Batch: A1	**Roll No.: 16010123012**
Experiment / assignment / tutorial No.: 06	
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.

Resources used:

- React.js (create-react-app)
- Axios (for HTTP requests)
- React Router DOM (for navigation)
- MockAPI (https://mockapi.io) as backend server

Expected OUTCOME of Experiment:

CO2: 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:

Write details about the following content

- useNavigate - Used to programmatically navigate between routes.

```
const navigate = useNavigate();
navigate('/add');
```

- Axios - Promise-based HTTP client for React and Node.js.

Simplifies GET, POST, PUT, DELETE requests, Supports interceptors for logging or adding headers.

- Routes in React - Define navigation paths to render different components.

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

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.

Methodology:

1. Created React app with create-react-app.
2. Installed Axios and React Router DOM dependencies.
3. Defined studentService.js for API requests (getAllStudents, createStudent, updateStudent, deleteStudent).
4. Implemented StudentList.js to fetch and display students using Axios GET.
5. Implemented AddStudent.js with a form to send POST request.
6. Implemented EditStudent.js with form to update student using PUT request.
7. Added Delete button in StudentList calling DELETE API.
8. Added routes in App.js for navigation between views.

Implementation Details:

1. App.js – Sets up routes for StudentList, AddStudent, EditStudent
2. studentService.js – Centralized Axios service with interceptors & CRUD functions
3. StudentList.js – Fetches student list, displays table, and allows delete/edit
4. AddStudent.js – Form to add new student (POST request).
5. EditStudent.js – Form to update student (PUT request).
6. useNavigate – Used for smooth navigation after add/edit/delete.

App.js

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

```
import StudentList from './components/StudentList';
import AddStudent from './components/AddStudent';
import EditStudent from './components/EditStudent';

function App() {
  return (
    <Router>
      <div className="App">
        <nav className="navbar">
          <h1 style={{ textAlign: 'center', margin: '20px 0' }}>Student Management
System</h1>
        </nav>

        <main className="main-content">
          <Routes>
            <Route path="/" element={<StudentList />} />
            <Route path="/add" element={<AddStudent />} />
            <Route path="/edit/:id" element={<EditStudent />} />
          </Routes>
        </main>
      </div>
    </Router>
  );
}

export default App;
```

studentService.js

```
import axios from 'axios';
const API_BASE_URL = 'https://68beac919c70953d96ed2874.mockapi.io/students';

const apiClient = axios.create({
  baseURL: API_BASE_URL,
  timeout: 10000,
  headers: {
    'Content-Type': 'application/json',
  },
});

apiClient.interceptors.response.use(
  (response) => response,
  (error) => {
    console.error('API Error:', error.response?.data || error.message);
```

```
    return Promise.reject(error);
  }
);

// API Service
export const studentService = {
  getAllStudents: async () => {
    try {
      const response = await apiClient.get('/');
      return response.data;
    } catch (error) {
      throw new Error(`Failed to fetch students: ${error.message}`);
    }
  },

  getStudentById: async (id) => {
    try {
      const response = await apiClient.get(`/${id}`);
      return response.data;
    } catch (error) {
      throw new Error(`Failed to fetch student with ID ${id}: ${error.message}`);
    }
  },

  createStudent: async (studentData) => {
    try {
      const apiData = {
        name: studentData.name,
        email: studentData.email,
        rollNo: studentData.rollNo
      };

      const response = await apiClient.post('/', apiData);
      return response.data;
    } catch (error) {
      throw new Error(`Failed to create student: ${error.message}`);
    }
  },

  updateStudent: async (id, studentData) => {
    try {
      const apiData = {
        name: studentData.name,
```

```
        email: studentData.email,
        rollNo: studentData.rollNo
      };

      const response = await apiClient.put(`/${id}`, apiData);
      return response.data;
    } catch (error) {
      throw new Error(`Failed to update student with ID ${id}: ${error.message}`);
    }
  },

  deleteStudent: async (id) => {
    try {
      await apiClient.delete(`/${id}`);
      return { success: true, message: 'Student deleted successfully' };
    } catch (error) {
      throw new Error(`Failed to delete student with ID ${id}: ${error.message}`);
    }
  },
};

export const {
  getAllStudents,
  getStudentById,
  createStudent,
  updateStudent,
  deleteStudent
} = studentService;

export { apiClient };
export default studentService;
```

StudentList.js

```
import React, { useState, useEffect } from 'react';
import { useNavigate } from 'react-router-dom';
import { studentService } from '../services/studentService';
import './StudentList.css';

const StudentList = () => {
  const [students, setStudents] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  const navigate = useNavigate();
```

```
useEffect(() => {
  fetchStudents();
}, []);

const fetchStudents = async () => {
  try {
    setLoading(true);
    setError(null);
    const studentsData = await studentService.getAllStudents();
    setStudents(studentsData);
  } catch (err) {
    setError(err.message || 'Failed to fetch students. Please try again later.');
  } finally {
    setLoading(false);
  }
};

const handleDelete = async (studentId) => {
  if (window.confirm('Are you sure you want to delete this student?')) {
    try {
      await studentService.deleteStudent(studentId);
      setStudents(students.filter(student => student.id !== studentId));
      alert('Student deleted successfully!');
    } catch (err) {
      alert('Failed to delete student. Please try again.');
    }
  }
};

const handleEdit = (studentId) => {
  navigate(`/edit/${studentId}`);
};

if (loading) {
  return (
    <div className="loading-container">
      <div className="loading-spinner"></div>
      <p>Loading students...</p>
    </div>
  );
}

if (error) {
```

```
  return (
    <div className="error-container">
      <h2>Error</h2>
      <p>{error}</p>
      <button onClick={fetchStudents} className="retry-btn">
        Retry
      </button>
    </div>
  );
}

return (
  <div className="student-list-container">
    <div className="header">
      <h2>Student List</h2>
      <button
        onClick={() => navigate('/add')}
        className="add-btn"
      >
        Add New Student
      </button>
    </div>

    {students.length === 0 ? (
      <div className="no-students">
        <p>No students found.</p>
        <button onClick={() => navigate('/add')} className="add-btn">
          Add First Student
        </button>
      </div>
    ) : (
      <div className="table-container">
        <table className="students-table">
          <thead>
            <tr>
              <th>Roll No</th>
              <th>Name</th>
              <th>Email</th>
              <th>Actions</th>
            </tr>
          </thead>
          <tbody>
            {students.map(student => (
```

```
                <tr key={student.id}>
                  <td>{student.rollNo}</td>
                  <td>{student.name}</td>
                  <td>{student.email}</td>
                  <td className="actions">
                    <button
                      onClick={() => handleEdit(student.id)}
                      className="edit-btn"
                    >
                      Edit
                    </button>
                    <button
                      onClick={() => handleDelete(student.id)}
                      className="delete-btn"
                    >
                      Delete
                    </button>
                  </td>
                </tr>
              ))}
            </tbody>
          </table>
        </div>
      )}
    </div>
  );
};

export default StudentList;
```

AddStudent.js

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

const AddStudent = () => {
  const [student, setStudent] = useState({
    name: '',
    email: '',
    rollNo: ''
  });
  const [loading, setLoading] = useState(false);
```

```
const [errors, setErrors] = useState({});
const navigate = useNavigate();

const handleChange = (e) => {
  const { name, value } = e.target;
  setStudent(prev => ({
    ...prev,
    [name]: value
  }));

  if (errors[name]) {
    setErrors(prev => ({
      ...prev,
      [name]: ''
    }));
  }
};

const validateForm = () => {
  const newErrors = {};

  if (!student.name.trim()) {
    newErrors.name = 'Name is required';
  }

  if (!student.email.trim()) {
    newErrors.email = 'Email is required';
  } else if (!/\S+@\S+\.\S+/.test(student.email)) {
    newErrors.email = 'Email is invalid';
  }

  if (!student.rollNo.trim()) {
    newErrors.rollNo = 'Roll Number is required';
  }

  setErrors(newErrors);
  return Object.keys(newErrors).length === 0;
};

const handleSubmit = async (e) => {
  e.preventDefault();

  if (!validateForm()) {
```

```
      return;
    }

    setLoading(true);
    try {
      await studentService.createStudent(student);
      alert('Student added successfully!');
      navigate('/');
    } catch (error) {
      alert(error.message || 'Failed to add student. Please try again.');
    } finally {
      setLoading(false);
    }
  };

  const handleCancel = () => {
    navigate('/');
  };

  return (
    <div className="student-form-container">
      <div className="form-header">
        <h2>Add New Student</h2>
        <button onClick={handleCancel} className="cancel-btn">
          Back to List
        </button>
      </div>

      <form onSubmit={handleSubmit} className="student-form">
        <div className="form-group">
          <label htmlFor="name">Name *</label>
          <input
            type="text"
            id="name"
            name="name"
            value={student.name}
            onChange={handleChange}
            className={errors.name ? 'error' : ''}
            placeholder="Enter student name"
          />
          {errors.name && <span className="error-message">{errors.name}</span>}
        </div>
```

```
      <div className="form-group">
        <label htmlFor="email">Email *</label>
        <input
          type="email"
          id="email"
          name="email"
          value={student.email}
          onChange={handleChange}
          className={errors.email ? 'error' : ''}
          placeholder="Enter email address"
        />
        {errors.email && <span className="error-message">{errors.email}</span>}
      </div>

      <div className="form-group">
        <label htmlFor="rollNo">Roll Number *</label>
        <input
          type="text"
          id="rollNo"
          name="rollNo"
          value={student.rollNo}
          onChange={handleChange}
          className={errors.rollNo ? 'error' : ''}
          placeholder="Enter roll number"
        />
        {errors.rollNo && <span className="error-message">{errors.rollNo}</span>}
      </div>

      <div className="form-actions">
        <button
          type="button"
          onClick={handleCancel}
          className="cancel-btn"
          disabled={loading}
        >
          Cancel
        </button>
        <button
          type="submit"
          className="submit-btn"
          disabled={loading}
        >
          {loading ? 'Adding...' : 'Add Student'}
```

```
          </button>
        </div>
      </form>
    </div>
  );
};

export default AddStudent;
```

EditStudent.js

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

const EditStudent = () => {
  const [student, setStudent] = useState({
    name: '',
    email: '',
    rollNo: ''
  });
  const [loading, setLoading] = useState(false);
  const [fetchLoading, setFetchLoading] = useState(true);
  const [errors, setErrors] = useState({});
  const [fetchError, setFetchError] = useState(null);
  const navigate = useNavigate();
  const { id } = useParams();

  useEffect(() => {
    const fetchStudent = async () => {
      try {
        setFetchLoading(true);
        setFetchError(null);
        const studentData = await studentService.getStudentById(id);

        setStudent({
          name: studentData.name,
          email: studentData.email,
          rollNo: studentData.rollNo
        });
      } catch (error) {
        setFetchError(error.message || 'Failed to fetch student data. Please try
again.');
```

```
      } finally {
        setFetchLoading(false);
      }
    };

    fetchStudent();
  }, [id]);

  const retryFetch = async () => {
    try {
      setFetchLoading(true);
      setFetchError(null);
      const studentData = await studentService.getStudentById(id);

      setStudent({
        name: studentData.name,
        email: studentData.email,
        rollNo: studentData.rollNo
      });
    } catch (error) {
      setFetchError(error.message || 'Failed to fetch student data. Please try
again.');
    } finally {
      setFetchLoading(false);
    }
  };

  const handleChange = (e) => {
    const { name, value } = e.target;
    setStudent(prev => ({
      ...prev,
      [name]: value
    }));
    if (errors[name]) {
      setErrors(prev => ({
        ...prev,
        [name]: ''
      }));
    }
  };

  const validateForm = () => {
    const newErrors = {};
```

```
  if (!student.name.trim()) {
    newErrors.name = 'Name is required';
  }

  if (!student.email.trim()) {
    newErrors.email = 'Email is required';
  } else if (!/\S+@\S+\.\S+/.test(student.email)) {
    newErrors.email = 'Email is invalid';
  }

  if (!student.rollNo.trim()) {
    newErrors.rollNo = 'Roll Number is required';
  }

  setErrors(newErrors);
  return Object.keys(newErrors).length === 0;
};

const handleSubmit = async (e) => {
  e.preventDefault();

  if (!validateForm()) {
    return;
  }

  setLoading(true);
  try {
    const response = await studentService.updateStudent(id, student);
    console.log('Student updated:', response);

    alert('Student updated successfully!');
    navigate('/');
  } catch (error) {
    alert(error.message || 'Failed to update student. Please try again.');
  } finally {
    setLoading(false);
  }
};

const handleCancel = () => {
  navigate('/');
};
```

```
if (fetchLoading) {
  return (
    <div className="loading-container">
      <div className="loading-spinner"></div>
      <p>Loading student data...</p>
    </div>
  );
}

if (fetchError) {
  return (
    <div className="error-container">
      <h2>Error</h2>
      <p>{fetchError}</p>
      <div className="error-actions">
        <button onClick={retryFetch} className="retry-btn">
          Retry
        </button>
        <button onClick={handleCancel} className="cancel-btn">
          Back to List
        </button>
      </div>
    </div>
  );
}

return (
  <div className="student-form-container">
    <div className="form-header">
      <h2>Edit Student (ID: {id})</h2>
      <button onClick={handleCancel} className="cancel-btn">
        ← Back to List
      </button>
    </div>

    <form onSubmit={handleSubmit} className="student-form">
      <div className="form-group">
        <label htmlFor="name">Name *</label>
        <input
          type="text"
          id="name"
          name="name"
```

```
          value={student.name}
          onChange={handleChange}
          className={errors.name ? 'error' : ''}
          placeholder="Enter student name"
        />
        {errors.name && <span className="error-message">{errors.name}</span>}
      </div>

      <div className="form-group">
        <label htmlFor="email">Email *</label>
        <input
          type="email"
          id="email"
          name="email"
          value={student.email}
          onChange={handleChange}
          className={errors.email ? 'error' : ''}
          placeholder="Enter email address"
        />
        {errors.email && <span className="error-message">{errors.email}</span>}
      </div>

      <div className="form-group">
        <label htmlFor="rollNo">Roll Number *</label>
        <input
          type="text"
          id="rollNo"
          name="rollNo"
          value={student.rollNo}
          onChange={handleChange}
          className={errors.rollNo ? 'error' : ''}
          placeholder="Enter roll number"
        />
        {errors.rollNo && <span className="error-message">{errors.rollNo}</span>}
      </div>

      <div className="form-actions">
        <button
          type="button"
          onClick={handleCancel}
          className="cancel-btn"
          disabled={loading}
        >
```

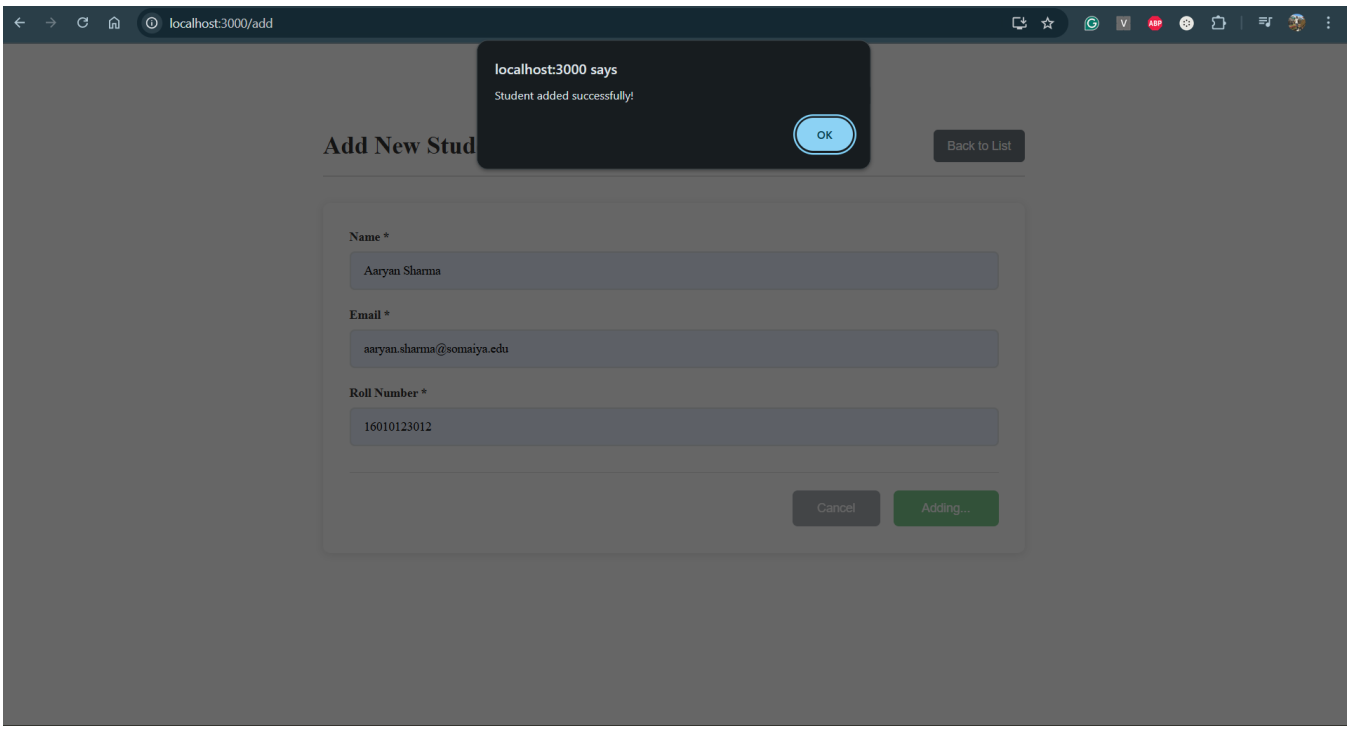
```
            Cancel
          </button>
          <button
            type="submit"
            className="submit-btn"
            disabled={loading}
          >
            {loading ? 'Updating...' : 'Update Student'}
          </button>
        </div>
      </form>
    </div>
  );
};

export default EditStudent;
```

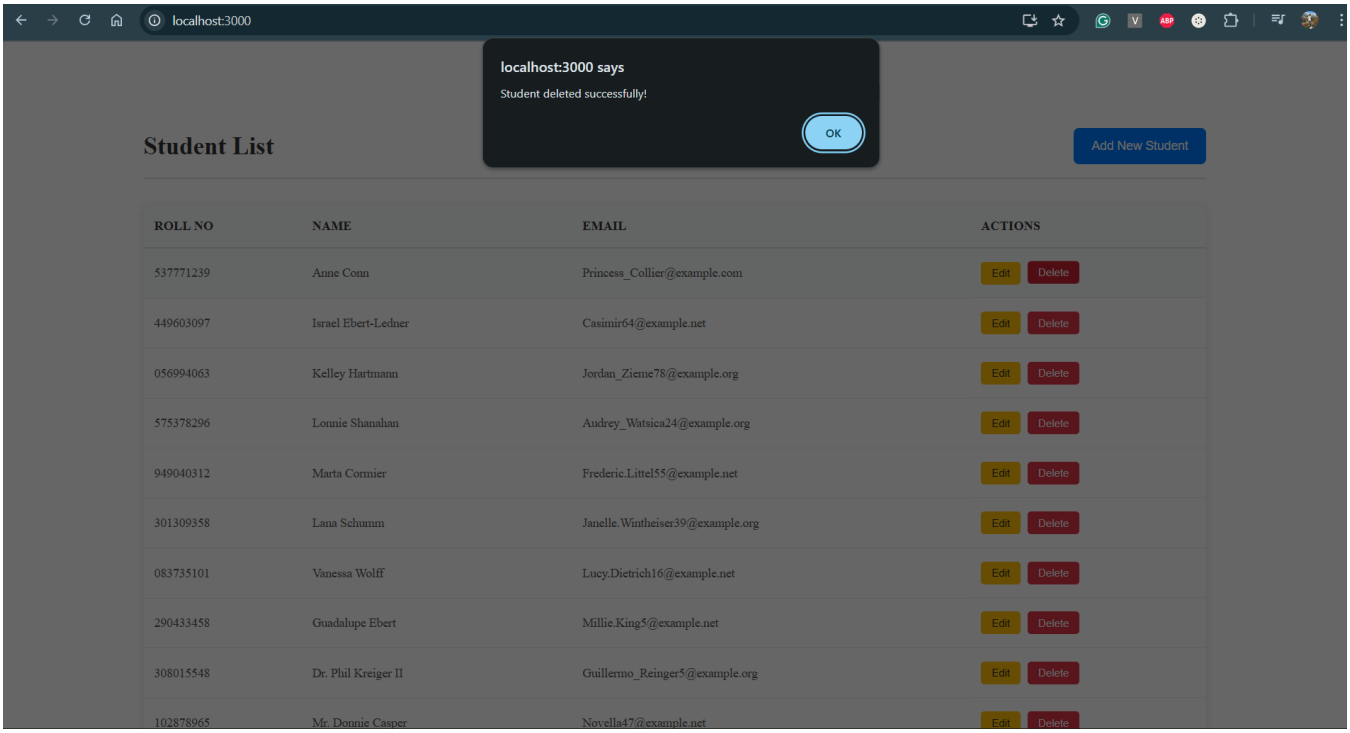
Output:

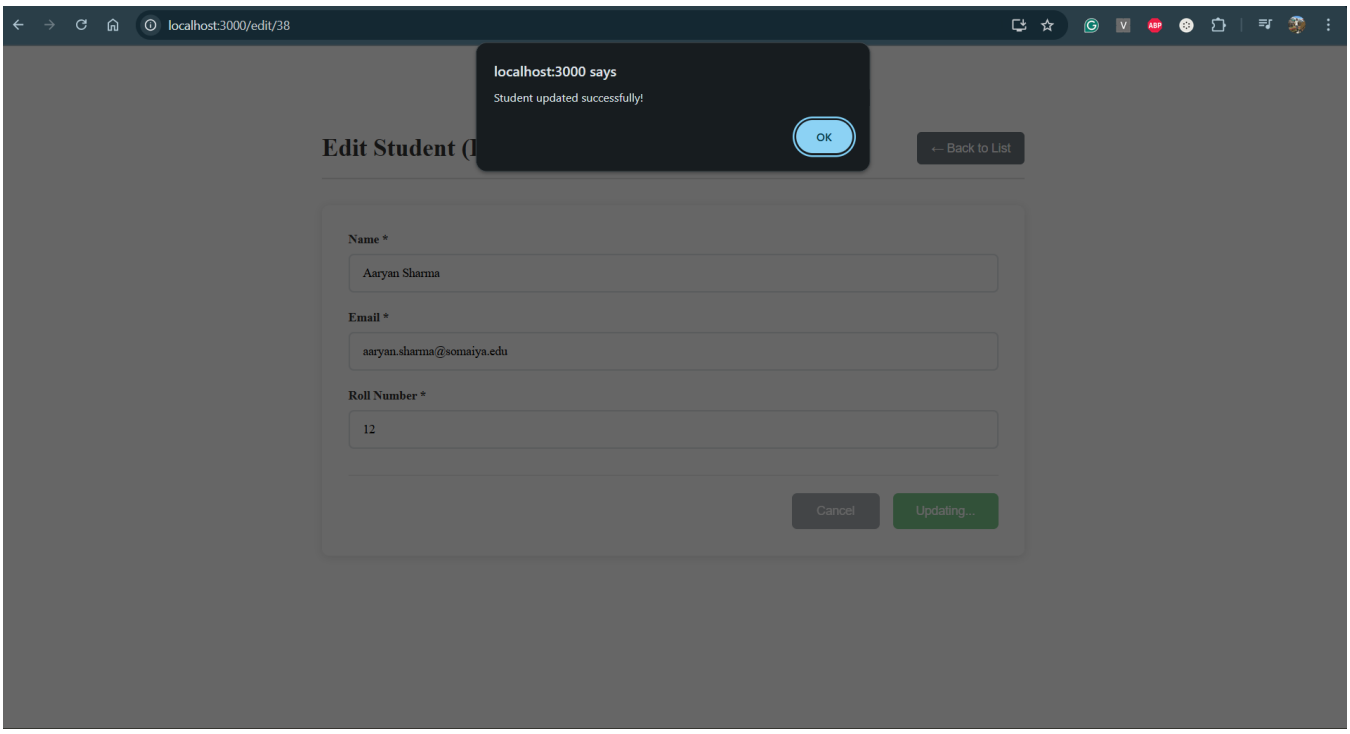
Student Management System

Student List Add New Student

ROLL NO	NAME	EMAIL	ACTIONS
537771239	Anne Conn	Princess_Collier@example.com	Edit Delete
449603097	Israel Ebert-Ledner	Casimir64@example.net	Edit Delete
056994063	Kelley Hartmann	Jordan_Zieme78@example.org	Edit Delete
575378296	Lonnie Shanahan	Audrey_Watsica24@example.org	Edit Delete
949040312	Marta Cormier	Frederic.Littel55@example.net	Edit Delete
301309358	Lana Schumm	Janelle.Wintheiser39@example.org	Edit Delete
083735101	Vanessa Wolff	Lucy.Dietrich16@example.net	Edit Delete
290433458	Guadalupe Ebert	Millie.King5@example.net	Edit Delete
308015548	Dr. Phil Kreiger II	Guillermo_Reinger5@example.org	Edit Delete
102878965	Mr. Donnie Casper	Novella47@example.net	Edit Delete


localhost:3000/add
localhost:3000 says
Student added successfully!
OK
Add New Stud
Back to List
Name *
Aaryan Sharma
Email *
aaryan.sharma@somaiya.edu
Roll Number *
16010123012
Cancel
Adding...


localhost:3000
localhost:3000 says
Student deleted successfully!
OK
Student List
Add New Student
ROLL NO
NAME
EMAIL
ACTIONS
537771239 Anne Conn Princess_Collier@example.com Edit Delete
449603097 Israel Ebert-Ledner Casimir64@example.net Edit Delete
056994063 Kelley Hartmann Jordan_Zieme78@example.org Edit Delete
575378296 Lonnie Shanahan Audrey_Watsica24@example.org Edit Delete
949040312 Marta Cormier Frederic.Littel55@example.net Edit Delete
301309358 Lana Schumm Janelle.Wintheiser39@example.org Edit Delete
083735101 Vanessa Wolff Lucy.Dietrich16@example.net Edit Delete
290433458 Guadalupe Ebert Millie.King5@example.net Edit Delete
308015548 Dr. Phil Kreiger II Guillermo_Reinger5@example.org Edit Delete
102878965 Mr. Donnie Casper Novella47@example.net Edit Delete



Conclusion:
In this experiment, we successfully demonstrated how to use Axios in a React application to interact with a mock RESTful API server. By implementing CRUD operations, we built a complete Student Management System where users can view, add, update, and delete student records. The use of Axios simplified API handling with its clean syntax, automatic JSON conversion, and error handling, while React Router (useNavigate) ensured smooth navigation across different views. This experiment clearly illustrated the integration of front-end and back-end concepts using modern web development tools and frameworks.

Postlab questions:
Different ways to Add Api in React/Javascript with example.
1. Using Fetch API - The fetch() method is built into modern browsers.
Pros: Built-in, no extra library.
Cons: No automatic JSON transformation for errors, less features compared to Axios.

```
// Fetch API (GET Request)
fetch('https://mockapi.io/students')
  .then(response => response.json())
  .then(data => {
    console.log("Students:", data);
  })
  .catch(error => console.error("Error fetching:", error));
```

2. **Using Async/Await with Fetch -** Makes code cleaner and easier to read.
Pros: Cleaner than .then().

Cons: Still lacks features like interceptors.

```
const fetchStudents = async () => {
  try {
    const response = await fetch('https://mockapi.io/students');
    const data = await response.json();
    console.log("Students:", data);
  } catch (error) {
    console.error("Error fetching:", error);
  }
};
fetchStudents();
```

3. Using Axios - Axios is a popular library for HTTP requests.

Pros: Easy syntax, interceptors, error handling, transforms automatically.

Cons: Requires installation.

```
import axios from 'axios';
axios.get('https://mockapi.io/students')
  .then(response => {
    console.log("Students:", response.data);
  })
  .catch(error => {
    console.error("Error fetching:", error);
  });
```

4. Using Async/Await with Axios - Combines Axios + async/await for clean code.

Pros: Clean, powerful, supports interceptors, error handling.

Cons: Needs installation.

```
import axios from 'axios';
const fetchStudents = async () => {
  try {
    const response = await axios.get(https://mockapi.io/students');
    console.log("Students:", response.data);
  } catch (error) {
    console.error("Error fetching:", error);
  }
};
fetchStudents();
```