**Batch: A-3 Roll No.: 16010122104**

**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:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**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.

**Pre Lab/ Prior Concepts:**

 **useNavigate**: useNavigate is a hook provided by React Router that allows developers to programmatically navigate between routes in a React application. It replaces the older useHistory hook from React Router v5. With useNavigate, developers can navigate to different routes based on user actions or certain conditions, such as after form submission or on successful login. It provides a navigate function, which accepts a path and an optional configuration object for things like state or navigation options.

 **Axios**: Axios is a popular JavaScript library used to make HTTP requests from both the browser and Node.js. It simplifies the process of sending asynchronous HTTP requests, such as GET, POST, PUT, and DELETE, and handles responses easily. Axios automatically transforms response data into JSON and supports features like request and response interceptors, cancellation of requests, and handling timeouts. It is commonly used in React applications for making API calls, managing data, and handling errors.

 **Routes in React**: Routes in React, powered by React Router, are used to define and manage the navigation between different views or pages in a React application. With React Router, developers can create dynamic routes that map a URL to a specific component. The Route component is used to declare these mappings, and the Routes component wraps all of the Route definitions to manage which component should render based on the current URL. React Router also provides other features like nested routes, route parameters, and redirecting to other routes.

**Methodology:**

{

  "name": "admin",

  "private": true,

  "version": "0.0.0",

  "type": "module",

  "scripts": {

    "dev": "vite",

    "build": "vite build",

    "lint": "eslint . --ext js,jsx --report-unused-disable-directives --max-warnings 0",

    "preview": "vite preview"

  },

  "dependencies": {

    "axios": "^1.7.4",

    "react": "^18.3.1",

    "react-dom": "^18.3.1",

    "react-router-dom": "^6.26.1",

    "react-toastify": "^10.0.5"

  },

  "devDependencies": {

    "@types/react": "^18.3.3",

    "@types/react-dom": "^18.3.0",

    "@vitejs/plugin-react": "^4.3.1",

    "autoprefixer": "^10.4.20",

    "eslint": "^8.57.0",

    "eslint-plugin-react": "^7.35.0",

    "eslint-plugin-react-hooks": "^4.6.2",

    "eslint-plugin-react-refresh": "^0.4.9",

    "postcss": "^8.4.41",

    "tailwindcss": "^3.4.10",

    "vite": "^5.4.1"

  }

}

**Implementation Details:**

 **Project Initialization**

* Start by creating a new project directory.
* In your terminal, navigate to the project directory and initialize a new Vite + React app:

bash

Copy code

npm create vite@latest

* Follow the prompts to name your project and select React as the framework.

 **Install Dependencies**

* After setting up Vite, install the dependencies listed in package.json:

bash

Copy code

npm install axios react react-dom react-router-dom react-toastify

* Install the development dependencies for ESLint, Tailwind CSS, and other tooling:

bash

Copy code

npm install -D @types/react @types/react-dom @vitejs/plugin-react autoprefixer eslint eslint-plugin-react eslint-plugin-react-hooks eslint-plugin-react-refresh postcss tailwindcss vite

 **Vite Configuration**

* In your project root, create a Vite configuration file (vite.config.js) and add @vitejs/plugin-react:

javascript

Copy code

import { defineConfig } from 'vite';

import react from '@vitejs/plugin-react';

export default defineConfig({

plugins: [react()],

});

 **Tailwind CSS Configuration**

* Initialize Tailwind CSS in your project:

bash

Copy code

npx tailwindcss init -p

* This will generate tailwind.config.js and postcss.config.js.
* Configure the Tailwind CSS content paths in tailwind.config.js:

javascript

Copy code

module.exports = {

content: [

'./index.html',

'./src/\*\*/\*.{js,jsx,ts,tsx}'

],

theme: {

extend: {},

},

plugins: [],

};

 **Set Up Tailwind in CSS**

* In your main CSS file (typically src/index.css), import Tailwind’s base, components, and utilities:

css

Copy code

@tailwind base;

@tailwind components;

@tailwind utilities;

 **Configure ESLint**

* Set up an .eslintrc file for ESLint configuration in the root of your project:

json

Copy code

{

"extends": [

"eslint:recommended",

"plugin:react/recommended"

],

"plugins": [

"react",

"react-hooks"

],

"settings": {

"react": {

"version": "detect"

}

},

"rules": {

"react/react-in-jsx-scope": "off"

},

"env": {

"browser": true,

"node": true

}

}

 **Set Up React Toastify and Routing**

* Import react-toastify styles in your src/index.js or src/App.js:

javascript

Copy code

import 'react-toastify/dist/ReactToastify.css';

* Set up routing with react-router-dom in src/App.js:

javascript

Copy code

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

import Home from './pages/Home';

import About from './pages/About';

function App() {

return (

<Router>

<Routes>

<Route path="/" element={<Home />} />

<Route path="/about" element={<About />} />

</Routes>

</Router>

);

}

export default App;

**Steps for execution:**

 **Run Development Server**

* Start the development server with the command:

bash

Copy code

npm run dev

* This will run Vite and serve the application locally, usually on http://localhost:5173.

 **Build for Production**

* To build the application for production, run:

bash

Copy code

npm run build

* Vite will output production-ready files to the dist folder.

 **Preview Production Build**

* After building, preview the production build locally by running:

bash

Copy code

npm run preview

* This will simulate serving the application as it would appear in production.

 **Linting**

* Run ESLint to check for any code issues:

bash

Copy code

npm run lint

* ESLint will scan your project files and report any issues based on the configuration.

 **Test Toast Notifications and Routing**

* Use react-toastify to display notifications by calling toast in your components, and ensure routing is working by navigating to defined paths like /about in the app.

**Conclusion: We implemented a mock api for our project.**

**Postlab questions:**

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

Ans:

 **fetch API**: Simple and built-in JavaScript method for making HTTP requests. Suitable for basic use cases.

 **Axios**: A third-party library for making HTTP requests, providing extra features like interceptors, better error handling, and automatic JSON transformation.

 **async/await with fetch**: Allows cleaner and more readable asynchronous code using promises.

 **XMLHttpRequest**: The older, more verbose way of making API requests.

 **useEffect in React**: Used for performing side-effects, like fetching data from APIs when the component mounts.