FSD Lab 7

Aim: Develop a full stack web application using MERN stack to perform CRUD operations.

Objectives:

1. To develop full-stack web projects using the MERN stack.
2. To learn database connectivity using fetch api.
3. To perform insert, update, delete and search operations on database.

Theory:

1. What is MERN stack?

MERN Stack:

The MERN stack is a popular JavaScript stack used for building modern web applications. MERN stands for:

1. MongoDB: A NoSQL database that stores data in a flexible, JSON-like format.

2. Express.js: A back-end web application framework for Node.js that simplifies the process of building robust APIs.

3. React: A front-end library for building user interfaces, developed by Facebook. It allows developers to create reusable UI components.

4. Node.js: A JavaScript runtime that enables server-side execution of JavaScript, allowing the use of JavaScript for both client and server-side development.

The MERN stack provides a full JavaScript development environment for building scalable and dynamic web applications.

2. Use of Fetch api.

Use of Fetch API:

The Fetch API is a modern JavaScript API for making HTTP requests. It provides a cleaner and more flexible alternative to the older XMLHttpRequest. The Fetch API is commonly used in web development for the following purposes:

1. Data Retrieval: Fetch is used to make GET requests to retrieve data from a server or external API. It returns a Promise that resolves to the Response to that request.

```
javascript
fetch('https://api.example.com/data')
  .then(response => response.json())
  .then(data => console.log(data))
```

```
  .catch(error => console.error('Error:', error));
```

2. Sending Data (POST Requests): Fetch can be used to send data to a server using POST requests. This is common when submitting form data or sending data to an API endpoint.

```
javascript
fetch('https://api.example.com/save-data', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({ key: 'value' }),
})
.then(response => response.json())
.then(data => console.log(data))
.catch(error => console.error('Error:', error));
```

3. Handling Responses: The Fetch API allows for handling responses in various formats (JSON, text, etc.) and checking the status of the response.

```
javascript
fetch('https://api.example.com/data')
  .then(response => {
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    return response.json();
  })
  .then(data => console.log(data))
  .catch(error => console.error('Error:', error));
```

The Fetch API simplifies the process of making asynchronous requests and handling responses in web applications, making it a powerful tool for developers working with data retrieval and manipulation.

FAQ:

1. What makes MERN stack the fastest growing tech stack?

Ans.
MERN (MongoDB, Express.js, React, Node.js) cluster has become popular and is considered as one of the largest frameworks due to the following reasons:

1 . JavaScript is everywhere: MERN allows developers to use JavaScript for client- and server-side development. With Node.js on the server and React on the client, developers can use the

same language across the entire stack. This reduces context changes and makes it easier for developers to work on multiple areas of the application.

2. Full Edge Features: The MERN stack provides a complete and integrated framework for building modern web applications. Developers can seamlessly transition between front-end and back-end development by using a common language and technology framework.

3. React's component-based architecture: React's component-based architecture helps increase code reusability, making it easier to manage and manage complex applications. Components can be reused in different parts of the application, allowing for better development and easy scalability.

4. Rich Ecosystem and Community Support: Each part of the MERN group has a large and active community that provides comprehensive information, tutorials and third-party libraries. This rich ecosystem and community support helps the team grow as developers find solutions and best practices.

5. Performance and Scalability: Node.js is used server-side and is known for its non-blocking, event-driven architecture that is ideal for processing concurrent requests. This helps improve application performance and scalability, which is important for managing large user bases and workloads.

6. Flexibility of NoSQL databases: MongoDB is a NoSQL database used in the MERN group and handles different types of data and structures flexibly. This is especially useful in situations where the data schema will evolve over time and provides a better solution than relational data.

7. Rapid Prototyping: MERN stacking helps speed up prototyping and development. With tools like build-react-app and npm on the frontend and Express.js on the backend, developers can quickly build projects, manage dependencies, and focus on office building.

8. Single Language Stack: Using JavaScript across the entire stack shortens the developer's learning curve. They can be effective in any field without having to learn multiple languages, which is a key benefit of product development.

The combination of these factors makes the MERN group the ideal solution to provide effective, end-to-end solutions for startups, businesses and contract developers. productivity, capacity building and social welfare.

Project Structure:

1. Create a new React app for the front end:
   npx create-react-app student-regis-sys
   cd student-regis-sys

2. Set up the server using Express in a new directory named `server`:

```
mkdir server
cd server
npm init -y
npm install express mongoose cors
```

Front End (React):

App.js:
jsx

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';

const App = () => {
  const [students, setStudents] = useState([]);
  const [newStudent, setNewStudent] = useState({
    firstName: '',
    lastName: '',
    rollNo: '',
    password: '',
    confirmPassword: '',
    contactNumber: '',
  });

  useEffect(() => {
    // Fetch students on component mount
    axios.get('/api/students').then((response) => {
      setStudents(response.data);
    });
  }, []);

  const handleInputChange = (e) => {
    setNewStudent({ ...newStudent, [e.target.name]: e.target.value });
  };

  const handleInsert = () => {
    axios.post('/api/students', newStudent).then((response) => {
      setStudents([...students, response.data]);
      setNewStudent({
        firstName: '',
        lastName: '',
        rollNo: '',
        password: '',
```

```
      confirmPassword: '',
      contactNumber: '',
    });
  });
};

const handleDelete = (id) => {
  axios.delete(`/api/students/${id}`).then(() => {
    setStudents(students.filter((student) => student._id !== id));
  });
};

const handleUpdate = (id) => {
  // Implement update logic
};

return (
  <div>
    <h1>Student Registration System</h1>
    <form>
      {/* Input fields for new student */}
    </form>
    <button onClick={handleInsert}>Insert Student</button>

    {/* Display students in a table */}
    <table>
      {/* Table headers */}
      <tbody>
        {/* Table rows */}
      </tbody>
    </table>
  </div>
);
};

export default App;
```

Back End (Node.js and Express):

server/index.js:

```
javascript
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
const app = express();
```

```
app.use(cors());
app.use(express.json());

mongoose.connect('mongodb://localhost:27017/studentDB', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});

const studentSchema = new mongoose.Schema({
  firstName: String,
  lastName: String,
  rollNo: String,
  password: String,
  confirmPassword: String,
  contactNumber: String,
});

const Student = mongoose.model('Student', studentSchema);

app.get('/api/students', async (req, res) => {
  const students = await Student.find();
  res.json(students);
});

app.post('/api/students', async (req, res) => {
  const newStudent = new Student(req.body);
  const savedStudent = await newStudent.save();
  res.json(savedStudent);
});

app.delete('/api/students/:id', async (req, res) => {
  await Student.findByIdAndRemove(req.params.id);
  res.send('Student deleted successfully');
});

// Implement update route

const PORT = process.env.PORT || 5000;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});
```

To Run the Project:

1. Start the MongoDB server:

   mongod

2. Start the Express server (server directory):

   node index.js

3. Start the React app (main directory):

   npm start