Introduction

The MERN stack is a popular set of technologies used to build modern web applications. It consists of MongoDB, Express.js, React, and Node.js. Each technology plays a crucial role in the stack, contributing to a seamless development experience and robust application performance. This report will explain the role of each technology in the MERN stack and provide a step-by-step guide to setting up a MERN stack application.

---

1. Understanding the MERN Stack

1.1 MongoDB

- **Role**: MongoDB is a NoSQL database used for storing data in a flexible, JSON-like format called BSON (Binary JSON). It allows for scalable and efficient data storage and retrieval.
- **Features**:
    - Schema-less design, allowing for flexible and dynamic data models.
    - High performance and scalability.
    - Built-in replication and sharding for high availability and horizontal scaling.

1.2 Express.js

- **Role**: Express.js is a web application framework for Node.js. It provides a set of robust features for building web and mobile applications.
- **Features**:
    - Simplifies the creation of server-side applications and APIs.
    - Middleware support to handle HTTP requests, responses, and routing.
    - Lightweight and unopinionated, allowing developers to structure their applications as needed.

1.3 React

- **Role**: React is a front-end JavaScript library for building user interfaces. It allows developers to create reusable UI components and manage the state efficiently.
- **Features**:
    - Component-based architecture for reusable and maintainable code.
    - Virtual DOM for fast and efficient rendering.
    - Rich ecosystem with tools like React Router and Redux for enhanced functionality.

1.4 Node.js

- **Role**: Node.js is a runtime environment that allows JavaScript to be run on the server-side. It enables building scalable and high-performance applications using JavaScript.
- **Features**:
    - Asynchronous, event-driven architecture for handling concurrent connections efficiently.
    - Rich library of modules via npm (Node Package Manager).

- Cross-platform support.

---

2. Setting Up a MERN Stack Application

2.1 Prerequisites

- Node.js and npm installed.
- MongoDB installed and running.

2.2 Project Structure

```
java
code:
mern-app/
├── backend/
│   ├── node_modules/
│   ├── package.json
│   ├── server.js
│   ├── models/
│   │   └── user.js
│   ├── routes/
│   │   └── userRoutes.js
└── frontend/
    ├── node_modules/
    ├── public/
    ├── src/
    │   ├── components/
    │   │   └── UserList.js
    │   │   └── AddUser.js
    │   ├── App.js
    │   ├── index.js
    ├── package.json
```

2.3 Backend Setup

1. **Initialize the backend project:**

   ```
   bash
   code:
   mkdir backend
   cd backend
   npm init -y
   ```

2. **Install dependencies:**

   ```
   bash
   code:
   npm install express mongoose cors
   ```

3. **Create `server.js`:**

   ```
   javascript
   code:
   const express = require('express');
   const mongoose = require('mongoose');
   ```

```
const cors = require('cors');
const app = express();
const port = 5000;

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

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

const userRoutes = require('./routes/userRoutes');
app.use('/users', userRoutes);

app.listen(port, () => {
    console.log(`Server running at http://localhost:${port}`);
});
```

4. **Create User model (`models/user.js`):**

```
javascript
code:
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
    name: String,
    age: Number
});

module.exports = mongoose.model('User', userSchema);
```

5. **Create User routes (`routes/userRoutes.js`):**

```
javascript
code:
const express = require('express');
const router = express.Router();
const User = require('../models/user');

router.get('/', async (req, res) => {
    const users = await User.find();
    res.json(users);
});

router.post('/', async (req, res) => {
    const newUser = new User(req.body);
    await newUser.save();
    res.status(201).json(newUser);
});

module.exports = router;
```

6. **Start the backend server:**

```
bash
code:
node server.js
```

2.4 Frontend Setup

1. **Initialize the frontend project:**

2. bash

   code:
   npx create-react-app frontend
   cd frontend

3. **Install Axios:**

   bash
   code:
   npm install axios

4. **Edit `src/App.js`:**

   javascript
   code:
   import React from 'react';
   import UserList from './components/UserList';
   import AddUser from './components/AddUser';

   function App() {
       return (

               <h1>MERN Stack App</h1>
               <AddUser />
               <UserList />

       );
   }

   export default App;

5. **Create `src/components/UserList.js`:**

   javascript
   code:
   import React, { useEffect, useState } from 'react';
   import axios from 'axios';

   function UserList() {
       const [users, setUsers] = useState([]);

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

       return (

               <h2>User List</h2>

                   {users.map(user => (
                       <li key={user._id}>{user.name} - {user.age} years
   old
                   ))}

       );
   }

```
export default UserList;
```

6. **Create `src/components/AddUser.js`:**

```
javascript
code:
import React, { useState } from 'react';
import axios from 'axios';

function AddUser() {
    const [name, setName] = useState('');
    const [age, setAge] = useState('');

    const addUser = () => {
        axios.post('http://localhost:5000/users', { name, age:
parseInt(age) })
            .then(response => console.log(response.data))
            .catch(error => console.error('Error:', error));
    };

    return (
        <div>
            <h2>Add User</h2>
            <input type="text" value={name} onChange={(e) =>
setName(e.target.value)} placeholder="Name" />
            <input type="number" value={age} onChange={(e) =>
setAge(e.target.value)} placeholder="Age" />
            <button onClick={addUser}>Add User</button>
        </div>
    );
}

export default AddUser;
```

7. **Start the frontend server:**

```
bash
code:
npm start
```

2.5 Integration and Testing

1. **Ensure both backend and frontend servers are running:**
   - Backend: `node server.js` (http://localhost:5000)
   - Frontend: `npm start` (http://localhost:3000)
2. **Testing the application:**
   - Open the browser and navigate to http://localhost:3000.
   - You should see the "MERN Stack App" header with a list of users and a form to add a new user.
   - Use the input fields to add a new user and click the "Add User" button.
   - The new user should appear in the list.

Screenshots of the Working Application

- **Initial Screen:**
- **Adding a User:**
- **Updated User List:**

Conclusion

The MERN stack is a powerful set of technologies for building modern web applications. By combining MongoDB, Express.js, React, and Node.js, developers can create robust, scalable, and maintainable applications. This report has provided an overview of the MERN stack, the role of each technology, and a step-by-step guide to setting up a MERN stack application.