

**Batch: C1**

**Roll No.: 16010122221**

**Experiment / assignment / tutorial No.4**

**Grade: AA / AB / BB / BC / CC / CD / DD**

**Signature of the Staff In-charge with date**

**Title: Demonstrate axios to Create Mock API Server**

**AIM:** To Implement the React Axios

**Problem Definition:**

Build a React application that interacts with a RESTful API using Axios to perform CRUD (Create, Read, Update, Delete) operations. The application should allow users to view, add, update, and delete data from the server. The application should allow users to view, add, update, and delete student data, with smooth navigation between different views using the `useNavigate` hook.

**Requirements:**

- Create a new React application using create-react-app.
- Install Axios using `npm install axios`.
- Install react-router-dom to handle navigation (`npm install react-router-dom`).

**Data Fetching:**

Create a component (StudentList.js) that fetches a list of students from a RESTful API endpoint (e.g., `https://api.example.com/students`) and displays them in a table or list. Handle loading states and errors during the fetch process.

### **Adding a New Student:**

- Implement a form component (AddStudent.js) that allows users to add a new student record.
- Use Axios to send a POST request to the API with the new student data.
- Upon successful submission, navigate the user back to the student list view using useNavigate and display the newly added student in the list.

### **Updating Student Data:**

- Implement an edit functionality in a separate component (EditStudent.js) that allows users to update an existing student's information.
- Use Axios to send a PUT request to the API with the updated student data.
- Upon successful submission, navigate the user back to the student list view using useNavigate, and reflect the updated student information in the list.

### **Deleting a Student:**

- Add a delete button next to each student in the list.
- When the delete button is clicked, use Axios to send a DELETE request to the API.
- Upon successful deletion, the student should be removed from the list without requiring a page reload.

### **Navigation:**

- Use useNavigate to smoothly navigate between different components/views (StudentList, AddStudent, EditStudent).
- Ensure that the browser's back and forward buttons work correctly to navigate between the views.

### Resources used:

For this experiment, several key resources were utilized to build a React application that interacts with a mock API server using Axios. The following tools and technologies were employed:

1. **React:** A JavaScript library for building user interfaces, used here to create the various components (StudentList, AddStudent, EditStudent) for the CRUD operations.
2. **Axios:** A promise-based HTTP client for making API requests, used for data fetching (GET), submission (POST), update (PUT), and deletion (DELETE) of student records.
3. **React Router:** Specifically, `react-router-dom` was used for handling the routing and navigation within the app, enabling smooth transitions between the views.
4. **create-react-app:** A tool that allows for quick setup of a React project without configuration, helping kickstart the development process.

---

### Expected OUTCOME of Experiment:

**CO 2:** Illustrate the concepts of various front-end, back-end web application development technologies & frameworks using different web development tools.

---

### Books/ Journals/ Websites referred:

1. Shelly Powers Learning Node O' Reilly 2 nd Edition, 2016.
2. Ethan Brown, Learning JavaScript: JavaScript Essentials for Modern Application Development, O'Reilly Media, 3rd Edition, 2016
3. **Axios Documentation:** Official documentation for Axios, explaining how to perform HTTP requests and handle responses in both Node.js and browser environments. Available at: <https://axios-http.com/docs/intro>

### Pre Lab/ Prior Concepts:

#### Write details about the following content

- `useNavigate`  
  
`useNavigate` is a hook provided by React Router for programmatic navigation between different routes in a React application. It allows developers to redirect users to different pages without using anchor tags or causing a page reload. This hook is often used after an action like form submission or data updates to navigate the user to a specific route.
  - It replaces the older `useHistory` hook in React Router v6.
  - It returns a function that can be used to navigate to a new route programmatically.

- It can push a new entry onto the history stack or replace the current one, allowing fine control over navigation behavior.

```
import { useNavigate } from "react-router-dom";
```

- **Axios**

Axios is a popular JavaScript library used to make HTTP requests from both the browser and Node.js environments. It is often used in React applications to fetch data from RESTful APIs and handle CRUD operations (Create, Read, Update, Delete).

- Supports promises for handling asynchronous requests.
- Provides an easy-to-use API for making HTTP requests such as GET, POST, PUT, DELETE.
- Automatically transforms JSON data.
- Allows intercepting requests and responses for advanced features like logging or error handling.
- Supports cancellation of requests, request timeouts, and file uploads.

```
import axios from "axios";
```

- **Routes in React**

Routes in React are part of the **React Router** library, which is used for handling navigation and routing in single-page applications (SPAs). React Router allows you to define different components that render based on the URL path, enabling smooth navigation within the app without reloading the entire page.

- **<Route>**: Defines a route path and the component to render when the path matches.
- **<Routes>**: A wrapper that manages the collection of routes in the app, ensuring that only the first matching route is rendered.
- **Dynamic Routing**: You can pass route parameters in the URL to load specific data, like /student/:id to access a particular student's details.
- **Linking**: The <Link> component allows you to navigate between routes without reloading the page.

```
import { BrowserRouter as Router, Routes, Route } from "react-router-dom";
```

## Methodology:

### 1. Setup and Installation:

- Begin by creating a new React project using create-react-app:

```
bash
npx create-react-app student-management
```

### 2. Project Structure:

- Organize the project by creating components for handling CRUD operations. For instance:
- Each component will handle a specific function: adding, editing, displaying, and deleting student data.

### 3. Student List (Read Operation):

- Create the StudentList.js component to display a list of students by fetching data from a mock API using Axios.
- Use useEffect to trigger the Axios GET request when the component loads.
- Example of fetching students:

```
js
Copy code
useEffect(() => {
  axios
    .get('https://jsonplaceholder.typicode.com/users')
    .then((response) => setStudents(response.data))
    .catch((error) => console.error(error));
}, []);
```

### 4. Add a New Student (Create Operation):

- Create the AddStudent.js component with a form to input student details.
- Use the Axios POST method to send form data to the mock API.
- Example of submitting form data:

```
js
Copy code
const handleSubmit = (e) => {
  e.preventDefault();
  axios
    .post('https://jsonplaceholder.typicode.com/users', { name, email })
    .then(() => navigate('/'))
```

```
.catch((error) => console.error(error));  
};
```

#### 5. Edit Student Details (Update Operation):

- Create the EditStudent.js component to edit existing student details.
- Fetch the current student details using Axios GET and update the data using Axios PUT.
- Use the useParams hook to get the student ID from the URL for fetching the specific student record:

```
js  
Copy code  
const { id } = useParams();  
useEffect(() => {  
  axios.get(`https://jsonplaceholder.typicode.com/users/${id}`).then(...)  
}, [id]);
```

#### 6. Delete a Student (Delete Operation):

- Add a delete button next to each student in the StudentList.js component.
- When the button is clicked, use the Axios DELETE method to remove the student:

```
js  
Copy code  
const handleDelete = (id) => {  
  axios.delete(`https://jsonplaceholder.typicode.com/users/${id}`).then(...);  
};
```

#### 7. Routing and Navigation:

- Use **React Router** for smooth navigation between different views.
- Define routes for displaying the list, adding a student, editing a student, etc. in App.js:

```
js  
Copy code  
<Routes>  
  <Route path="/" element={<StudentList />} />  
  <Route path="/add" element={<AddStudent />} />  
  <Route path="/edit/:id" element={<EditStudent />} />  
</Routes>
```

- Use the useNavigate hook to navigate users to different views after successful actions like adding or editing a student.

#### 8. Testing the Application:

- Test each CRUD operation to ensure that the data is fetched, added, updated, and deleted correctly using Axios.

- Verify that the navigation between components is smooth and functional with React Router.

#### 9. Deployment:

- Once the application is fully tested, it can be deployed to a hosting platform like Netlify, Vercel, or GitHub Pages for easy access and demonstration.

### Implementation Details:

#### App.jsx

```
import './App.css';
import React from 'react';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import AddStudent from './components/AddStudent';
import StudentList from './components/StudentList';
import EditStudent from './components/EditStudent';

function App() {
  return (
    <div className="App">
      <Router>
        <Routes>
          <Route path="/" element={<AddStudent />} />
          <Route path="/students" element={<StudentList />} />
          <Route path="/edit/:id" element={<EditStudent />} />
        </Routes>
      </Router>
    </div>
  );
}

export default App;
```



### AddStudent.jsx

```
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import '../App.css';

export default function AddStudent() {
  const [name, setName] = useState("");
  const [email, setEmail] = useState("");
  const [sid, setSid] = useState("");
  const [students, setStudents] = useState([]);
  const [notification, setNotification] = useState("");

  const navigate = useNavigate();

  const handleAddStudent = (e) => {
    e.preventDefault();
    const newStudent = { name, email, sid };
    setStudents([...students, newStudent]);
    setName("");
    setEmail("");
    setSid("");
    setNotification("Student added successfully!");
    setTimeout(() => setNotification(""), 3000);  };

  const handleShowStudentList = () => {
    navigate('/students', { state: { students } });
  };

  return (
    <div className="form-container">
      <h2 className="form-heading">Add Student</h2>
      {notification} && <div className="notification">{notification}</div>
      <form onSubmit={handleAddStudent}>
        <input
          type="text"
          value={name}
          onChange={e => setName(e.target.value)}
          placeholder="Name"
          required
          className="input-field">
```





```
    />
    <input
      type="email"
      value={email}
      onChange={e => setEmail(e.target.value)}
      placeholder="Email"
      required
      className="input-field"
    />
    <input
      type="text"
      value={sid}
      onChange={e => setSid(e.target.value)}
      placeholder="SID"
      required
      className="input-field"
    />
    <button type="submit" className="submit-btn">Add Student</button>
  </form>
  <button onClick={handleShowStudentList} className="submit-btn">Show Student
List</button>
</div>
);
}
```



### StudentList.jsx

```
import React from 'react';
import { useLocation } from 'react-router-dom';
import '../App.css';

export default function StudentList() {
  const location = useLocation();
  const students = location.state?.students || [];

  return (
    <div className="student-list">
      <h2>Student List</h2>
      {students.length > 0 ? (
        <ul>
          {students.map((student, index) => (
            <li key={index} className="student-item">
              <p><strong>Name:</strong> {student.name}</p>
              <p><strong>Email ID:</strong> {student.email}</p>
              <p><strong>SID:</strong> {student.sid}</p>
            </li>
          ))}
        </ul>
      ) : (
        <p>No student details available.</p>
      )}
    </div>
  );
}
```



### EditStudent.jsx

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { useParams, useNavigate } from 'react-router-dom';
import '../App.css';

const EditStudent = () => {
  const { id } = useParams();
  const navigate = useNavigate();
  const [name, setName] = useState("");
  const [email, setEmail] = useState("");

  useEffect(() => {
    axios.get(`https://jsonplaceholder.typicode.com/posts/${id}`)
      .then(response => {
        setName(response.data.title);
        setEmail(response.data.body);
      })
      .catch(error => {
        console.error("Error fetching student:", error);
      });
  }, [id]);

  const handleSubmit = (e) => {
    e.preventDefault();
    axios.put(`https://jsonplaceholder.typicode.com/posts/${id}`, { name, email })
      .then(response => {
        console.log("Student updated successfully", response);
        navigate('/');
      })
      .catch(error => {
        console.error("Error updating student:", error);
      });
  };

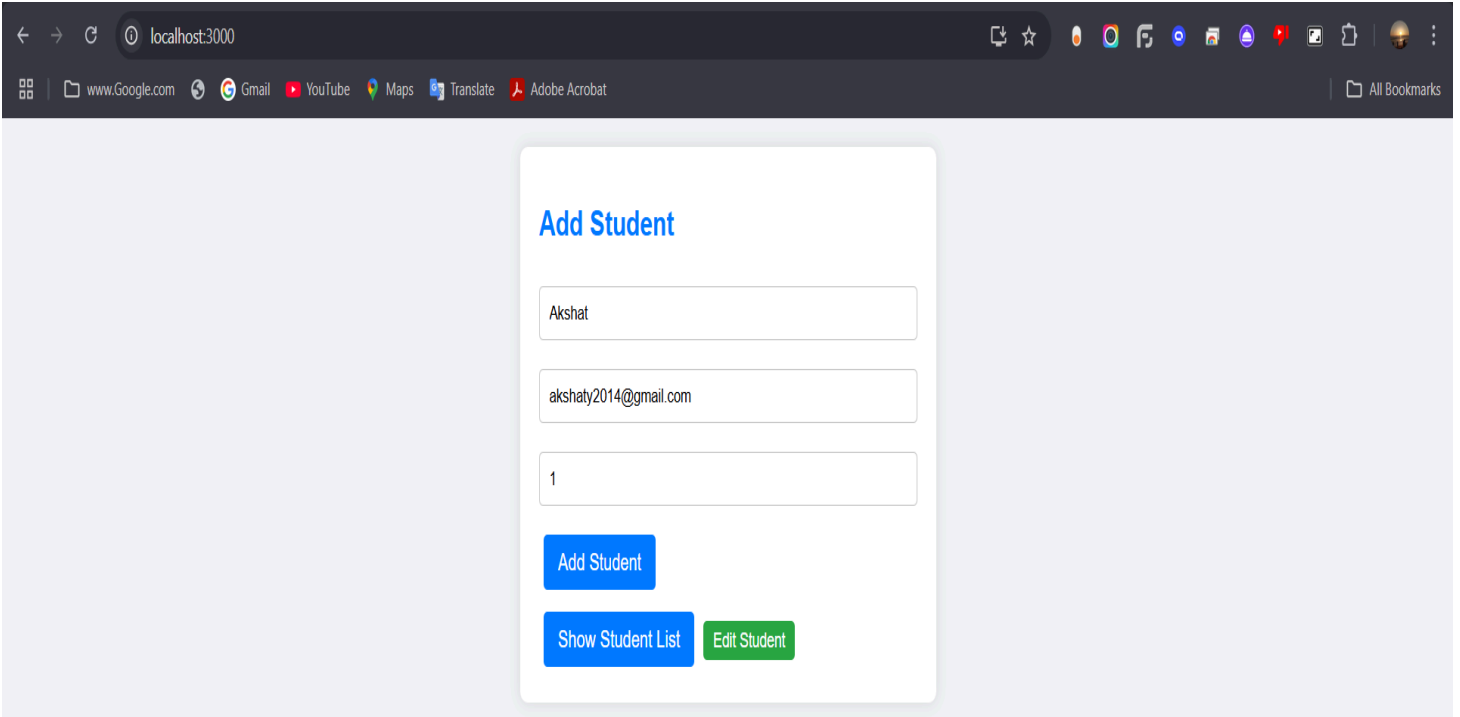
  return (
    <form onSubmit={handleSubmit} className="edit-student-container">
      <h2>Edit Student</h2>
    </form>
  );
};
```



```
<input
  type="text"
  value={name}
  onChange={e => setName(e.target.value)}
  placeholder="Name"
  required
  className="input-field"
/>
<input
  type="email"
  value={email}
  onChange={e => setEmail(e.target.value)}
  placeholder="Email"
  required
  className="input-field"
/>
<button type="submit" className="submit-btn">Update Student</button>
</form>
);
};

export default EditStudent;
```

## Screenshots-



A screenshot of a web browser showing a form titled "Add Student". The form has three input fields: the first contains "Akshat", the second contains "akshaty2014@gmail.com", and the third contains "1". Below the input fields are three buttons: a blue "Add Student" button, a blue "Show Student List" button, and a green "Edit Student" button. The browser's address bar shows "localhost:3000".

**Add Student**

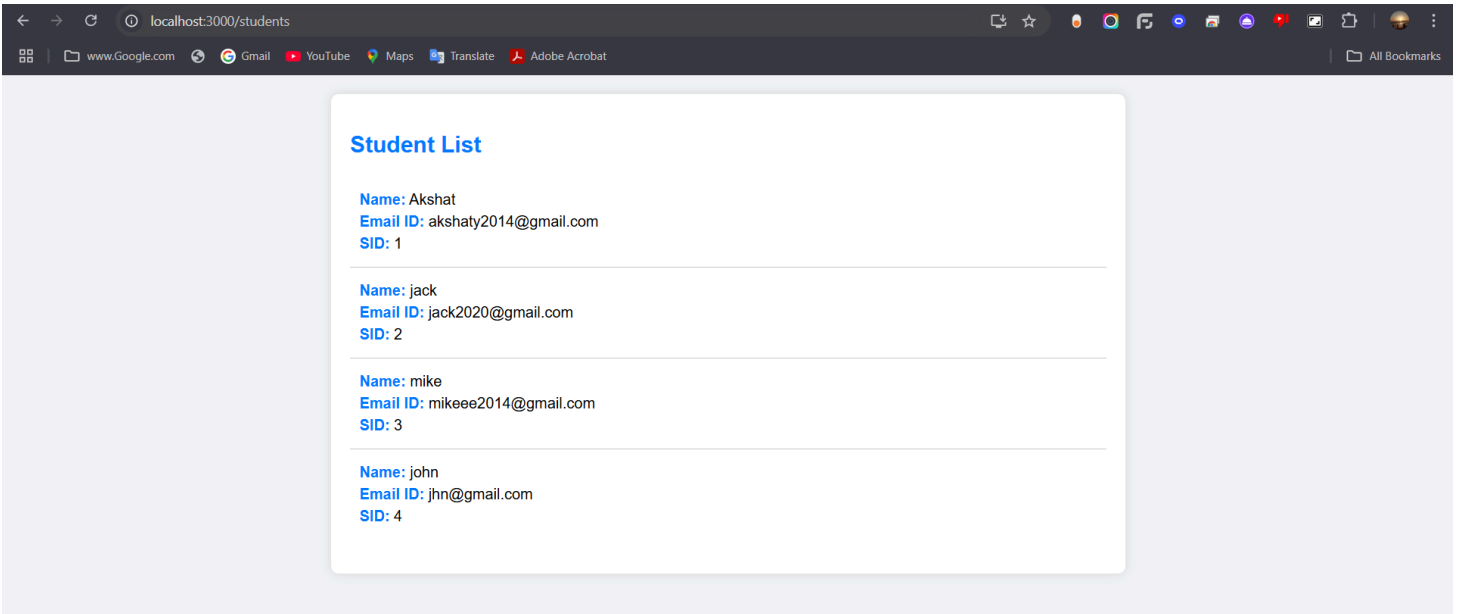
Akshat

akshaty2014@gmail.com

1

Add Student





Show Student List Edit Student











A screenshot of a web browser showing a page titled "Student List". The page displays a list of four students, each with their Name, Email ID, and SID. The browser's address bar shows "localhost:3000/students".

**Student List**

<b>Name:</b> Akshat
<b>Email ID:</b> akshaty2014@gmail.com
<b>SID:</b> 1
<b>Name:</b> jack
<b>Email ID:</b> jack2020@gmail.com
<b>SID:</b> 2
<b>Name:</b> mike
<b>Email ID:</b> mikeee2014@gmail.com
<b>SID:</b> 3
<b>Name:</b> john
<b>Email ID:</b> jhn@gmail.com
<b>SID:</b> 4

    localhost:3000/edit/1

 |  www.Google.com   Gmail  YouTube  Maps  Translate  Adobe Acrobat

## Edit Student

### **Conclusion:**

In conclusion, the implementation of a React application utilizing Axios for CRUD operations, along with React Router for seamless navigation, effectively demonstrates the integration of front-end and back-end technologies in web development.

### **Postlab questions:**

1) Different ways to Add Api in React/Javascript with example.

### **Using Fetch API**

The Fetch API is a built-in JavaScript function for making HTTP requests. It returns a Promise that resolves to the response of the request.

### **Example:**

```
javascript  
Copy code  
import React, { useEffect, useState } from 'react';
```

```
const FetchExample = () => {  
  const [data, setData] = useState([]);  
  
  useEffect(() => {  
    fetch('https://jsonplaceholder.typicode.com/users')  
      .then((response) => response.json())  
      .then((data) => setData(data))  
      .catch((error) => console.error('Error fetching data:', error));  
  }, []);  
  
  return (  
    <div>  
      <h1>User List</h1>  
      <ul>  
        {data.map((user) => (  
          <li key={user.id}>{user.name}</li>  
        ))}  
      </ul>  
    </div>  
  );  
};  
  
export default FetchExample;
```

## 2. Using Axios

Axios is a promise-based HTTP client for JavaScript that is often used in React applications for making API calls.

### Example:

```
javascript  
Copy code  
import React, { useEffect, useState } from 'react';  
import axios from 'axios';  
  
const AxiosExample = () => {  
  const [data, setData] = useState([]);  
  
  useEffect(() => {  
    axios.get('https://jsonplaceholder.typicode.com/users')  
      .then((response) => setData(response.data))  
      .catch((error) => console.error('Error fetching data:', error));  
  });  
};
```



```
}, []);  
  
return (  
  <div>  
    <h1>User List</h1>  
    <ul>  
      {data.map((user) => (  
        <li key={user.id}>{user.name}</li>  
      ))}  
    </ul>  
  </div>  
)  
};
```

```
export default AxiosExample;
```

### 3. Using Async/Await with Fetch

Using async/await syntax can make your code cleaner and easier to read when dealing with asynchronous code.

```
import React, { useEffect, useState } from 'react';  
  
const AsyncAwaitExample = () => {  
  const [data, setData] = useState([]);  
  
  const fetchData = async () => {  
    try {  
      const response = await fetch('https://jsonplaceholder.typicode.com/users');  
      const result = await response.json();  
      setData(result);  
    } catch (error) {  
      console.error('Error fetching data:', error);  
    }  
  }  
  
  useEffect(() => {  
    fetchData();  
  }, []);  
  
  return (  
    <div>  
      <h1>User List</h1>
```

```
<ul>
  {data.map((user) => (
    <li key={user.id}>{user.name}</li>
  ))}
</ul>
</div>
);
};
```

```
export default AsyncAwaitExample;
```

#### 4. Using React Query

React Query is a powerful data-fetching library that simplifies data fetching and caching in React applications.

##### Example:

```
import React from 'react';
import { useQuery } from 'react-query';
import axios from 'axios';

const ReactQueryExample = () => {
  const { data, error, isLoading } = useQuery('users', () =>
    axios.get('https://jsonplaceholder.typicode.com/users').then(res => res.data)
  );

  if (isLoading) return <div>Loading...</div>;
  if (error) return <div>Error fetching data</div>;

  return (
    <div>
      <h1>User List</h1>
      <ul>
        {data.map((user) => (
          <li key={user.id}>{user.name}</li>
        ))}
      </ul>
    </div>
  );
};

export default ReactQueryExample;
```

## 5. Using Redux Middleware (Redux Thunk)

If you are using Redux for state management, you can use middleware like Redux Thunk to handle API calls.

### Example:

```
import axios from 'axios';

export const fetchUsers = () => {
  return (dispatch) => {
    dispatch({ type: 'FETCH_USERS_REQUEST' });
    axios.get('https://jsonplaceholder.typicode.com/users')
      .then(response => {
        dispatch({ type: 'FETCH_USERS_SUCCESS', payload: response.data });
      })
      .catch(error => {
        dispatch({ type: 'FETCH_USERS_FAILURE', payload: error.message });
      });
  };
};

import React, { useEffect } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import { fetchUsers } from './actions';

const UsersComponent = () => {
  const dispatch = useDispatch();
  const users = useSelector((state) => state.users);
  const loading = useSelector((state) => state.loading);
  const error = useSelector((state) => state.error);

  useEffect(() => {
    dispatch(fetchUsers());
  }, [dispatch]);

  if (loading) return <div>Loading...</div>;
  if (error) return <div>Error fetching data</div>;

  return (
    <div>
      <h1>User List</h1>
      <ul>
        {users.map((user) => (
```



```
<li key={user.id}>{user.name}</li>
    )}
  </ul>
</div>
);
};

export default UsersComponent;
```