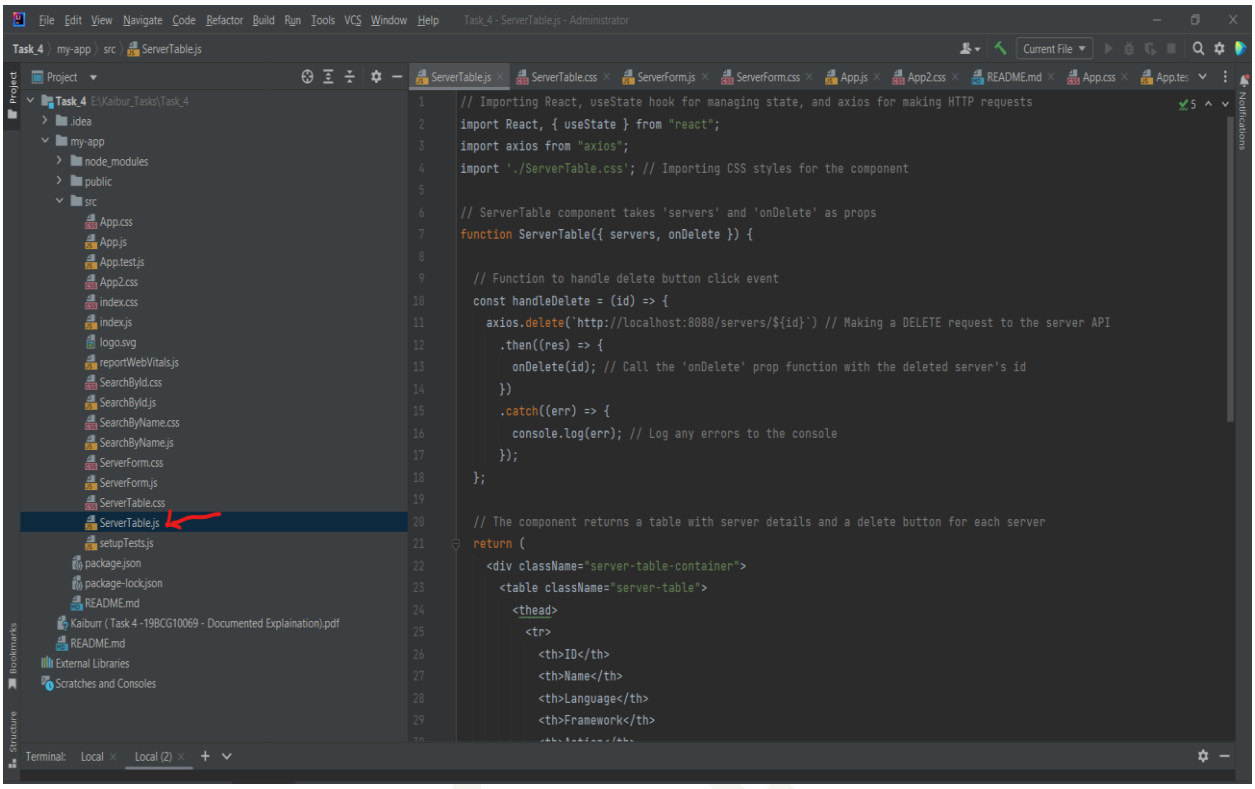
iv. Okay, after this we are ready to make our "***React UI Components***"

3. **Create UI Components**:

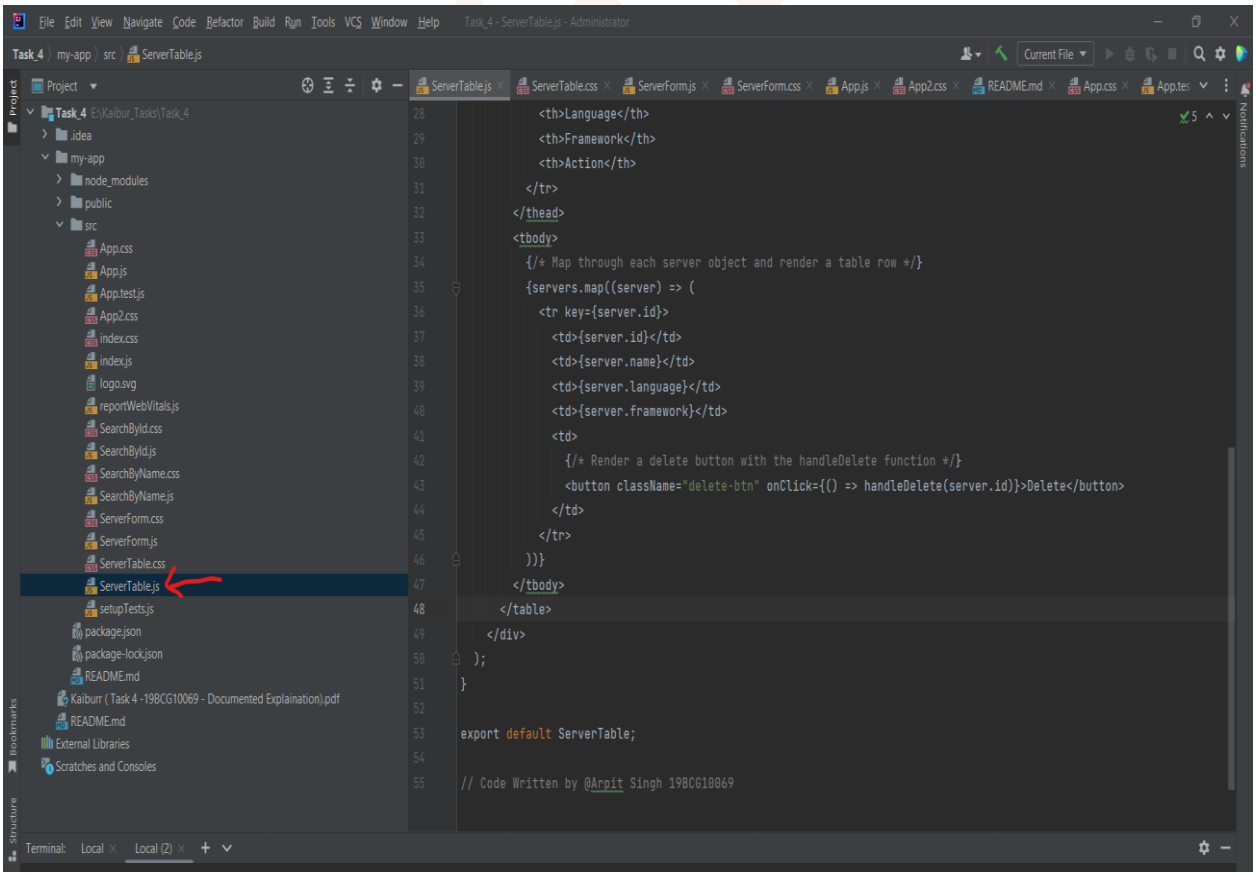
- **Note: All the Files discussed in this step will be created in the "*src*" directory of "*my-app*" React Project. Ex: "*E:\Kaibur_Tasks\Task_4\my-app\src*".**
- **Proper *Logical comments* have been added in each code file for better understanding.**

a. ***ServerTable.js(logic)***:

i. Create a File named '***ServerTable.js***' with the ***following code***:



```
// Importing React, useState hook for managing state, and axios for making HTTP requests
import React, { useState } from "react";
import axios from "axios";
import './ServerTable.css'; // Importing CSS styles for the component

// ServerTable component takes 'servers' and 'onDelete' as props
function ServerTable({ servers, onDelete }) {

  // Function to handle delete button click event
  const handleDelete = (id) => {
    axios.delete(`http://localhost:8080/servers/${id}`) // Making a DELETE request to the server API
      .then((res) => {
        onDelete(id); // Call the 'onDelete' prop function with the deleted server's id
      })
      .catch((err) => {
        console.log(err); // Log any errors to the console
      });
  };

  // The component returns a table with server details and a delete button for each server
  return (
    <div className="server-table-container">
      <table className="server-table">
        <thead>
          <tr>
            <th>ID</th>
            <th>Name</th>
            <th>Language</th>
            <th>Framework</th>
```

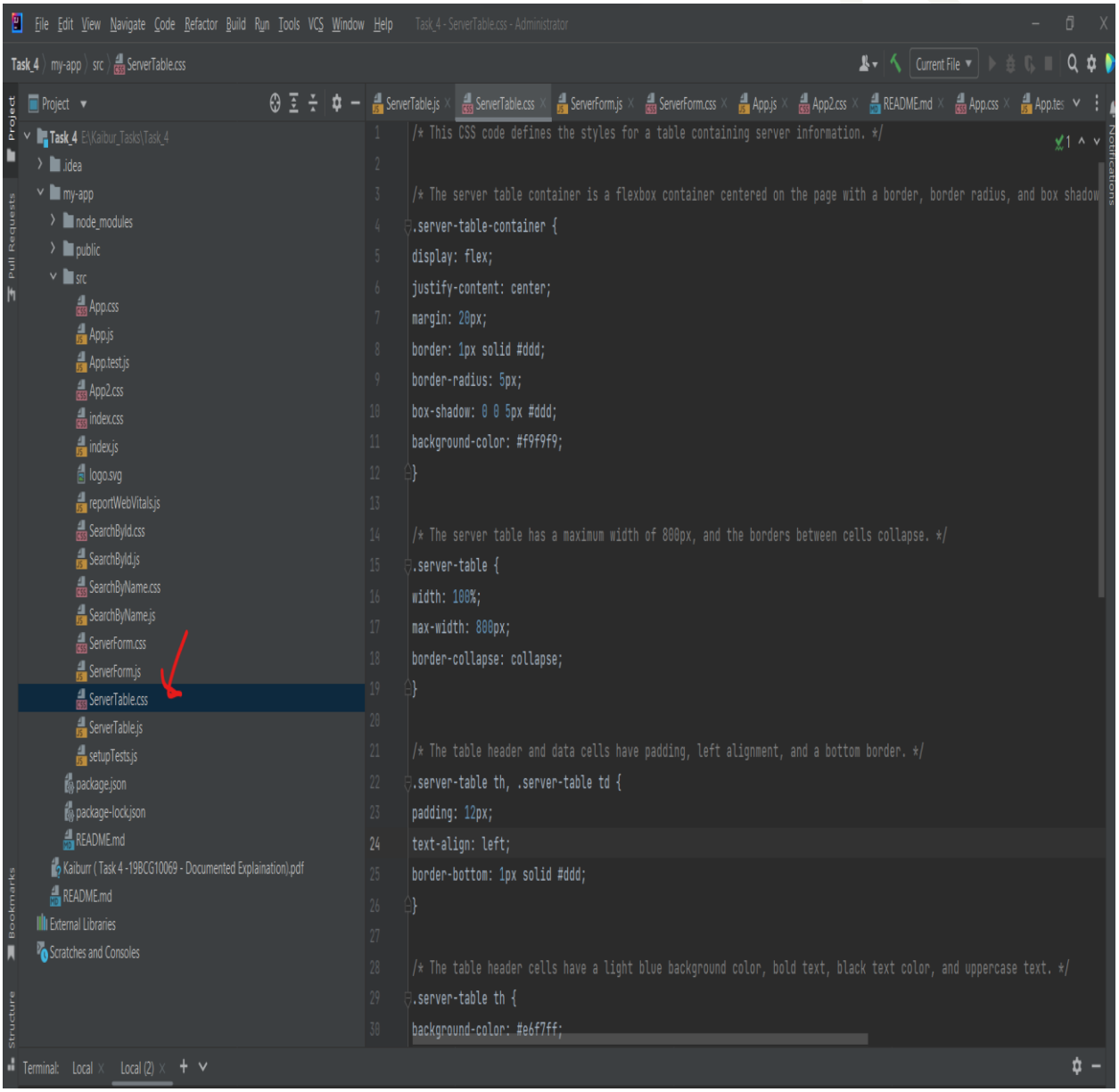


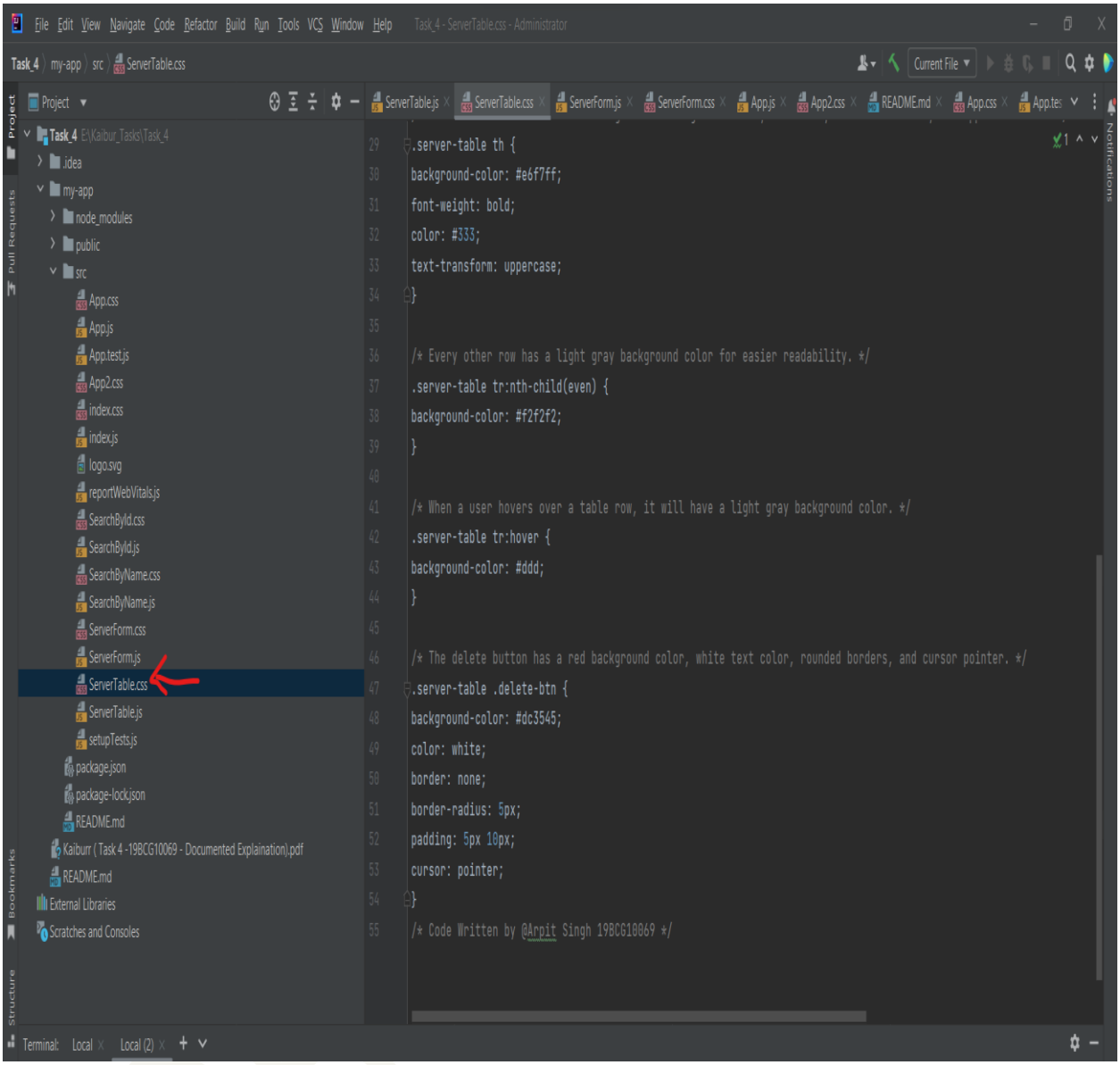
```
            <th>Language</th>
            <th>Framework</th>
            <th>Action</th>
          </tr>
        </thead>
        <tbody>
          {/* Map through each server object and render a table row */}
          {servers.map((server) => (
            <tr key={server.id}>
              <td>{server.id}</td>
              <td>{server.name}</td>
              <td>{server.language}</td>
              <td>{server.framework}</td>
              <td>
                {/* Render a delete button with the handleDelete function */}
                <button className="delete-btn" onClick={() => handleDelete(server.id)}>Delete</button>
              </td>
            </tr>
          ))}
        </tbody>
      </table>
    </div>
  );
}

export default ServerTable;

// Code Written by @Arpit Singh 19BCG10069
```

- In summary, **this component takes in an *array of servers as props* and renders a table with server details and a *delete button* for each server. When the delete button is clicked, *a DELETE request is sent to the server API* to delete the corresponding server, and the '*onDelete*' prop function is called to *update the list of servers*.**

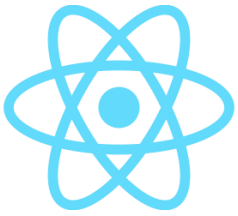
b. ***ServerTable.css(style)***:

i. Create a File named '***ServerTable.css***' with the ***following code***:



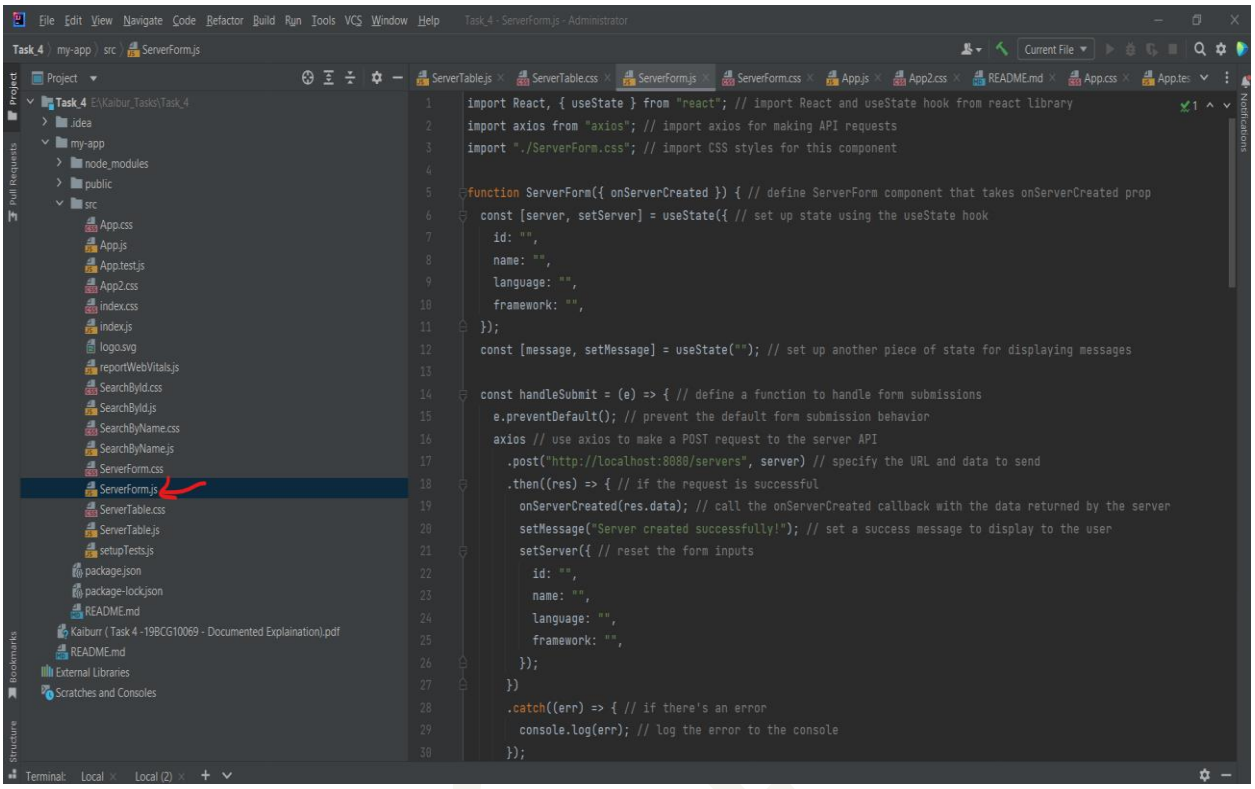


- In summary, **This CSS code provides styling for a server table component. It defines the *layout and appearance* of the table and its rows, including colors, fonts, and borders. It also adds *a hover effect to the rows and a delete* button with a red background color. The container for the table has a *flex display, is centred, and has a box shadow*.**

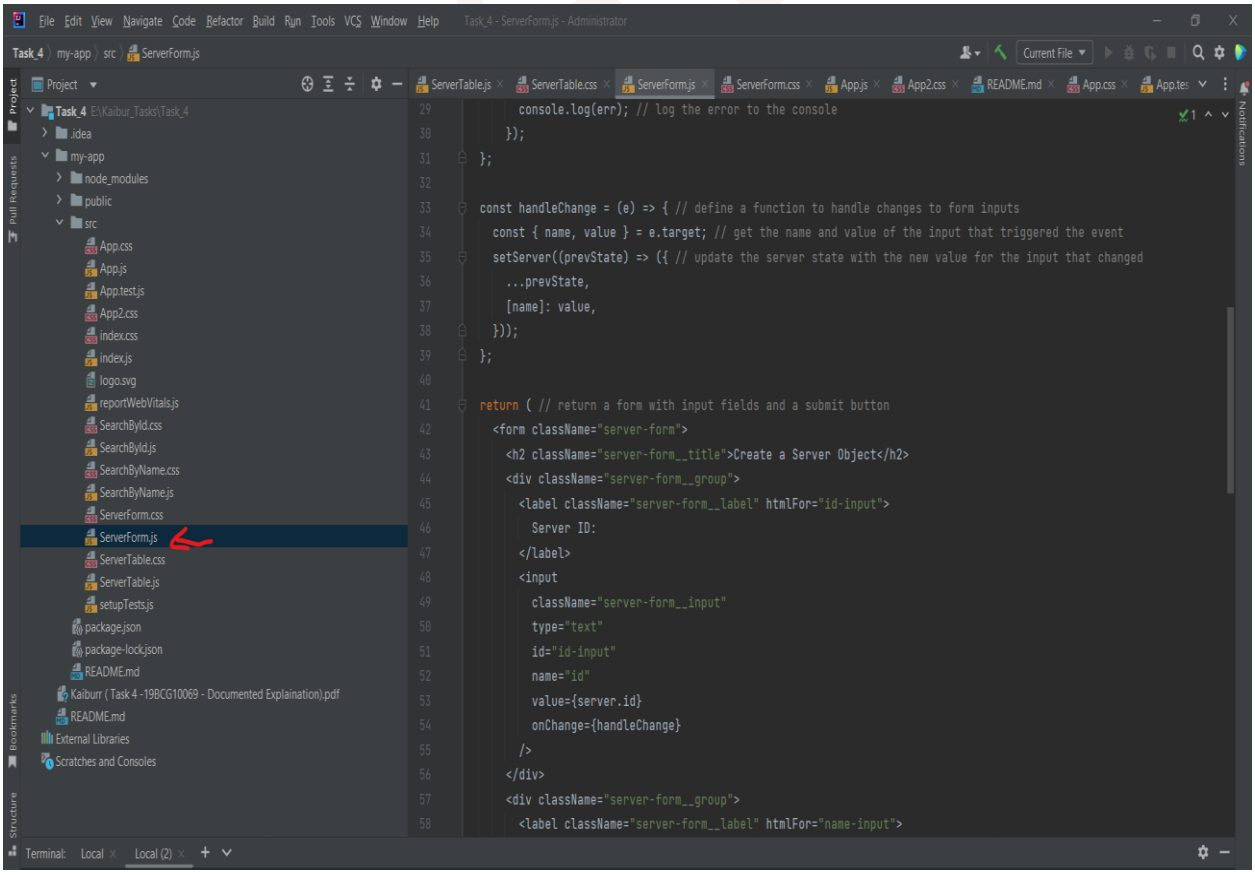


c. ***ServerForm.js(logic)***:

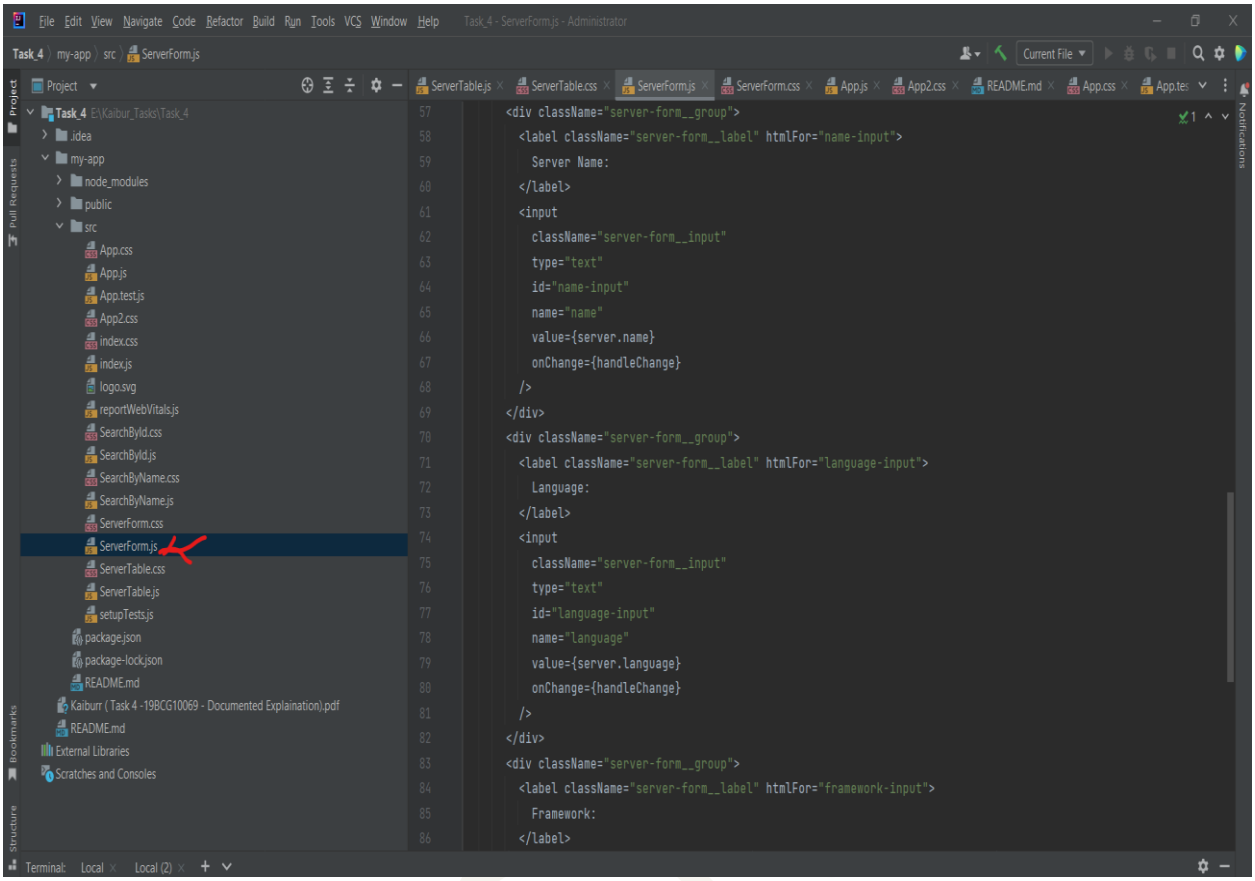
i. Create a File named '***ServerFrom.js***' with the ***following code***:



```
import React, { useState } from "react"; // import React and useState hook from react library
import axios from "axios"; // import axios for making API requests
import "./ServerForm.css"; // import CSS styles for this component

function ServerForm({ onServerCreated }) { // define ServerForm component that takes onServerCreated prop
  const [server, setServer] = useState({ // set up state using the useState hook
    id: "",
    name: "",
    language: "",
    framework: "",
  });
  const [message, setMessage] = useState(""); // set up another piece of state for displaying messages

  const handleSubmit = (e) => { // define a function to handle form submissions
    e.preventDefault(); // prevent the default form submission behavior
    axios // use axios to make a POST request to the server API
      .post("http://localhost:8080/servers", server) // specify the URL and data to send
      .then((res) => { // if the request is successful
        onServerCreated(res.data); // call the onServerCreated callback with the data returned by the server
        setMessage("Server created successfully!"); // set a success message to display to the user
        setServer({ // reset the form inputs
          id: "",
          name: "",
          language: "",
          framework: "",
        });
      })
      .catch((err) => { // if there's an error
        console.log(err); // log the error to the console
      });
```



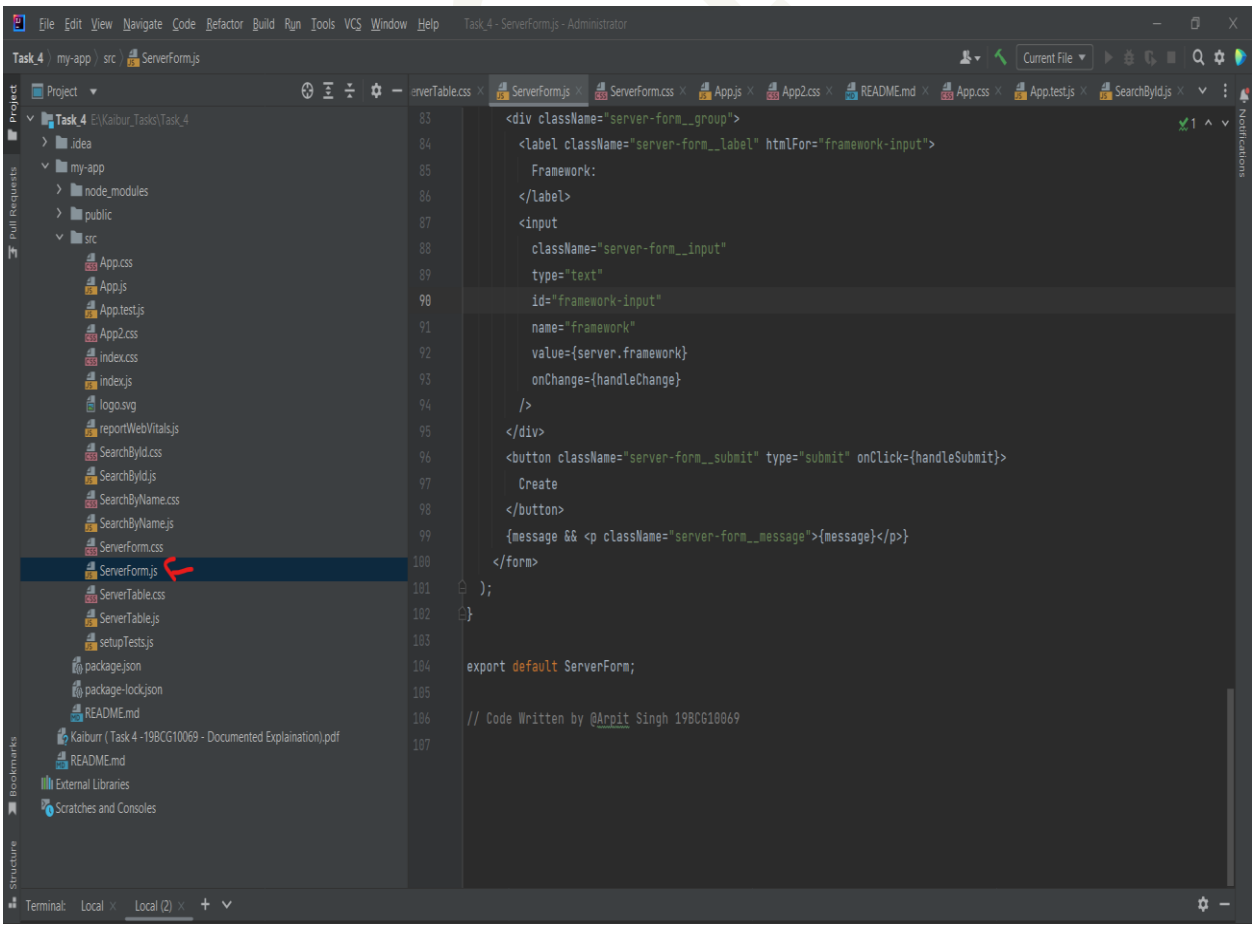
```
        console.log(err); // log the error to the console
      });
  };

  const handleChange = (e) => { // define a function to handle changes to form inputs
    const { name, value } = e.target; // get the name and value of the input that triggered the event
    setServer((prevState) => ({ // update the server state with the new value for the input that changed
      ...prevState,
      [name]: value,
    }));
  };

  return ( // return a form with input fields and a submit button
    <form className="server-form">
      <h2 className="server-form__title">Create a Server Object</h2>
      <div className="server-form__group">
        <label className="server-form__label" htmlFor="id-input">
          Server ID:
        </label>
        <input
          className="server-form__input"
          type="text"
          id="id-input"
          name="id"
          value={server.id}
          onChange={handleChange}
        />
      </div>
      <div className="server-form__group">
        <label className="server-form__label" htmlFor="name-input">
```


File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
Task_4 - ServerForm.js - Administrator
Task_4 > my-app > src > ServerForm.js
Current File
Project
Task 4 E:\Kaibur_Tasks\Task_4
.idea
my-app
node_modules
public
src
App.css
App.js
App.test.js
App2.css
index.css
index.js
logo.svg
reportWebVitals.js
SearchById.css
SearchById.js
SearchByName.css
SearchByName.js
ServerForm.css
ServerForm.js
ServerTable.css
ServerTable.js
setupTests.js
package.json
package-lock.json
README.md
Kaibur (Task 4 -19BCG10069 - Documented Explaination).pdf
README.md
External Libraries
Scratches and Consoles
ServerTable.js
ServerTable.css
ServerForm.js
ServerForm.css
App.js
App2.css
README.md
App.css
App.tes

<label className="server-form__label" htmlFor="name-input">
Server Name:
</label>
<input
className="server-form__input"
type="text"
id="name-input"
name="name"
value={server.name}
onChange={handleChange}
/>

<label className="server-form__label" htmlFor="language-input">
Language:
</label>
<input
className="server-form__input"
type="text"
id="language-input"
name="language"
value={server.language}
onChange={handleChange}
/>

<label className="server-form__label" htmlFor="framework-input">
Framework:
</label>
Terminal: Local Local (2)
Notifications
Bookmarks
Structure
Pull Requests


File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
Task_4 - ServerForm.js - Administrator
Task_4 > my-app > src > ServerForm.js
Current File
Project
Task 4 E:\Kaibur_Tasks\Task_4
.idea
my-app
node_modules
public
src
App.css
App.js
App.test.js
App2.css
index.css
index.js
logo.svg
reportWebVitals.js
SearchById.css
SearchById.js
SearchByName.css
SearchByName.js
ServerForm.css
ServerForm.js
ServerTable.css
ServerTable.js
setupTests.js
package.json
package-lock.json
README.md
Kaibur (Task 4 -19BCG10069 - Documented Explaination).pdf
README.md
External Libraries
Scratches and Consoles
ServerForm.js
ServerForm.css
App.js
App2.css
README.md
App.css
App.test.js
SearchById.js

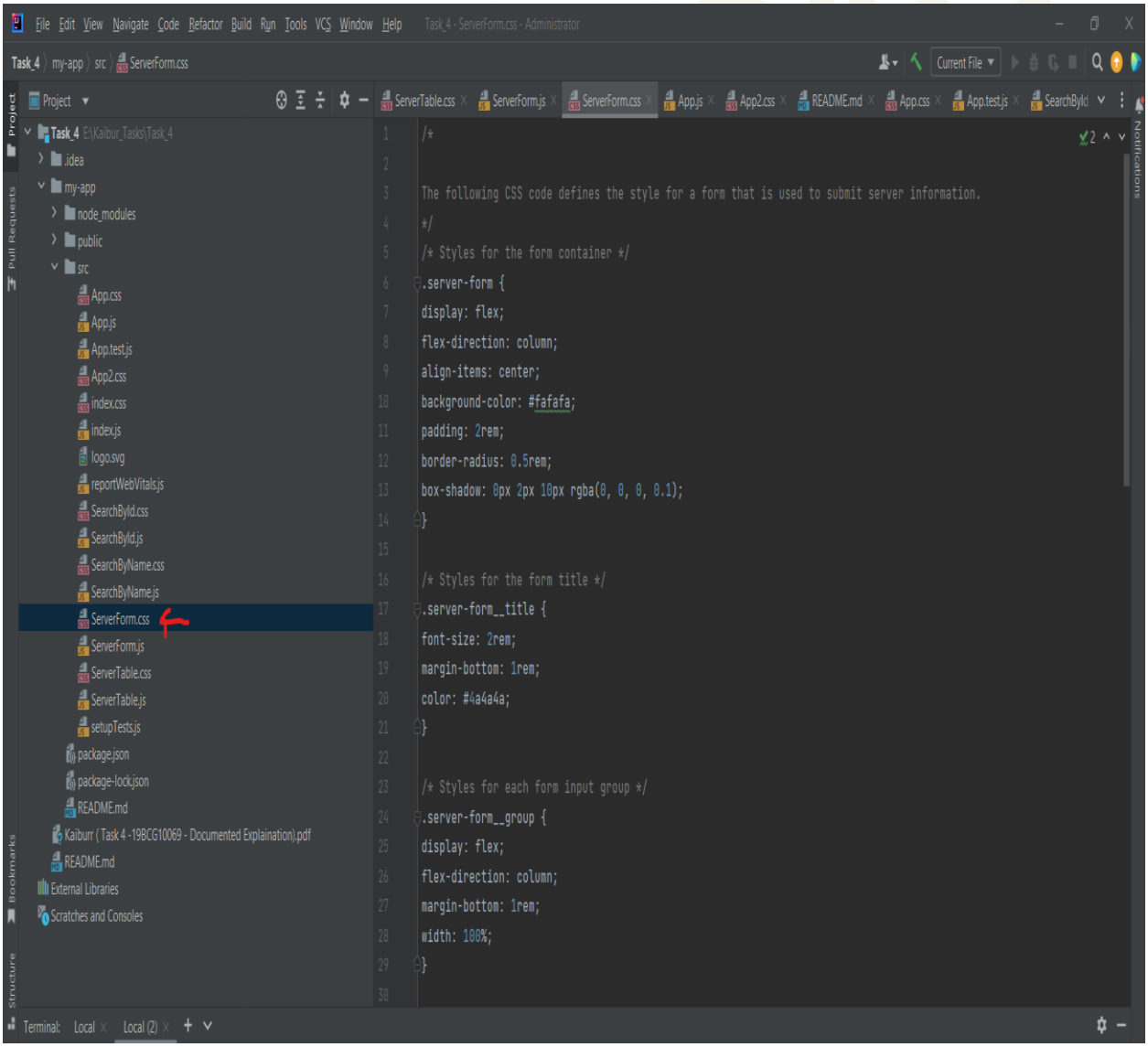
<label className="server-form__label" htmlFor="framework-input">
Framework:
</label>
<input
className="server-form__input"
type="text"
id="framework-input"
name="framework"
value={server.framework}
onChange={handleChange}
/>

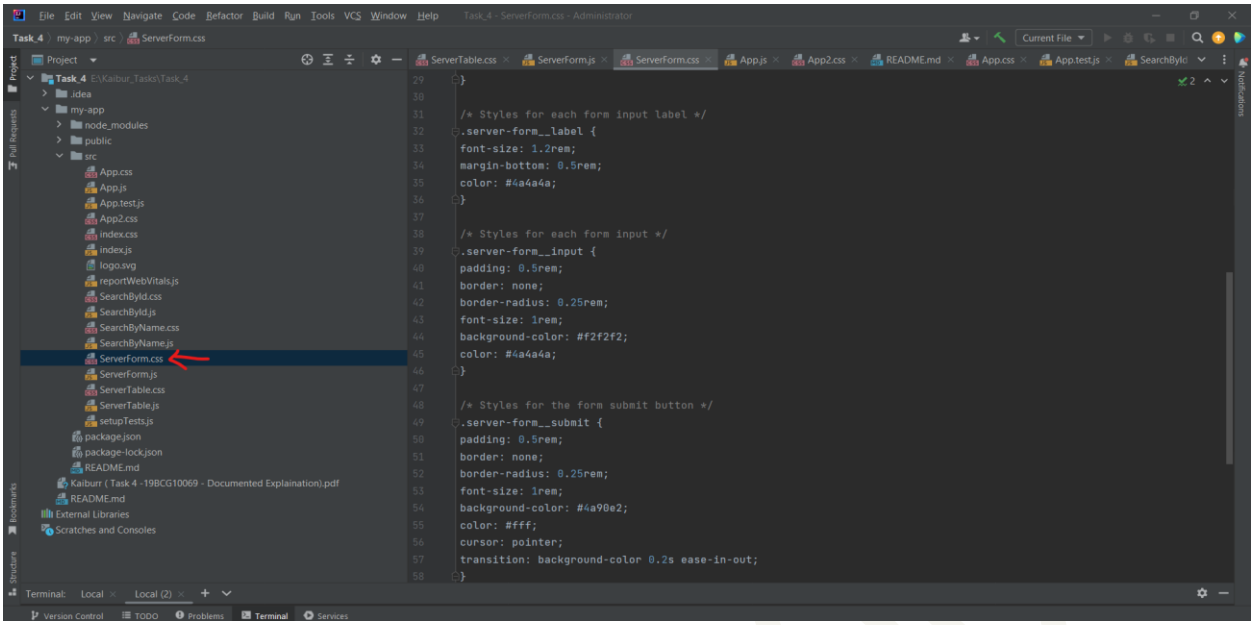
<button className="server-form__submit" type="submit" onClick={handleSubmit}>
Create
</button>
{message && {message} }
</form>
);
}
export default ServerForm;
// Code Written by @Arpit Singh 19BCG10069
Terminal: Local Local (2)
Notifications
Bookmarks
Structure
Pull Requests

- In summary, **the component uses axios to make a *POST request to a server endpoint, passing the server* object as the request body. When the server responds with a success status code, the component updates its state with a *success message* and clears the form. The component also provides an *onChange event listener* for *each input field to update the component's state* as the user types into the input field. Finally, the component provides a *button to submit the form and an onServerCreated prop* that gets called with the created server object when the server responds with a success status code.**

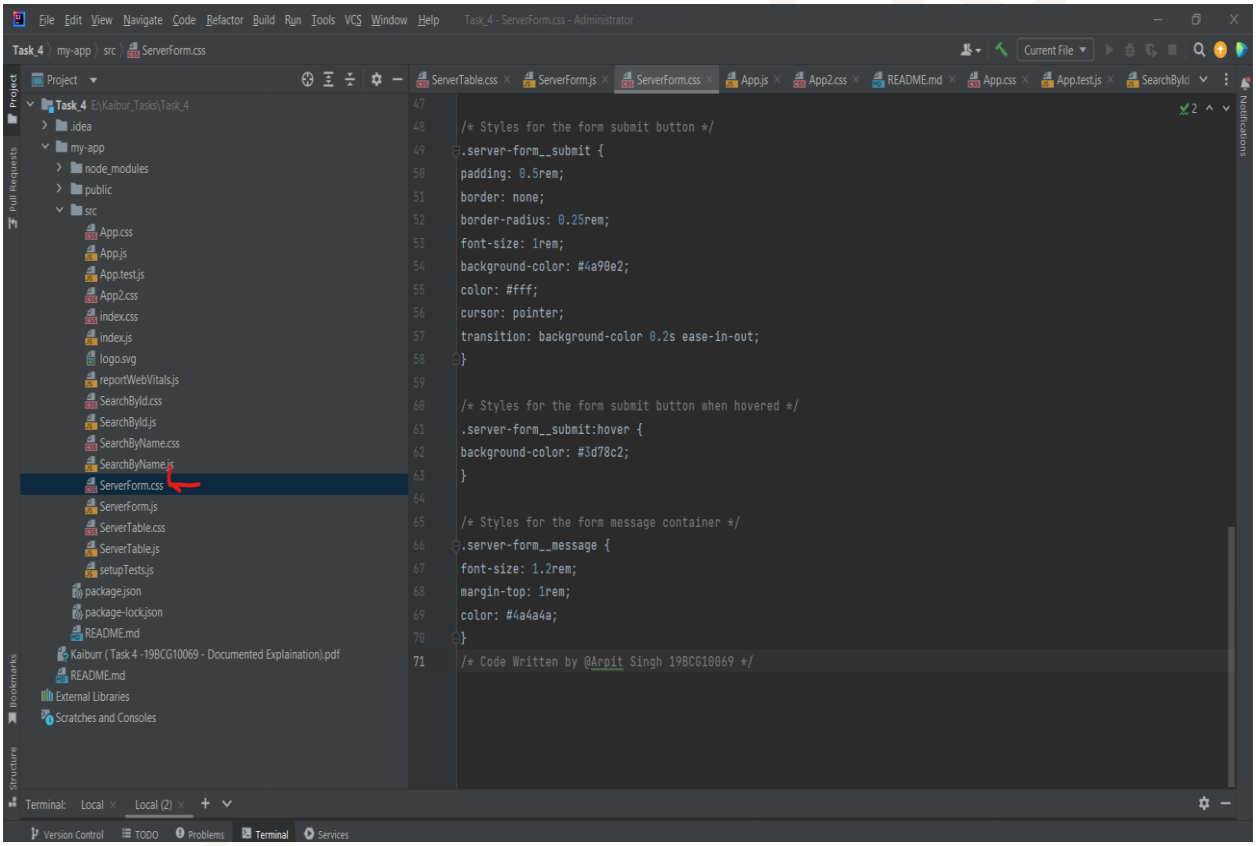
d. ***ServerForm.css(style)***:

i. Create a File named '***ServerForm.css***' with the ***following code***:





```
/* Styles for each form input label */
.server-form__label {
font-size: 1.2rem;
margin-bottom: 0.5rem;
color: #4a4a4a;
}

/* Styles for each form input */
.server-form__input {
padding: 0.5rem;
border: none;
border-radius: 0.25rem;
font-size: 1rem;
background-color: #f2f2f2;
color: #4a4a4a;
}

/* Styles for the form submit button */
.server-form__submit {
padding: 0.5rem;
border: none;
border-radius: 0.25rem;
font-size: 1rem;
background-color: #4a90e2;
color: #fff;
cursor: pointer;
transition: background-color 0.2s ease-in-out;
}
```



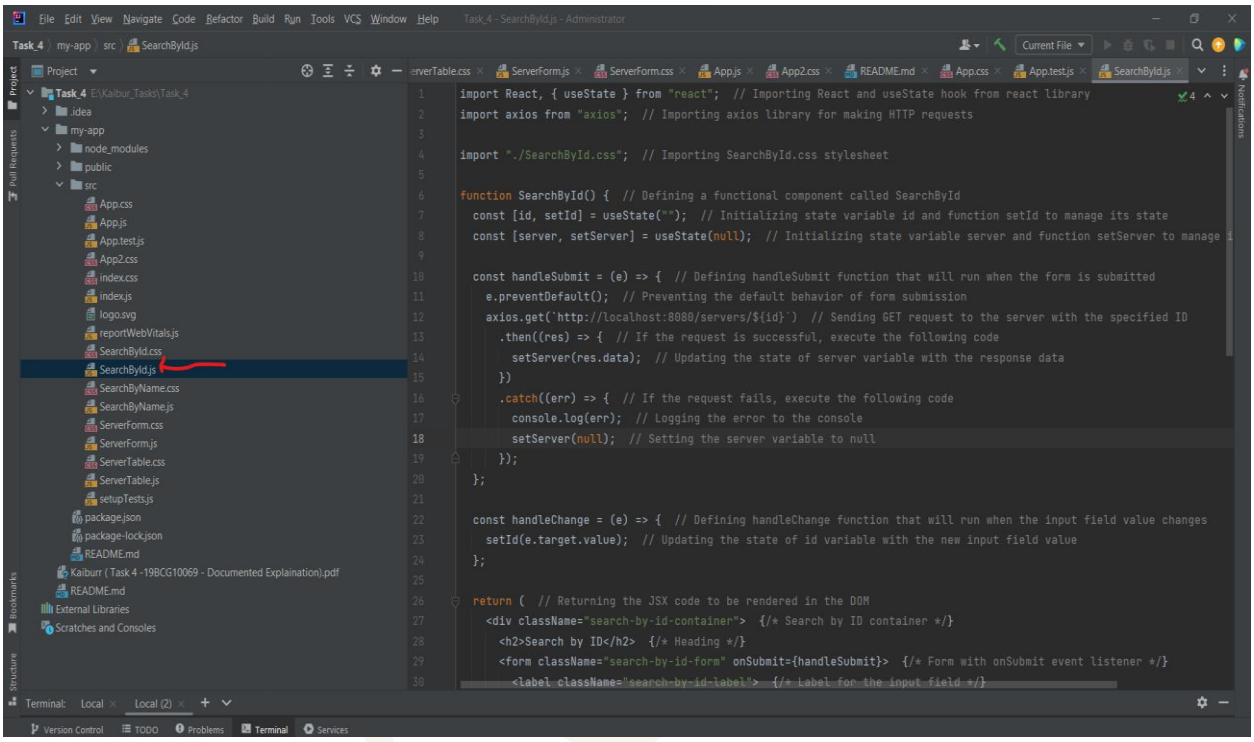
```
/* Styles for the form submit button */
.server-form__submit {
padding: 0.5rem;
border: none;
border-radius: 0.25rem;
font-size: 1rem;
background-color: #4a90e2;
color: #fff;
cursor: pointer;
transition: background-color 0.2s ease-in-out;
}

/* Styles for the form submit button when hovered */
.server-form__submit:hover {
background-color: #3d78c2;
}

/* Styles for the form message container */
.server-form__message {
font-size: 1.2rem;
margin-top: 1rem;
color: #4a4a4a;
}
/* Code Written by @Arpit Singh 19BCG10069 */
```

- In summary, **This CSS file contains styles for a form with a *title, input fields, and a submit button*. The styles define the layout and appearance of the form, including its background color, padding, border radius, box shadow, and font styles. The *input fields and submit button have specific styles* for their padding, border, border radius, font size, background color, and col-**

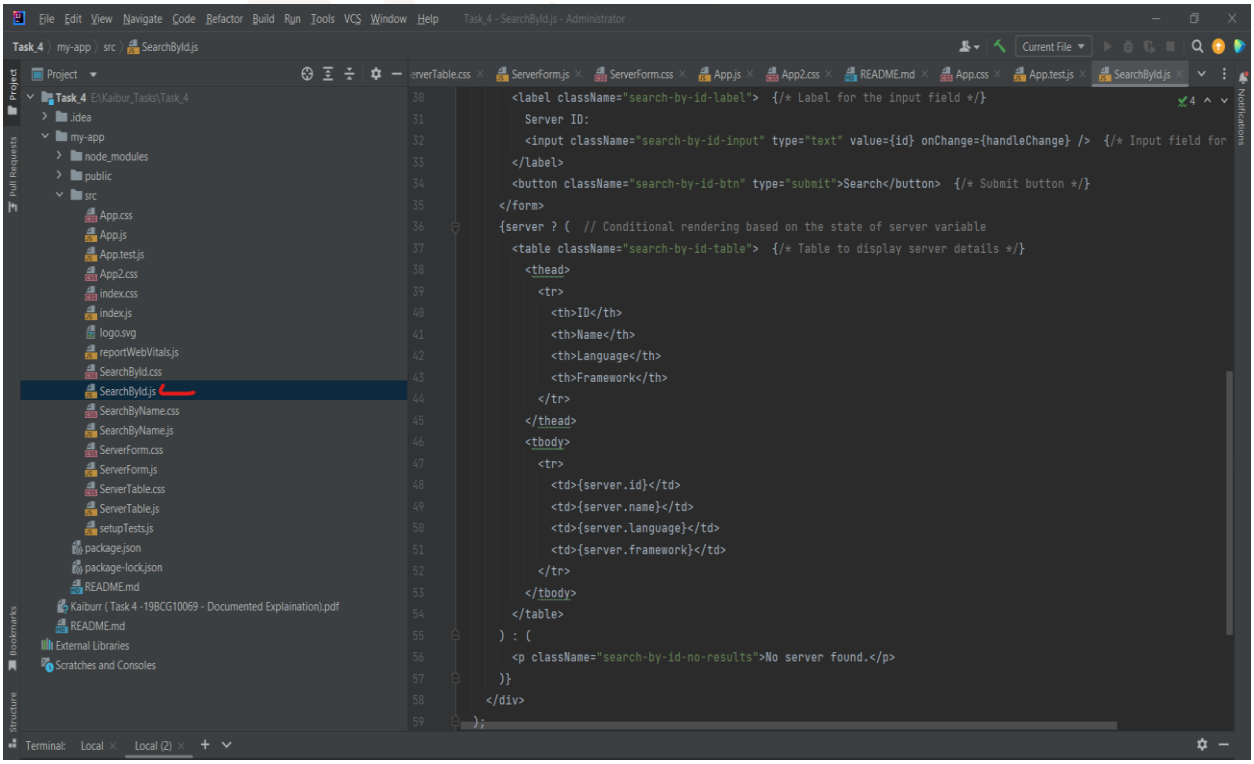
or. The *submit button also has a hover effect* to change its background color. Finally, there is a *message style* for displaying messages related to the form.

e. ***SearchById.js(logic)***:

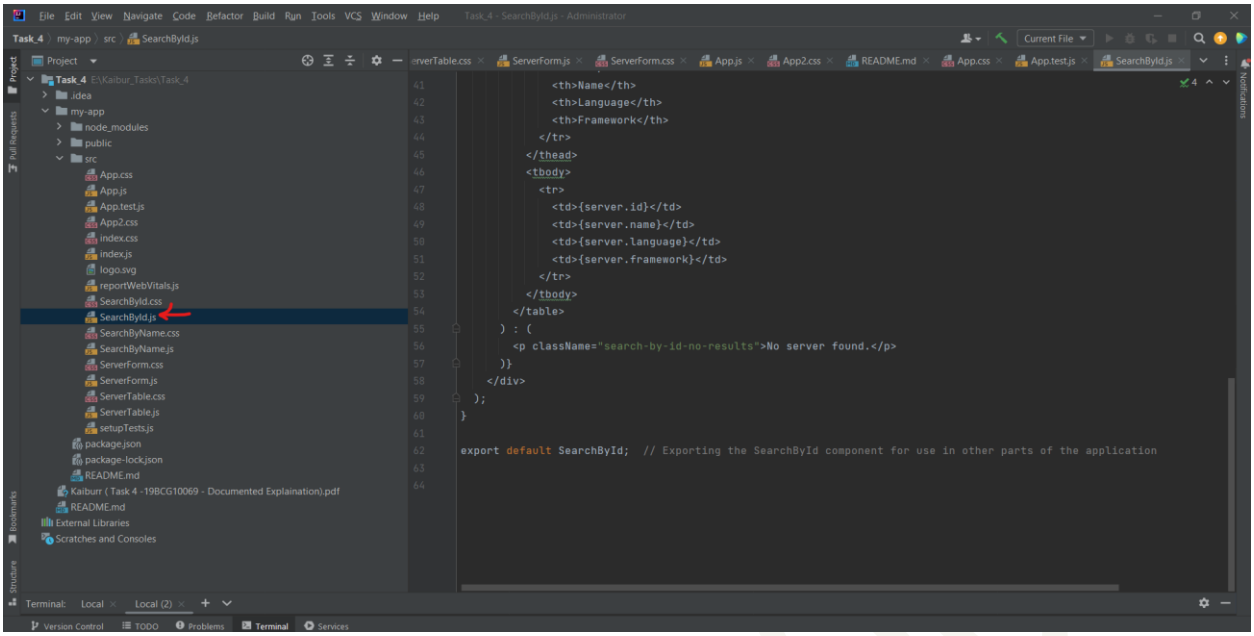
i. Create a File named '***SearchById.js***' with the ***following code***:



```
import React, { useState } from "react";  // Importing React and useState hook from react library
import axios from "axios";  // Importing axios library for making HTTP requests

import "./SearchById.css";  // Importing SearchById.css stylesheet

function SearchById() {  // Defining a functional component called SearchById
  const [id, setId] = useState("");  // Initializing state variable id and function setId to manage its state
  const [server, setServer] = useState(null);  // Initializing state variable server and function setServer to manage i

  const handleSubmit = (e) => {  // Defining handleSubmit function that will run when the form is submitted
    e.preventDefault();  // Preventing the default behavior of form submission
    axios.get(`http://localhost:8080/servers/${id}`)  // Sending GET request to the server with the specified ID
      .then((res) => {  // If the request is successful, execute the following code
        setServer(res.data);  // Updating the state of server variable with the response data
      })
      .catch((err) => {  // If the request fails, execute the following code
        console.log(err);  // Logging the error to the console
        setServer(null);  // Setting the server variable to null
      });
  };

  const handleChange = (e) => {  // Defining handleChange function that will run when the input field value changes
    setId(e.target.value);  // Updating the state of id variable with the new input field value
  };

  return (  // Returning the JSX code to be rendered in the DOM
    <div className="search-by-id-container">  {/* Search by ID container */}
      <h2>Search by ID</h2>  {/* Heading */}
      <form className="search-by-id-form" onSubmit={handleSubmit}>  {/* Form with onSubmit event listener */}
        <label className="search-by-id-label">  {/* Label for the input field */}
```



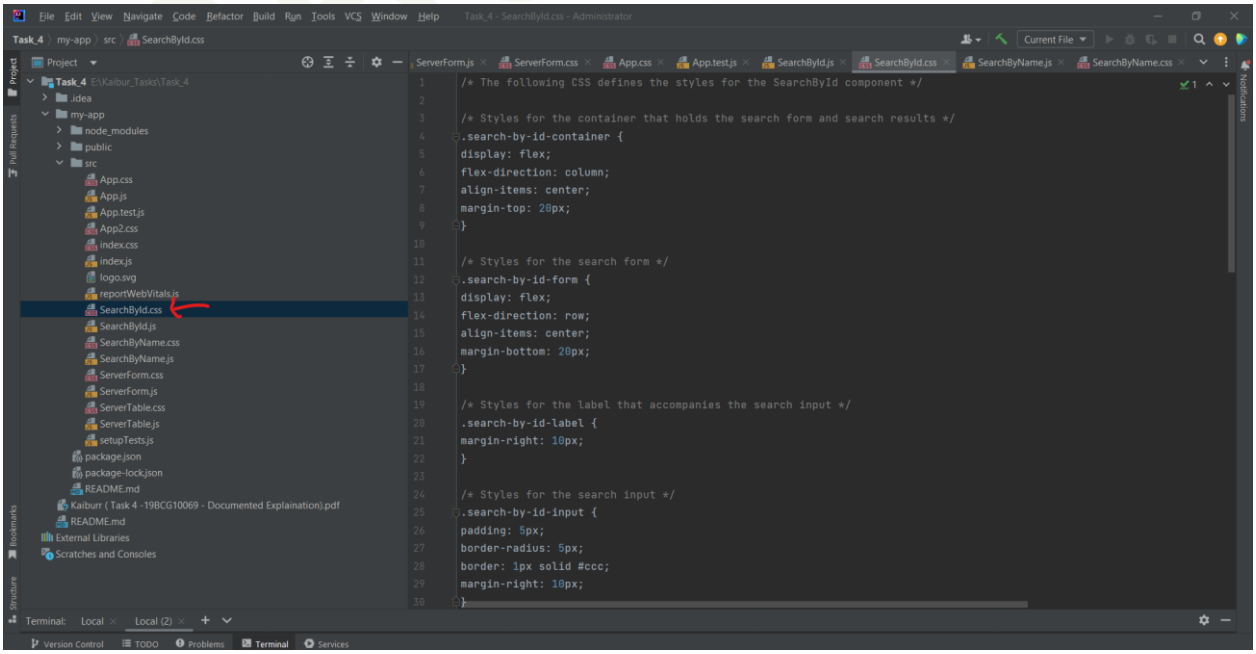
```
        <label className="search-by-id-label">  {/* Label for the input field */}
          Server ID:
          <input className="search-by-id-input" type="text" value={id} onChange={handleChange} />  {/* Input field for
        </label>
        <button className="search-by-id-btn" type="submit">Search</button>  {/* Submit button */}
      </form>
      {server ? (  // Conditional rendering based on the state of server variable
        <table className="search-by-id-table">  {/* Table to display server details */}
          <thead>
            <tr>
              <th>ID</th>
              <th>Name</th>
              <th>Language</th>
              <th>Framework</th>
            </tr>
          </thead>
          <tbody>
            <tr>
              <td>{server.id}</td>
              <td>{server.name}</td>
              <td>{server.language}</td>
              <td>{server.framework}</td>
            </tr>
          </tbody>
        </table>
      ) : (
        <p className="search-by-id-no-results">No server found.</p>
      )}
    </div>
  );
```

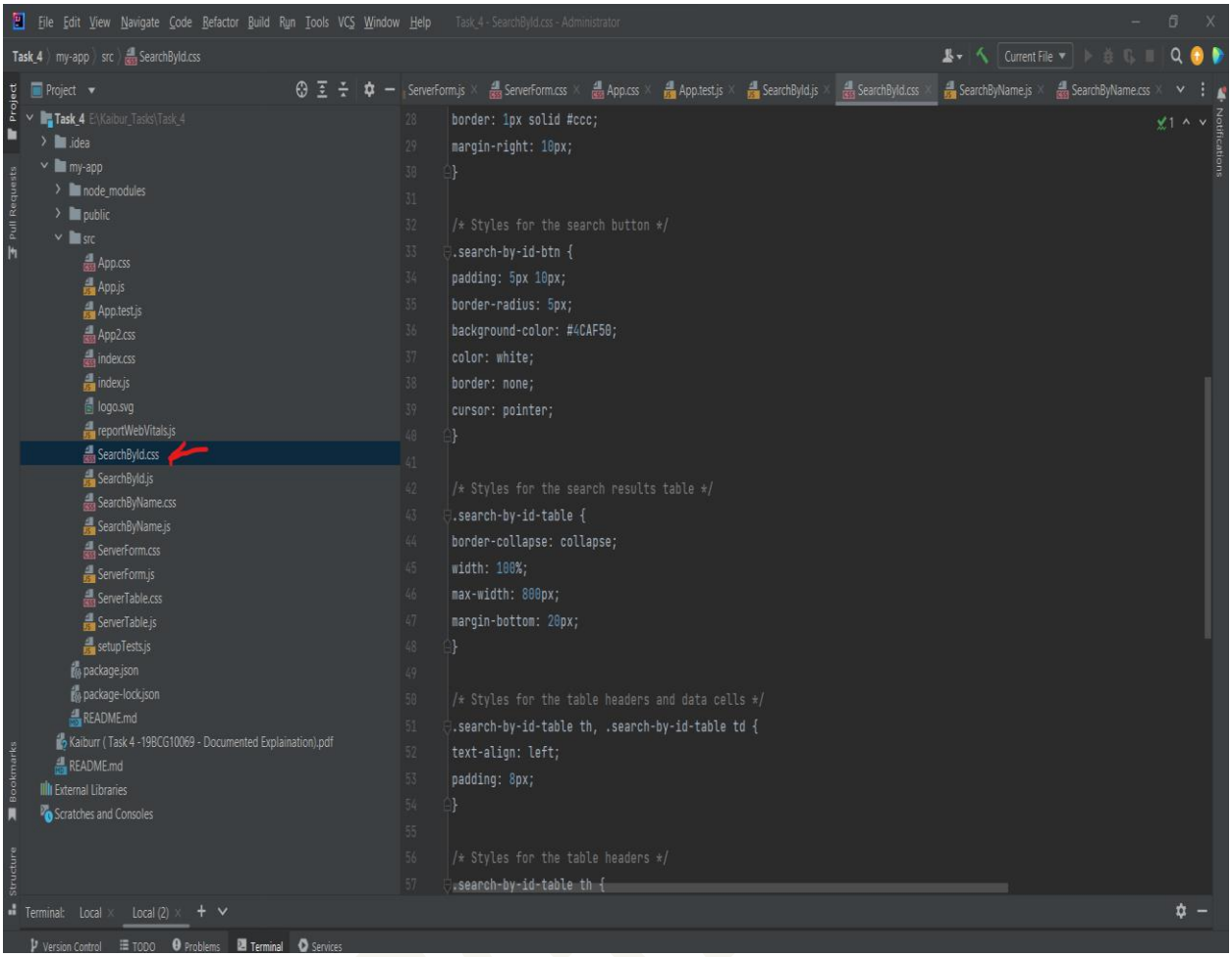


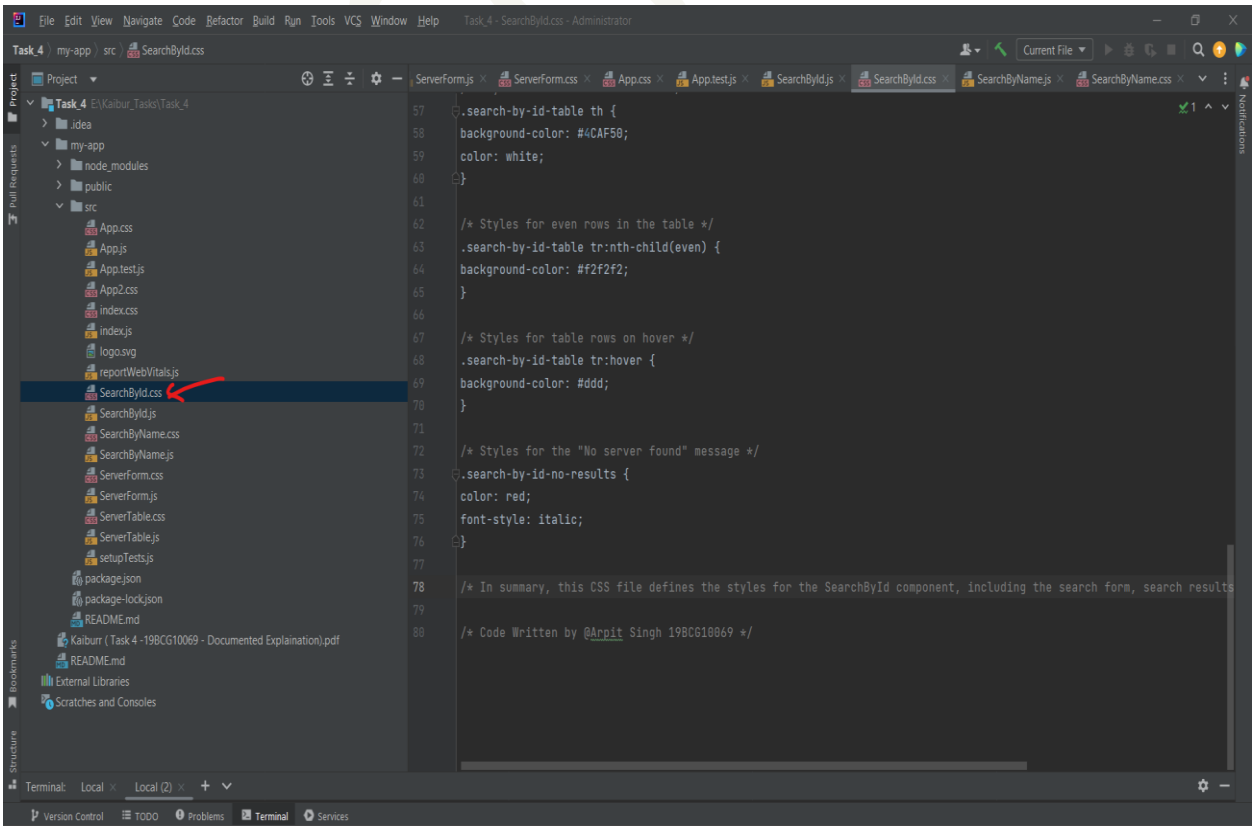
- In summary, **this code defines a functional component *SearchById* that allows the user to *search for a server in a database by its ID*. It renders a form with an input field for the user to *enter the server ID and a submit button*. When the user submits the form, an HTTP *GET request is sent to an API endpoint* with the ID parameter. If the request is *successful, the server data is returned and displayed in a table*. If the request fails, an error is logged to the console and a message is displayed stating that *no server was found*.**

f. ***SearchById.css(style)***:

i. Create a File named '***SearchById.css***' with the ***following code***:



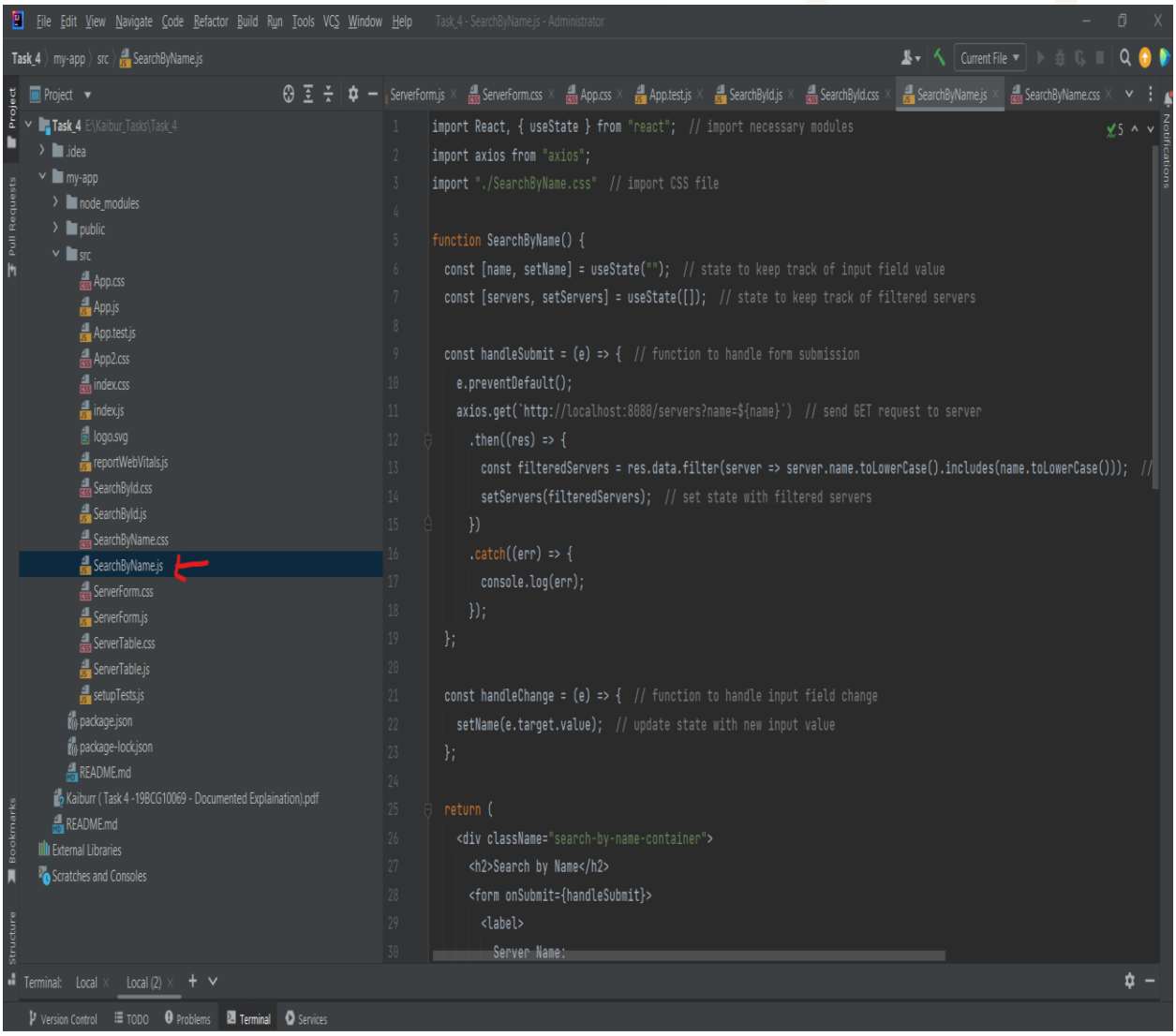

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
Task_4 - SearchById.css - Administrator
Task_4) my-app) src) SearchById.css
Current File
Project
ServerForm.js
ServerForm.css
App.css
App.test.js
SearchById.js
SearchById.css
SearchByName.js
SearchByName.css
Task_4 E:\Kaibur_Tasks\Task_4
.idea
my-app
node_modules
public
src
App.css
App.js
App.test.js
App2.css
index.css
index.js
logo.svg
reportWebVitals.js
SearchById.css
SearchById.js
SearchByName.css
SearchByName.js
ServerForm.css
ServerForm.js
ServerTable.css
ServerTable.js
setupTests.js
package.json
package-lock.json
README.md
Kaiburr (Task 4 -19BCG10069 - Documented Explaination).pdf
README.md
External Libraries
Scratches and Consoles
Terminal: Local Local (2)
Version Control TODO Problems Terminal Services
28 border: 1px solid #ccc;
29 margin-right: 10px;
30 }
31
32 /* Styles for the search button */
33 .search-by-id-btn {
34 padding: 5px 10px;
35 border-radius: 5px;
36 background-color: #4CAF50;
37 color: white;
38 border: none;
39 cursor: pointer;
40 }
41
42 /* Styles for the search results table */
43 .search-by-id-table {
44 border-collapse: collapse;
45 width: 100%;
46 max-width: 800px;
47 margin-bottom: 20px;
48 }
49
50 /* Styles for the table headers and data cells */
51 .search-by-id-table th, .search-by-id-table td {
52 text-align: left;
53 padding: 8px;
54 }
55
56 /* Styles for the table headers */
57 .search-by-id-table th {

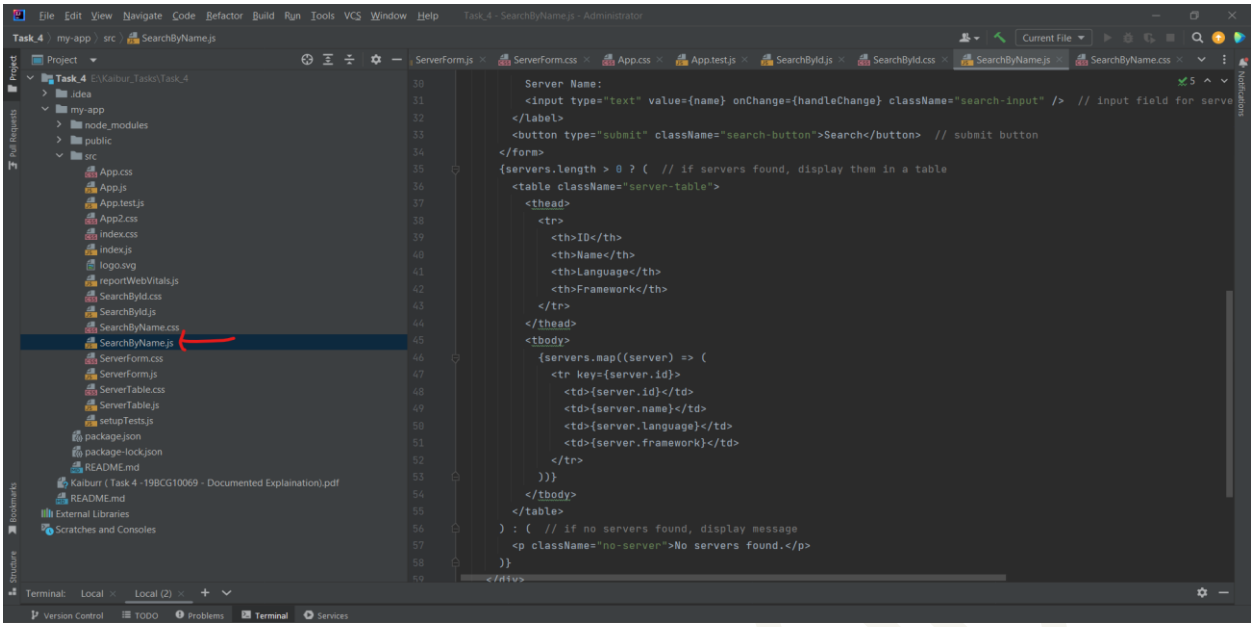

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
Task_4 - SearchById.css - Administrator
Task_4) my-app) src) SearchById.css
Current File
Project
ServerForm.js
ServerForm.css
App.css
App.test.js
SearchById.js
SearchById.css
SearchByName.js
SearchByName.css
Task_4 E:\Kaibur_Tasks\Task_4
.idea
my-app
node_modules
public
src
App.css
App.js
App.test.js
App2.css
index.css
index.js
logo.svg
reportWebVitals.js
SearchById.css
SearchById.js
SearchByName.css
SearchByName.js
ServerForm.css
ServerForm.js
ServerTable.css
ServerTable.js
setupTests.js
package.json
package-lock.json
README.md
Kaiburr (Task 4 -19BCG10069 - Documented Explaination).pdf
README.md
External Libraries
Scratches and Consoles
Terminal: Local Local (2)
Version Control TODO Problems Terminal Services
57 .search-by-id-table th {
58 background-color: #4CAF50;
59 color: white;
60 }
61
62 /* Styles for even rows in the table */
63 .search-by-id-table tr:nth-child(even) {
64 background-color: #f2f2f2;
65 }
66
67 /* Styles for table rows on hover */
68 .search-by-id-table tr:hover {
69 background-color: #ddd;
70 }
71
72 /* Styles for the "No server found" message */
73 .search-by-id-no-results {
74 color: red;
75 font-style: italic;
76 }
77
78 /* In summary, this CSS file defines the styles for the SearchById component, including the search form, search results
79
80 /* Code Written by @Arpit Singh 19BCG10069 */

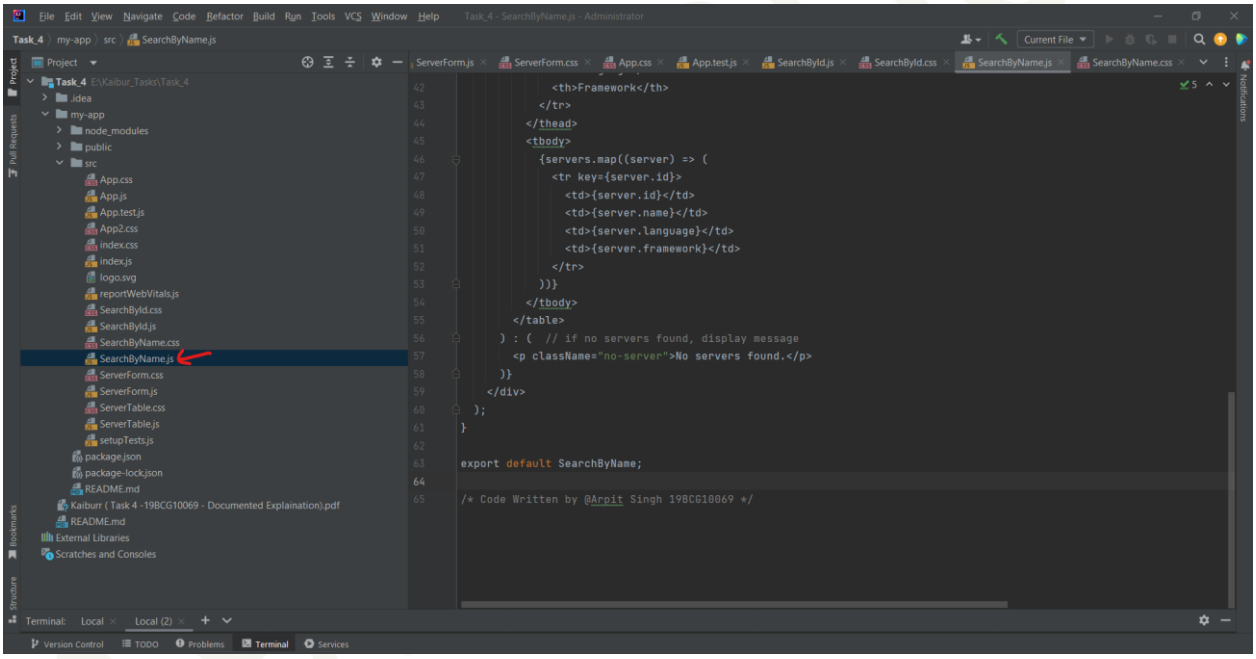
- In summary, **This CSS file contains styling rules for the components in the *SearchById React component*. It styles the *container, form, input fields, button, table, and message* that are used to search for and display server data. The styling includes *flexbox layout, borders, colors, font styles, and responsive design* for different screen sizes.**

g. ***SearchByName.js(logic)***:

i. Create a File named '***SearchByName.js***' with the ***following code***:



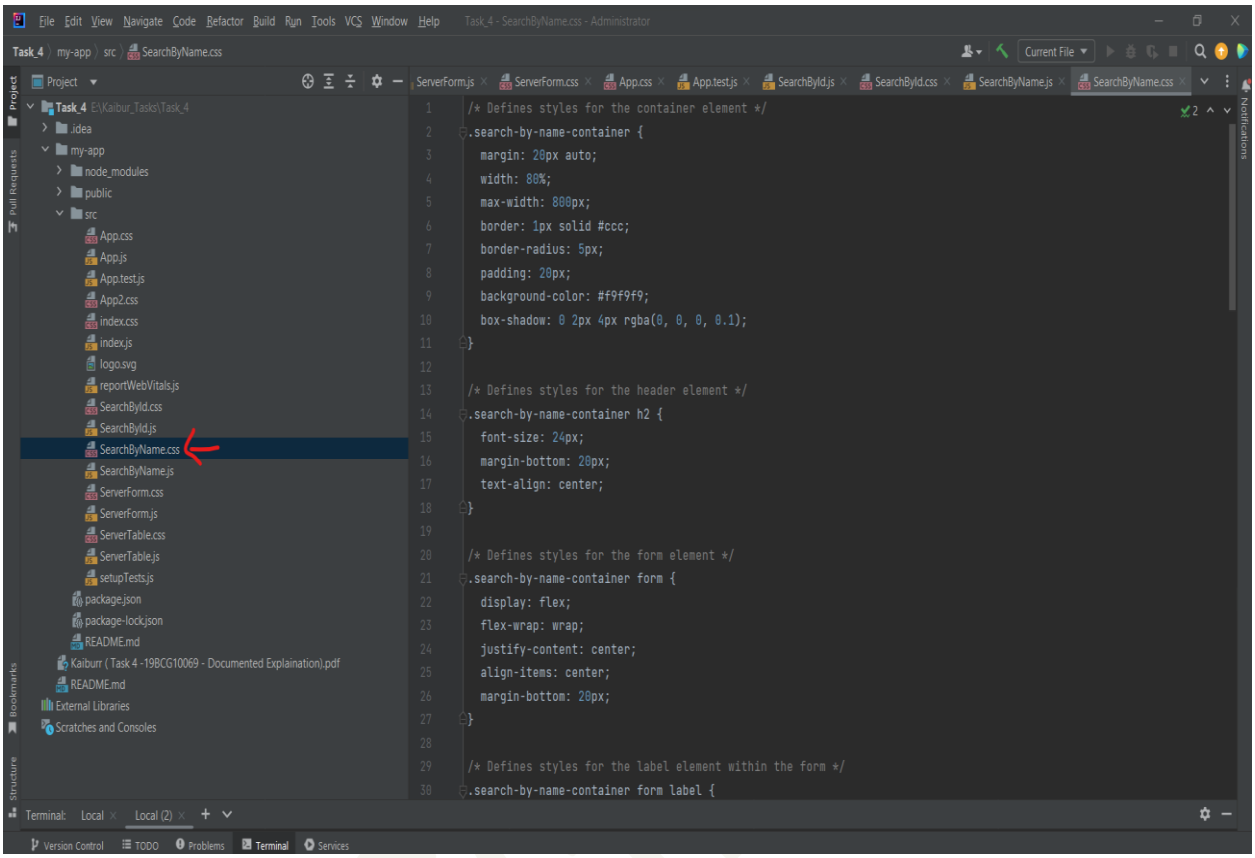




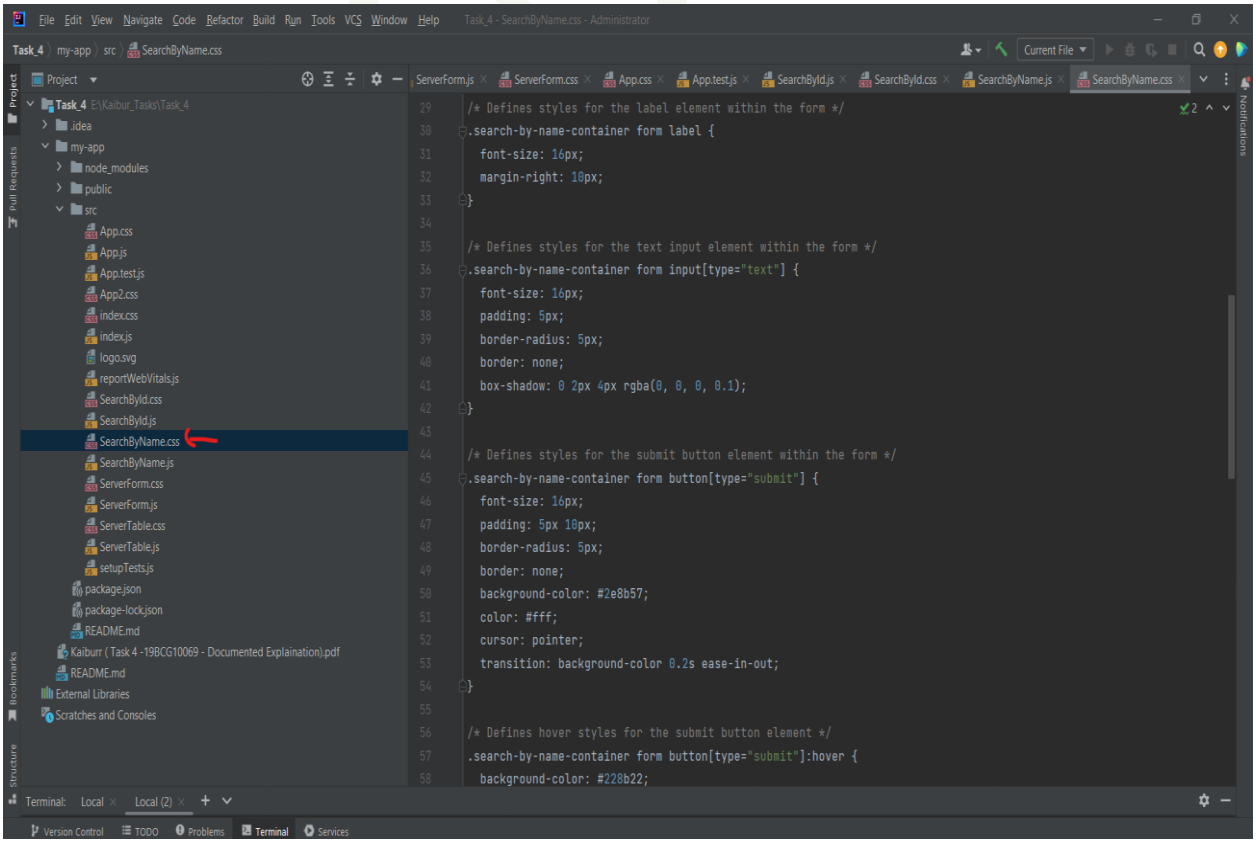
- In summary, **this code creates a *React component for searching servers by name*. It includes a form with an input field and submit button for entering and submitting the server name to search for. Upon form submission, a *GET request is sent to a server to retrieve a list of servers*. The retrieved servers are then *filtered based on the entered server name*, and the filtered servers are displayed in a table if any are found. If *no servers are found, a message is displayed instead.***

h. ***SearchByName.css(style)***:

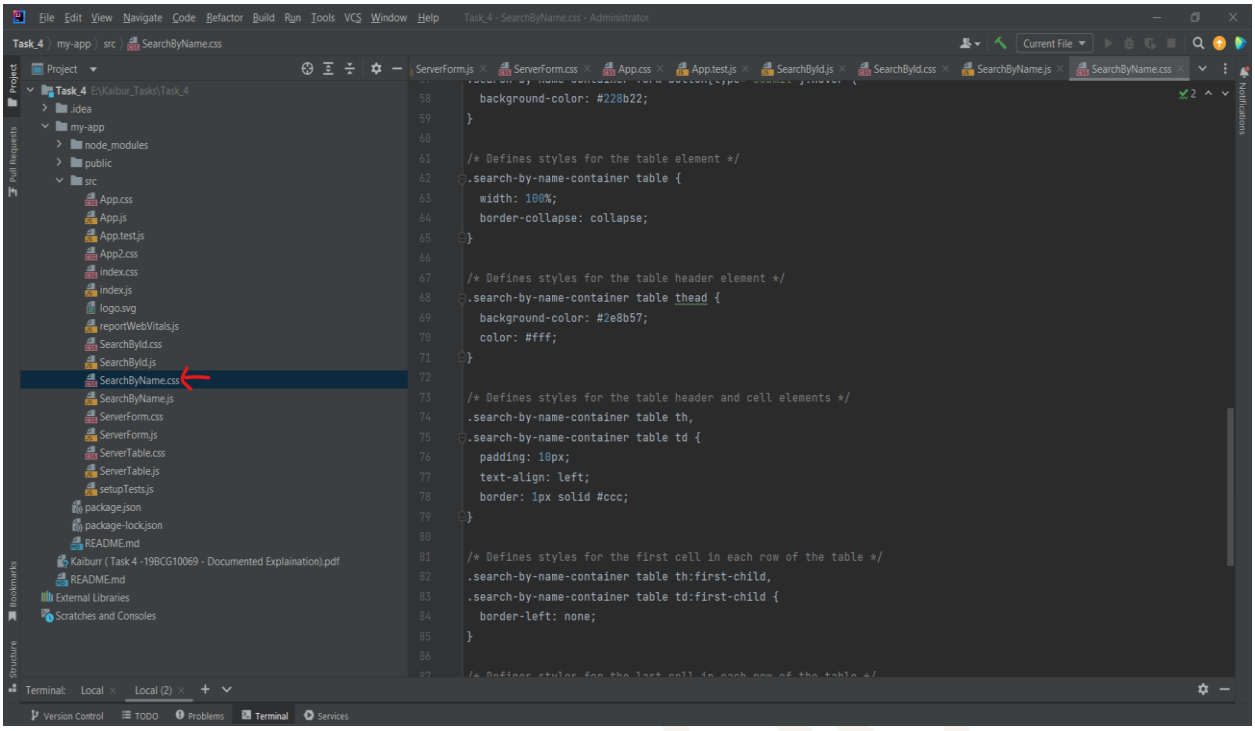
i. Create a File named '***SearchByName.css***' with the ***following code***:

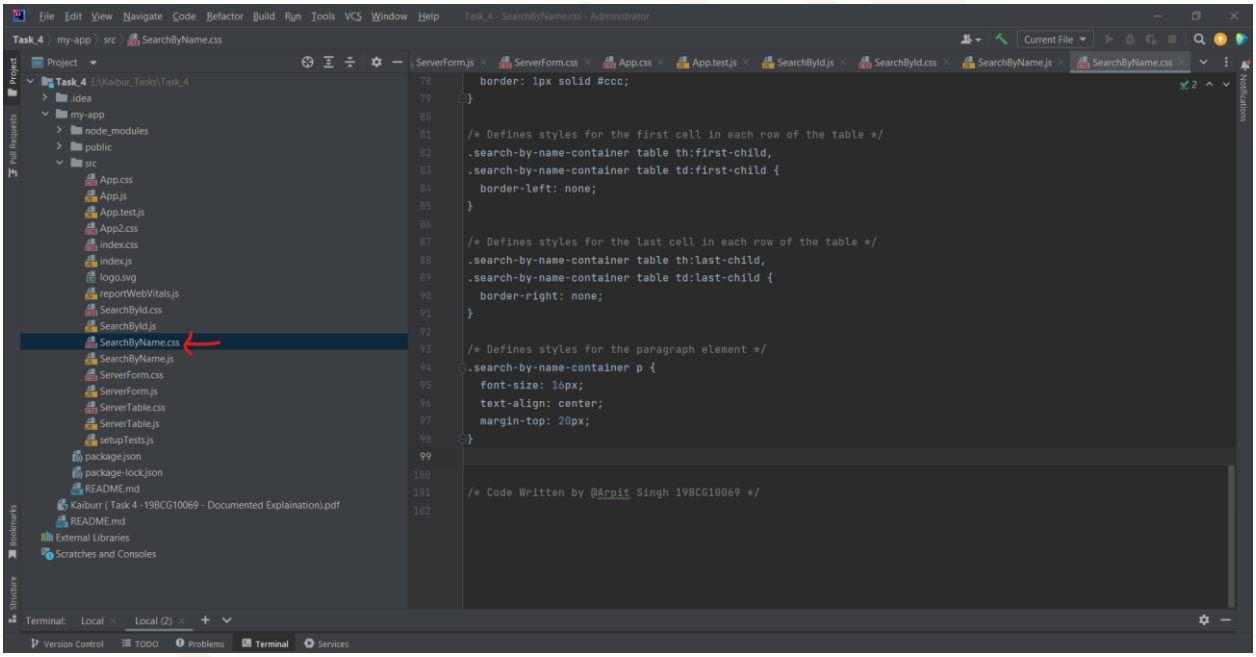


```
/* Defines styles for the container element */
.search-by-name-container {
  margin: 20px auto;
  width: 80%;
  max-width: 800px;
  border: 1px solid #ccc;
  border-radius: 5px;
  padding: 20px;
  background-color: #f9f9f9;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}

/* Defines styles for the header element */
.search-by-name-container h2 {
  font-size: 24px;
  margin-bottom: 20px;
  text-align: center;
}

/* Defines styles for the form element */
.search-by-name-container form {
  display: flex;
  flex-wrap: wrap;
  justify-content: center;
  align-items: center;
  margin-bottom: 20px;
}

/* Defines styles for the label element within the form */
.search-by-name-container form label {
```



```
/* Defines styles for the label element within the form */
.search-by-name-container form label {
  font-size: 16px;
  margin-right: 10px;
}

/* Defines styles for the text input element within the form */
.search-by-name-container form input[type="text"] {
  font-size: 16px;
  padding: 5px;
  border-radius: 5px;
  border: none;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}

/* Defines styles for the submit button element within the form */
.search-by-name-container form button[type="submit"] {
  font-size: 16px;
  padding: 5px 10px;
  border-radius: 5px;
  border: none;
  background-color: #2e8b57;
  color: #fff;
  cursor: pointer;
  transition: background-color 0.2s ease-in-out;
}

/* Defines hover styles for the submit button element */
.search-by-name-container form button[type="submit"]:hover {
  background-color: #228b22;
```



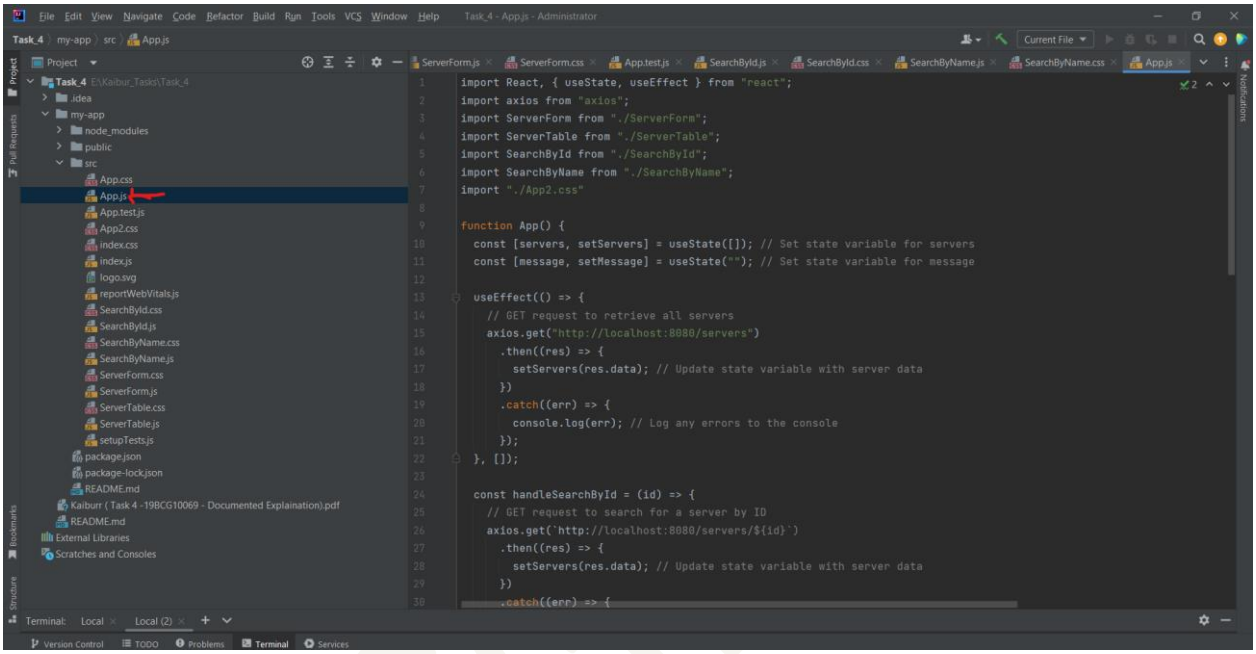


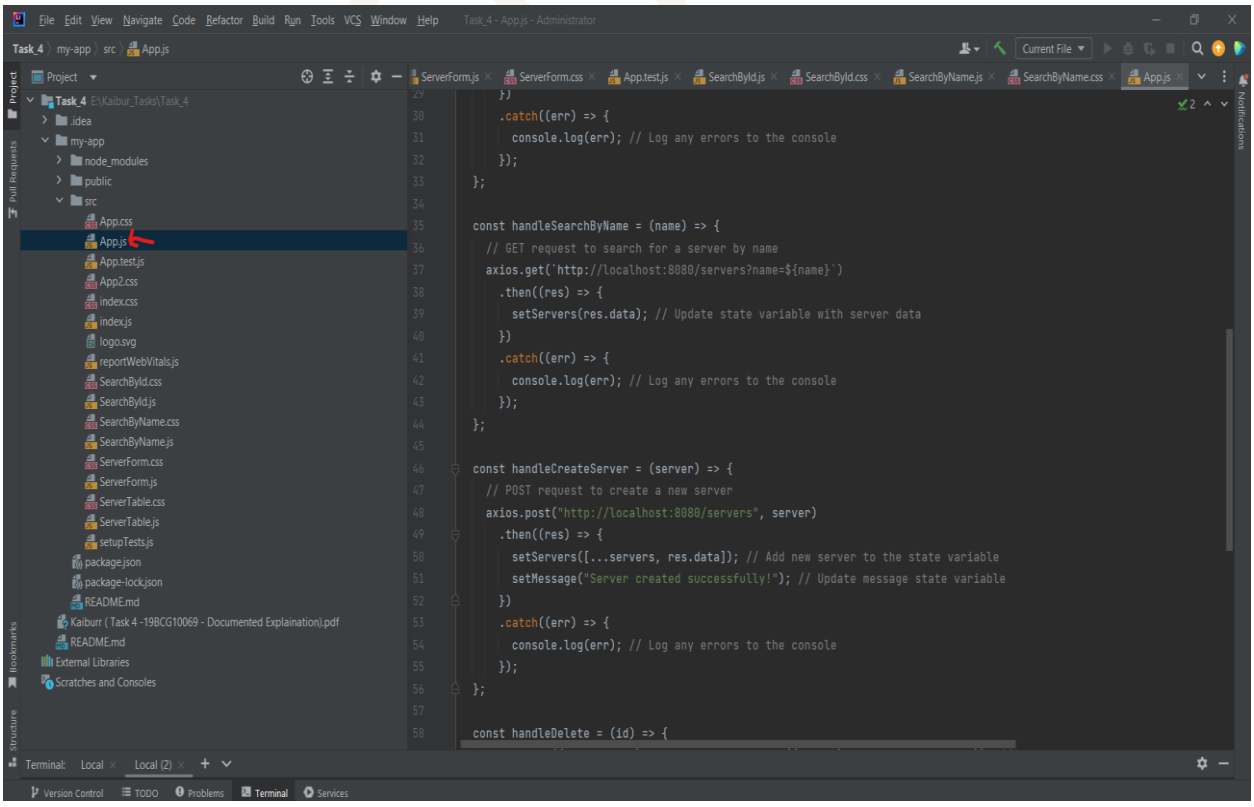
- In summary, **this is a CSS file that styles the *SearchByName component*. It defines the *appearance of the search form*, the table that displays *the search results, and the message that appears when no search results are found.* The styles include margins, padding, borders, font sizes, colors, and transitions. The CSS uses classes *to target specific elements in the component*, such as the container, the form, the input field, the submit button, the table, and the table headers and cells. The CSS also includes *hover effects for the sub-***

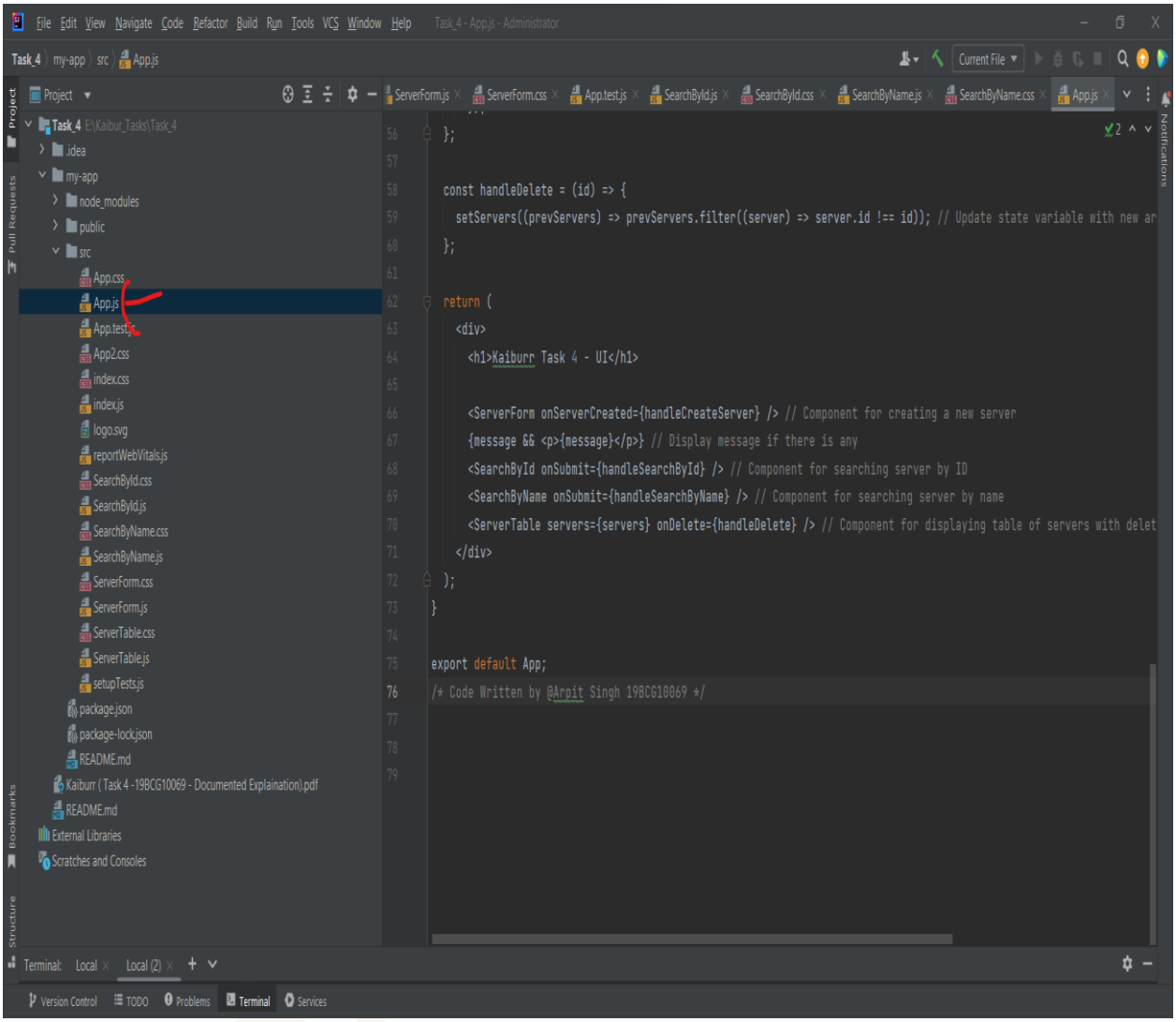
mit button and removes **the left and right borders of the first and last table cells, respectively.**

i. ***App.js(logic – entry point to our Application)***:

i. Create a File named '***App.js***' with the ***following code***:



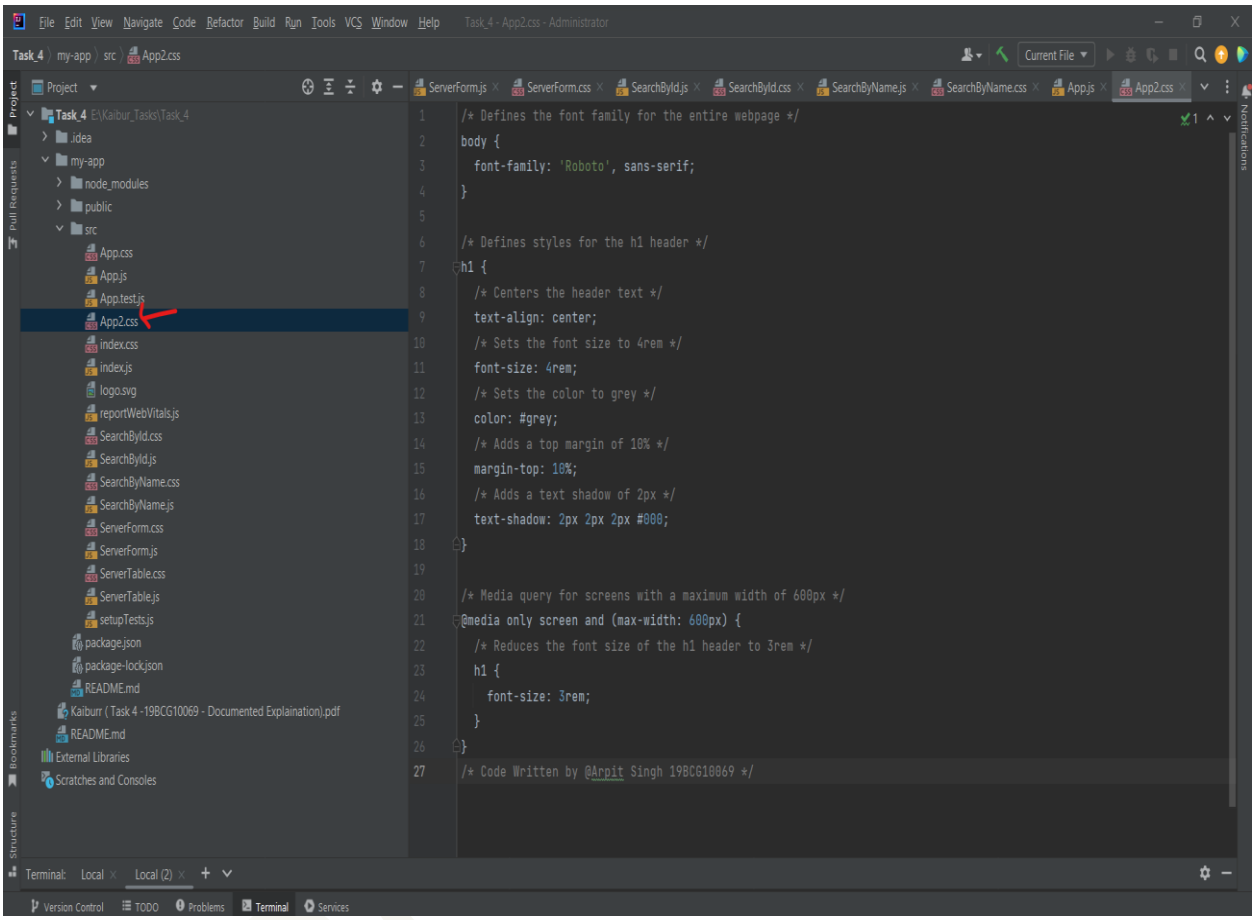




- In summary, this code defines a *React functional component called "**App**"*. The component sets up a ***UI for managing servers, including creating new servers, searching for servers by ID or name, and deleting servers***. The component uses ***hooks to manage state, including an array of servers, a message*** to display when a server is successfully created, and functions to handle server operations. *The component makes HTTP requests to a REST API running on **http://localhost:8080/servers** using the Axios library* to retrieve, create, and delete servers. The component renders a header, a form for creating new servers, search forms for searching for servers by ID or name, a table for displaying the list of servers, and a message displaying the success of the server creation operation.

j. ***App2.css(style – App.css already exists by default)***:

i. Create a File named '***App2.css***' with the ***following code***:



- In summary, **this code sets the *font family for the body of a webpage* to Roboto and defines styles for an *h1 header element*. It centres the header text, sets the font size to 4rem, the color to grey, adds a top margin of 10%, and a text shadow of 2px. It also includes a *media query that reduces the font size* of the header to 3rem for screens with a *maximum width of 600px*.**

- **That's it!** All the ***functional components*** required for the ***UI have been created*** along with the logic of handling/sending ***requests to our REST API (Task 1)***.

- **Problem:** the application will run fine with "***npm start***" command and the UI as we defined will also be displayed ***but the functionality will not work***. This Problem is caused due to **CORS policies**.

4. **Starting REST API and Handling CORS policies problem:**
    a. ***Solving CORS policy***:
        i. So, this problem arises because my ***Rest Api is deployed at the "port: 8080" and my React app is working with "port: 3000"***.

        ii. According to CORS policy a ***port cannot send request to another port*** until explicitly mentioned or using a proxy.

Error: Access to XMLHttpRequest at 'http://localhost:8080/servers' from origin 'http://localhost:3000' has been blocked by CORS policy: Response to preflight request doesn't pass access control check: No 'Access-Control-Allow-Origin' header is present on the requested resource.

        iii. This ***error message*** indicates that ***the browser is preventing the API call from being made due to a CORS policy violation***. The CORS policy is a security feature implemented by web browsers that ***restricts web pages from making requests to a different domain*** than the one the page was served from.

        iv. To ***resolve this issue***, you need to configure your ***server to allow cross-origin requests*** from the domain that your React application is running on.

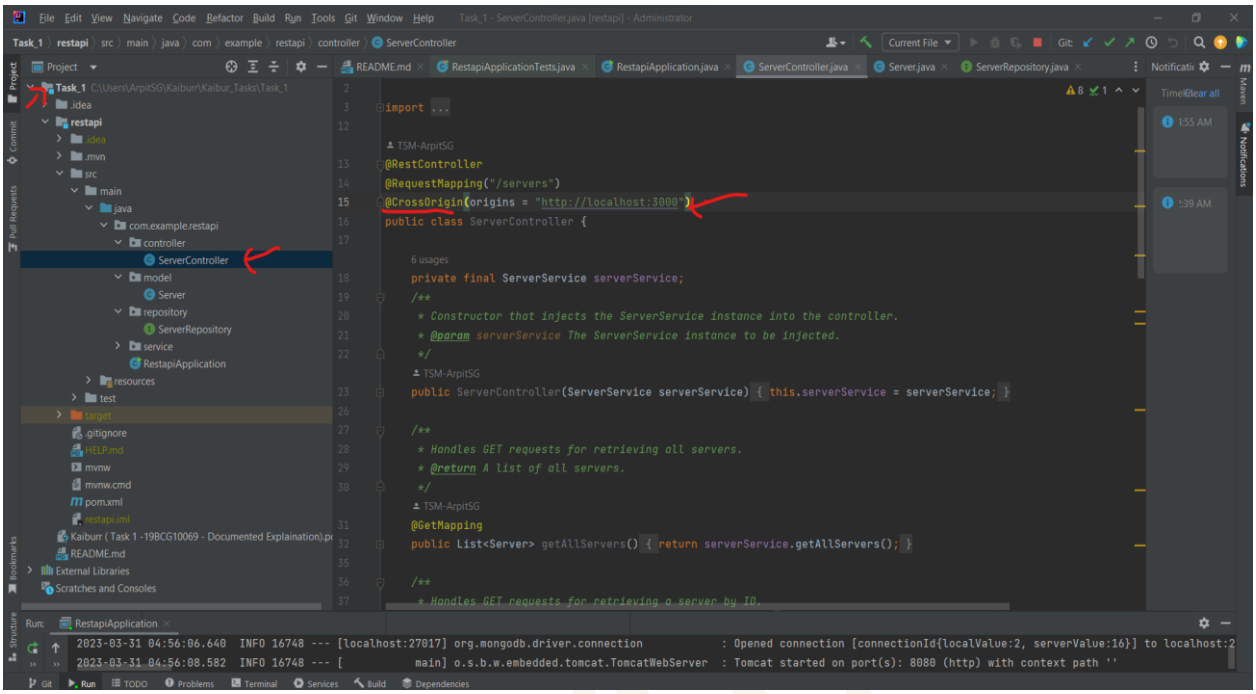
        v. One way to ***enable CORS on your server*** is to add the following **header to the response for each API request**:

        `Access-Control-Allow-Origin: http://localhost:3000`

        vi. To allow cross-origin requests from the specified origin ***http://localhost:3000***, you can add the **"@CrossOrigin"** annotation to your ***controller class or individual methods***. Here's an example of how to add it to our "**ServerController**" class:

```
"C:\Users\ArpitSG\Kaiburr\Kaibur_Tasks\Task_1\restapi\src\main\java\com\example\restapi\controller\ServerController.java"
```

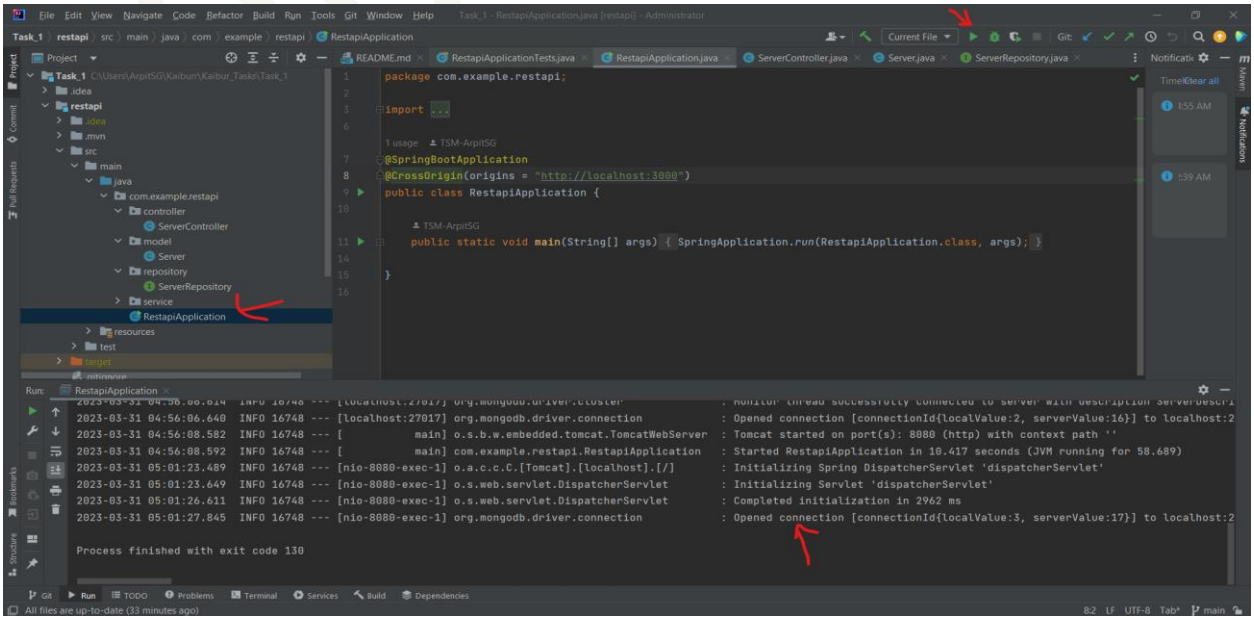
vii. Therefore we need to make some ***changes in the REST API*** we created in ***Task 1***.



b. ***Starting Task 1 Rest Api*:**

i. For the ***requests sent by user*** through the react app the ***REST API must be on***.

ii. Therefore, ***LAUNCH*** the application we created in ***Task 1(https://github.com/TSM-ArpitSG/Kaiburr/tree/main/Kaibur_Tasks/Task_1)***.

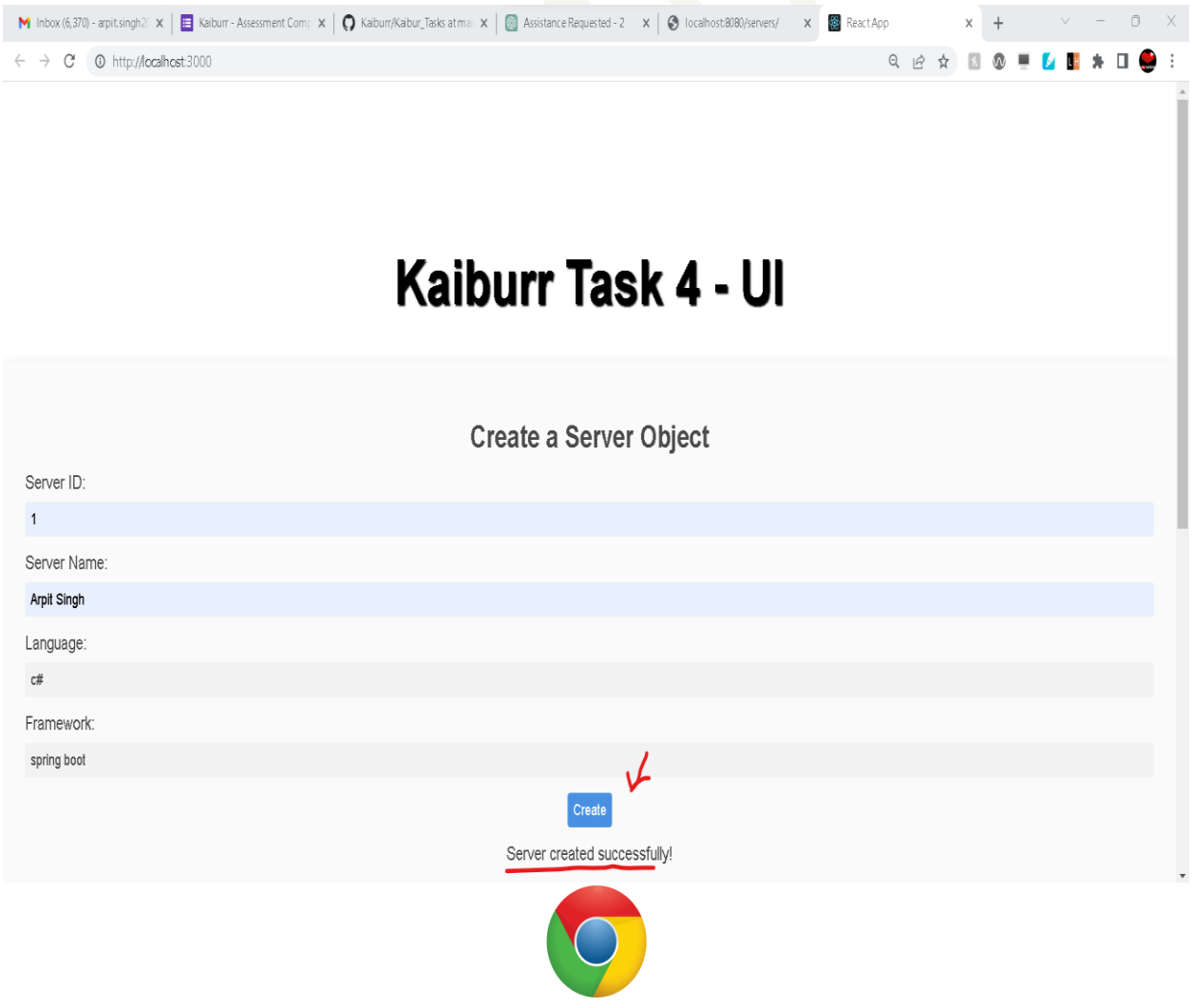


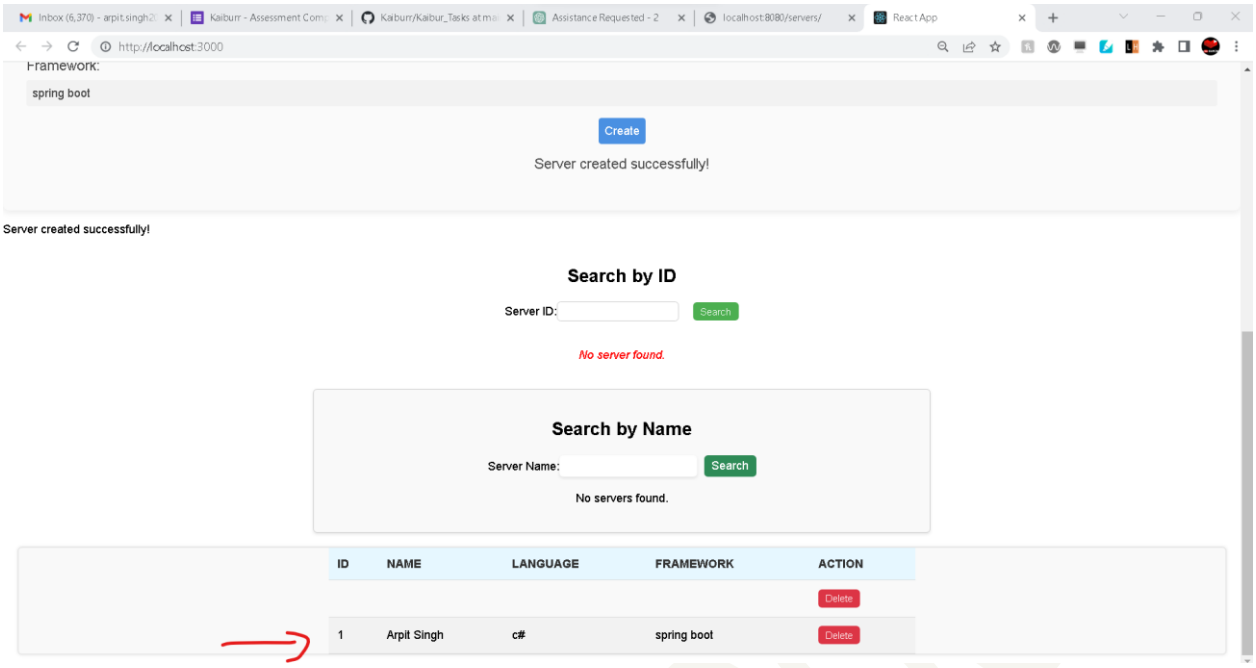
- Now we are good to go! We can ***start our React App by running "npm start"***☺.

```
PS E:\Kaibur_Tasks\Task_4\my-app> npm start

> my-app@0.1.0 start
> react-scripts start
  Local:            http://localhost:3000
  On Your Network:  http://192.168.0.100:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

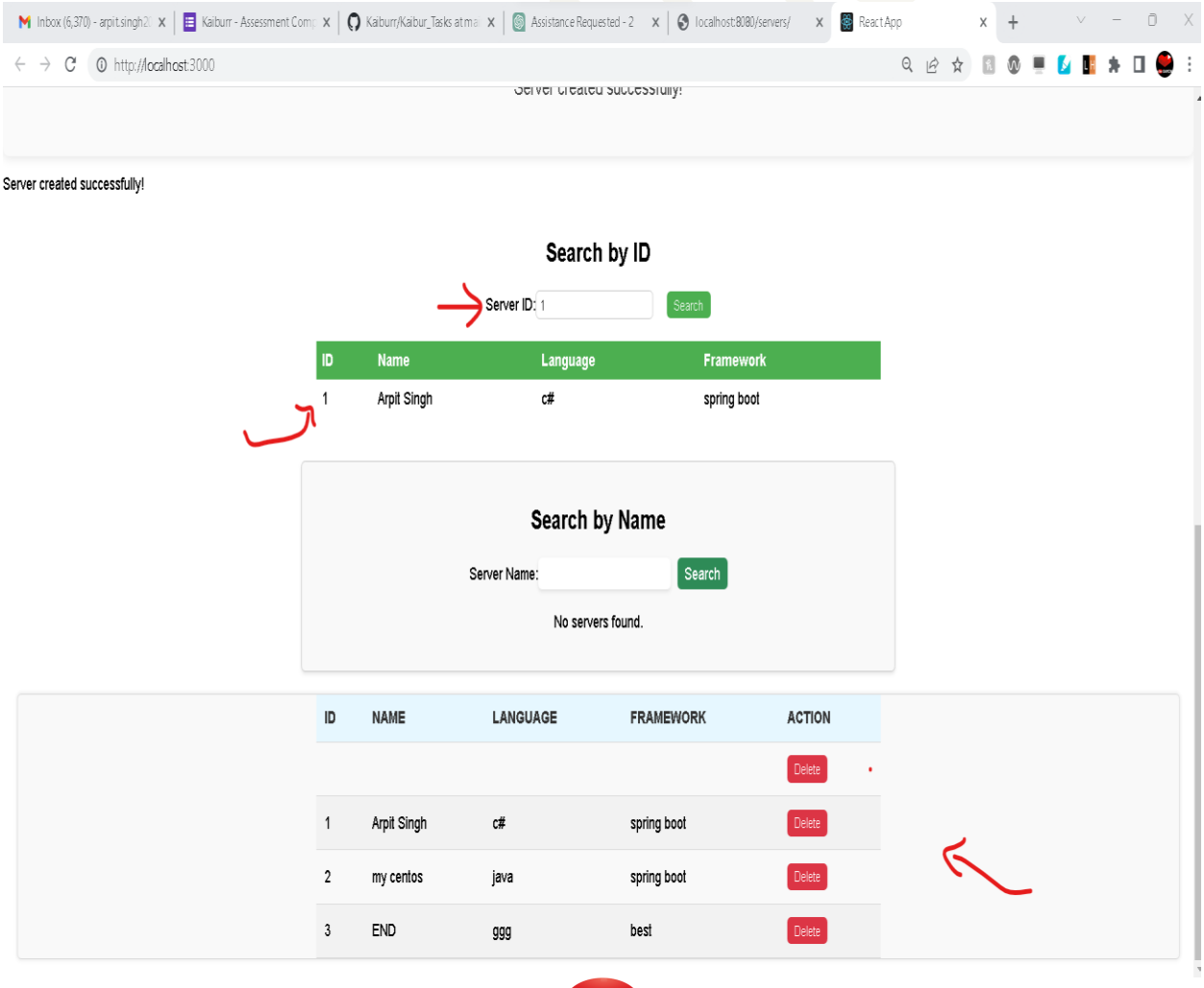
5. UI Testing & Output:

a. *Create Server:*

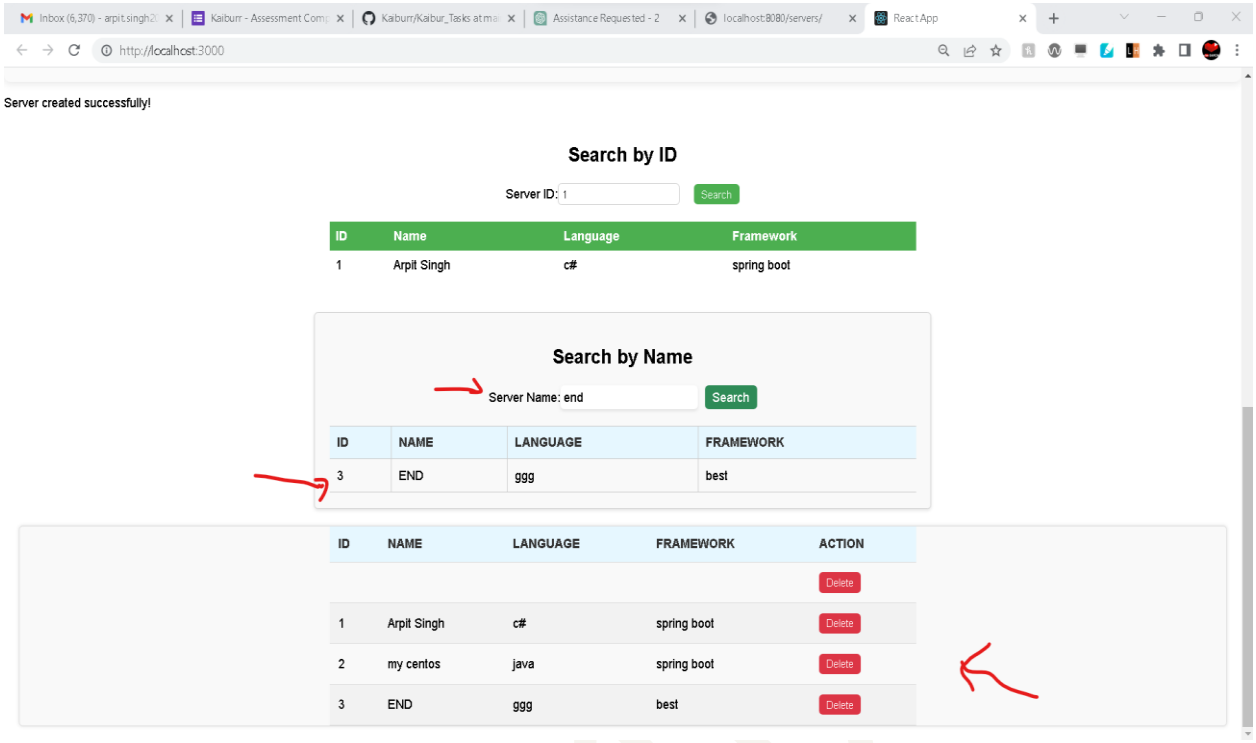




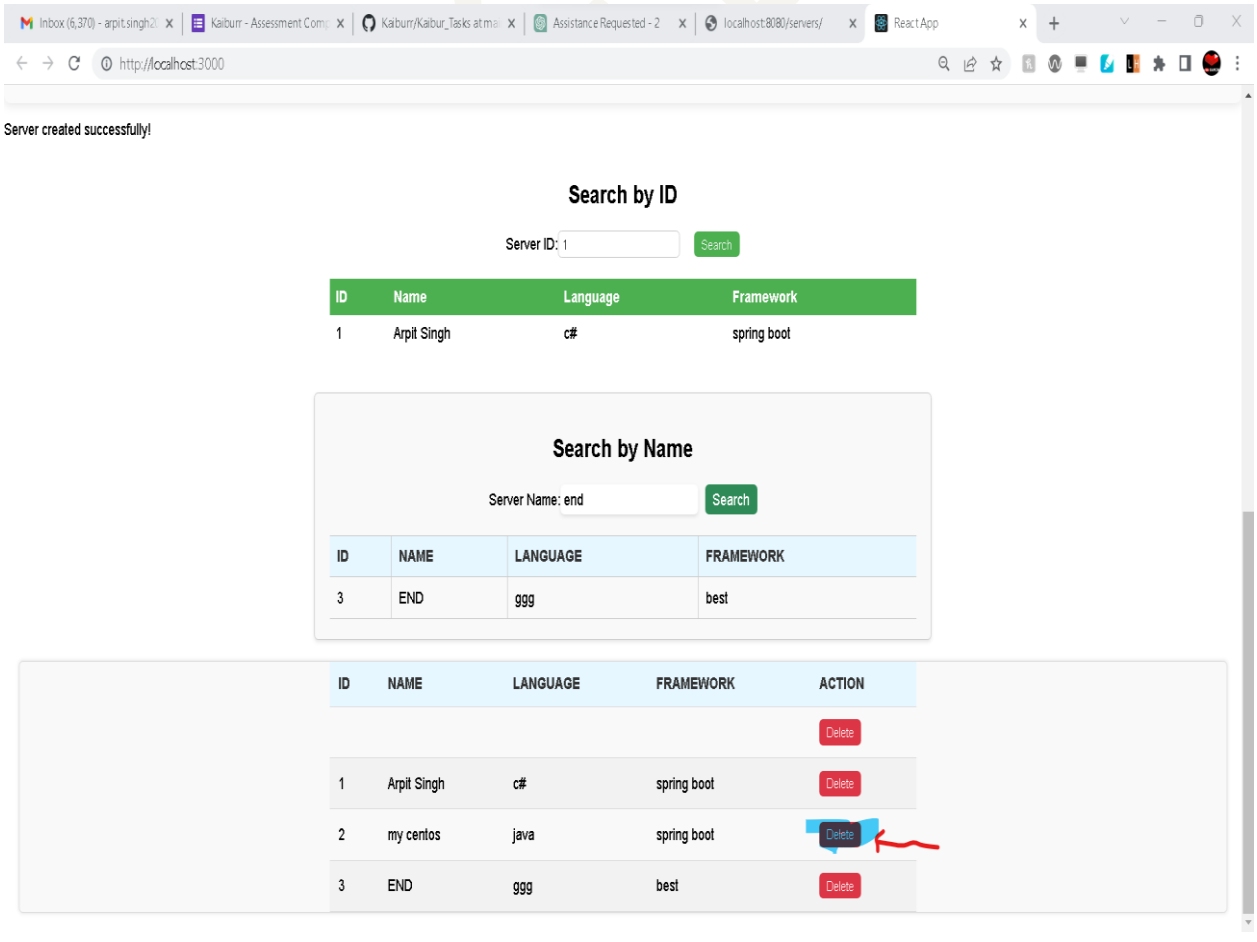
b. ***Search By Id:***

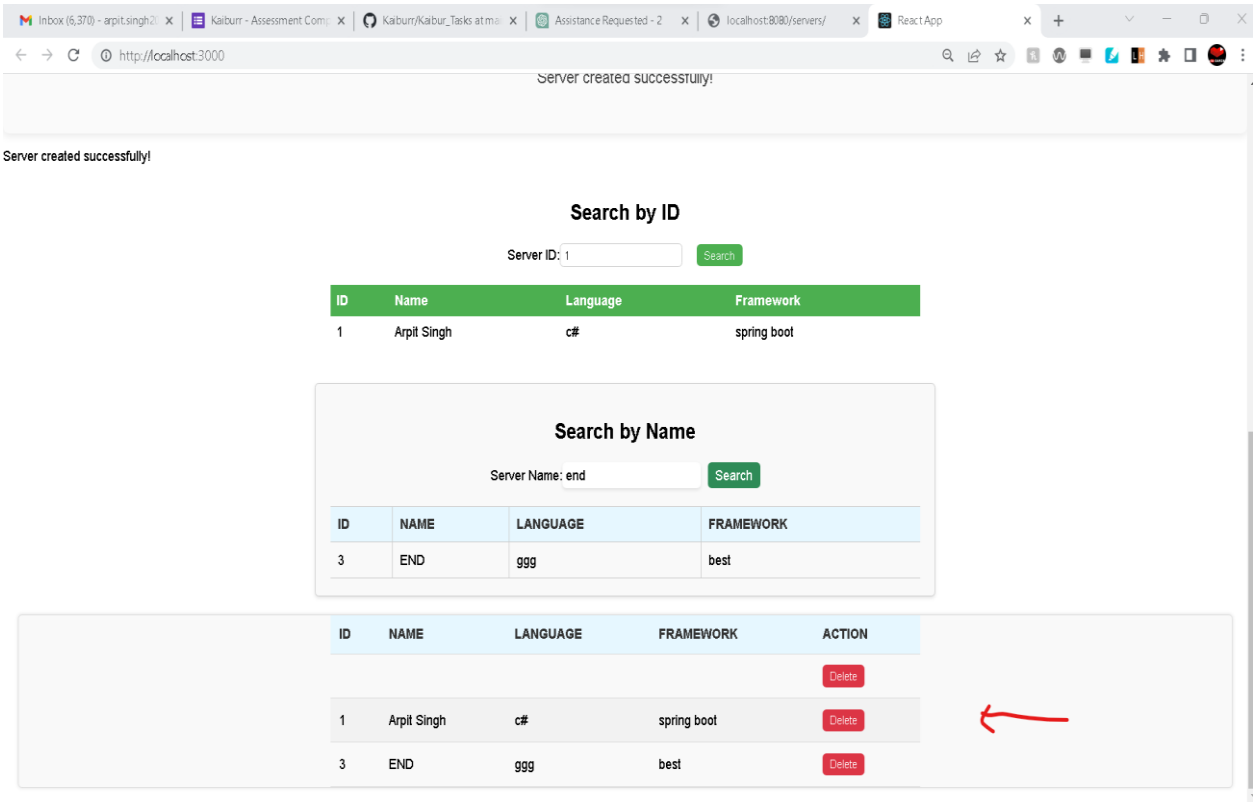


c. ***Search By Name:***



d. ***Delete:***





Summary of the steps to create a UI for REST API (Task 4):

1. Choose a UI framework (e.g. ReactJS).
2. Set up a new React project using create-react-app.
3. Install necessary packages, such as axios for HTTP requests and react-router-dom for client-side routing.
4. Create UI components, such as a form component to create new records, a table component to display records, and a delete button component to delete records.
5. Make API requests using axios to interact with the REST API.
6. Implement client-side routing using react-router-dom to navigate between different pages in the application.
7. Test the application by running **npm start** in the terminal to start the development server and open the application in a web browser.

Explanation on Tools/Technology(s) Used:

- **NodeJs**:
  a. Node.js is used in ***this project*** as a runtime environment for building the backend of the REST API. Node.js allows developers to use JavaScript to write server-side code, which can handle incoming HTTP requests, interact with databases, and respond to those requests with data or HTML pages. In this case, Node.js is used to create a RESTful API that provides access to the application's data, which can be consumed by the frontend application built with React. Node.js provides a lightweight and efficient way to build scalable and fast server-side applications, making it an ideal choice for building the backend of a REST API

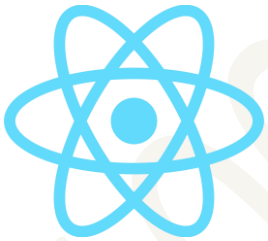
.

- **Npm**:
  a. In ***this project***, npm (Node Package Manager) is used to manage and install the dependencies required for the project. Dependencies are external packages or libraries that are required to run the project. npm allows developers to easily download and manage these dependencies, which can include things like React, Axios, and React Router.

  b. By using npm, developers can simply specify the dependencies they need in the package.json file, and then run a single command (npm install) to automatically download and install all the required packages. This makes it much easier to get started with a new project, since you don't need to manually download and configure all the dependencies yourself.

❖ **ReactJS**:

a. ReactJS is a popular front-end library used for building user interfaces. In ***this project***, ReactJS is used to create the user interface components of the web application that interacts with the REST API. With React, developers can create reusable UI components that can be used across multiple pages and can easily handle the state of the application. React also allows for easy implementation of client-side routing, which enables the user to navigate between different pages of the application without a full page refresh. ReactJS provides an efficient and easy-to-use way to build complex and responsive web applications with minimal code.



Thank You!
