Priyanshu Dhyani

Introduction to the MERN Stack

The MERN stack is a popular web development framework used to build dynamic web applications. The acronym "MERN" stands for MongoDB, Express.js, React, and Node.js. Each of these technologies plays a critical role in the development process, working together to create a powerful and efficient development environment.

Components of the MERN Stack

1. MongoDB
2. Express.js
3. React
4. Node.js

Roles of Each Technology

1. MongoDB

Role:

MongoDB is a NoSQL database that stores data in flexible, JSON-like documents. Unlike traditional relational databases, MongoDB does not use fixed table schemas, which allows for rapid and iterative development.

Features:

- Schema-less database: Offers flexibility in data structure.
- Scalability: Designed to scale out horizontally.
- Performance: Optimized for high write loads and data access patterns.

2. Express.js

Role:

Express.js is a lightweight web application framework for Node.js. It provides a robust set of features to develop web and mobile applications and serves as the backend for web applications in the MERN stack.

Features:

- Middleware: Simplifies the process of handling HTTP requests and responses.
- Routing: Offers a powerful system for defining routes.
- Integration: Easily integrates with various template engines.

3. React

Role:

React is a JavaScript library for building user interfaces. It allows developers to create reusable UI components and manage the state of their applications efficiently.

Features:
- Component-Based Architecture: Encourages the development of modular and reusable UI components.
- Virtual DOM: Enhances performance by minimizing direct manipulation of the DOM.
- One-Way Data Binding: Ensures predictable application behavior.

4. Node.js
Role:
Node.js is a JavaScript runtime built on Chrome's V8 engine. It allows developers to execute JavaScript on the server side and is used to build scalable network applications.

Features:
- Event-Driven Architecture: Handles multiple connections efficiently.
- Non-Blocking I/O: Allows for high concurrency.
- NPM (Node Package Manager): Provides access to thousands of libraries and modules.

Step-by-Step Guide to Setting Up a MERN Stack Application

Prerequisites
Before starting, ensure that you have the following installed on your system:
- Node.js and npm (Node Package Manager)
- MongoDB
- A code editor (e.g., Visual Studio Code)

Step 1: Setting Up the Backend with Node.js and Express.js

1. Create a new directory for your project:
```
mkdir mern-app
cd mern-app
```

2. Initialize a new Node.js project:
```
npm init -y
```

3. Install Express.js and other necessary packages:
```
npm install express mongoose body-parser cors
```

4. Create the server file (server.js):
```
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');
```

Priyanshu Dhyani

```
  const cors = require('cors');

  const app = express();
  const PORT = process.env.PORT || 5000;

  app.use(cors());
  app.use(bodyParser.json());

  // Connect to MongoDB
  mongoose.connect('mongodb://localhost:27017/mern-app', {
    useNewUrlParser: true,
    useUnifiedTopology: true,
  });
  const connection = mongoose.connection;
  connection.once('open', () => {
    console.log('MongoDB database connection established successfully');
  });
  // Define a simple schema and model
  const itemSchema = new mongoose.Schema({
    name: String,
  });
  const Item = mongoose.model('Item', itemSchema);

  // Define routes
  app.get('/items', (req, res) => {
    Item.find((err, items) => {
      if (err) {
        res.status(500).send(err);
      } else {
        res.json(items);
      }
    });
  });

  app.post('/items', (req, res) => {
    const newItem = new Item(req.body);
    newItem.save((err, item) => {
      if (err) {
        res.status(500).send(err);
      } else {
        res.status(201).json(item);
      }
    });
  });
  app.listen(PORT, () => {
    console.log(Server is running on port: ${PORT});
  });
```

5. Start the backend server:

```
  node server.js
```

Step 2: Setting Up the Frontend with React

1. Open a new terminal and navigate to the project directory:

```
  cd mern-app
```

2. Create a new React application:

```
  npx create-react-app client
  cd client
```

3. Install Axios for making HTTP requests:

```
  npm install axios
```

4. Create components for the frontend:

- Create a new file ItemList.js in the src directory:

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

    function ItemList() {
      const [items, setItems] = useState([]);

      useEffect(() => {
        axios.get('http://localhost:5000/items')
          .then(response => setItems(response.data))
          .catch(error => console.error('Error:', error));
      }, [ ]);

      return (
        <div>
          <h1>Item List</h1>
          <ul>
            {items.map((item, index) => (
              <li key={index}>{item.name}</li>
            ))}
          </ul>
        </div>
      );
    }

    export default ItemList;
```

- Create a new file AddItem.js in the src directory:

```
  import React, { useState } from 'react';
  import axios from 'axios';
  function AddItem() {
    const [name, setName] = useState('');

    const handleSubmit = (e) => {
      e.preventDefault();
      axios.post('http://localhost:5000/items', { name })
        .then(response => {
          console.log(response.data);
          setName('');
        })
        .catch(error => console.error('Error:', error));
    };
    return (
      <form onSubmit={handleSubmit}>
        <input
          type="text"
          placeholder="Add an item"
          value={name}
          onChange={(e) => setName(e.target.value)}
        />
        <button type="submit">Add Item</button>
      </form>
    );
  }
  export default AddItem;
```

Priyanshu Dhyani

5. Integrate components in the App.js file:

```
import React from 'react';
import ItemList from './ItemList';
import AddItem from './AddItem';

function App() {
  return (
    <div>
      <h1>My MERN App</h1>
      <AddItem />
      <ItemList />
    </div>
  );
}
export default App;
```

6. Start the React development server:

```
npm start
```

Step 3: Testing the Application

- Ensure both the backend server (running on http://localhost:5000) and the React development server (running on http://localhost:3000) are up and running.
- Open http://localhost:3000 in your browser.
- Add items using the input field and submit them.
- Verify that the items are displayed in the list below the input field.