**Assignment 7.5**

**Understanding and Setting Up a MERN Stack Application**

**Introduction**

The MERN stack is a popular technology stack used for building modern web applications. MERN stands for MongoDB, Express.js, React, and Node.js. Each component in this stack plays a critical role in the development process, enabling developers to create full-fledged, dynamic web applications.

**The MERN Stack Components**

**MongoDB**

Role:

MongoDB is a NoSQL database that stores data in a flexible, JSON-like format called BSON. It is designed to handle large volumes of data and allows for easy scalability.

Usage:

MongoDB serves as the database layer of the MERN stack, where all application data is stored and managed. It offers high performance, high availability, and easy scalability.

**Express.js**

Role:

Express.js is a web application framework for Node.js. It provides a robust set of features for building single and multi-page web applications.

Usage:

Express.js acts as the backend framework of the MERN stack. It simplifies the process of handling HTTP requests, middleware management, and routing. Express.js helps in building APIs to interact with MongoDB.

**React**

Role:

React is a JavaScript library for building user interfaces, particularly single-page applications where a fast, interactive user experience is required.

Usage:

React handles the frontend of the MERN stack. It allows developers to create reusable UI components, manage state efficiently, and build dynamic and responsive web interfaces.

**Node.js**

Role:

Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. It allows JavaScript to be used for server-side scripting, enabling the development of scalable and high-performance web applications.

Usage:

Node.js is the runtime environment that executes JavaScript on the server side. It provides the server environment for running the backend of the MERN stack application.

**Setting Up a MERN Stack Application**

**Prerequisites**

Before setting up a MERN stack application, ensure you have the following installed:

* Node.js and npm (Node Package Manager)
* MongoDB
* A code editor (e.g., Visual Studio Code)

**Step-by-Step Guide**

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

**Initialize a new Node.js project:**

mkdir mern-app

cd mern-app

npm init –y

**Install Express.js:**

npm install express

**Create the server file:**

Create a server.js file and add the following code:

const express = require('express');

const app = express();

const PORT = process.env.PORT || 5000;

app.use(express.json());

app.get('/', (req, res) => {

res.send('Hello, MERN Stack!');

});

app.listen(PORT, () => {

console.log(`Server is running on port ${PORT}`);

});

**Run the server:**

node server.js

Your Express server should be running on http://localhost:5000.

**Step 2: Connect to MongoDB**

**Install Mongoose:**

npm install mongoose

**Set up the MongoDB connection:**

In server.js, add the following code to connect to MongoDB:

const mongoose = require('mongoose');

mongoose.connect('mongodb://localhost:27017/mernapp', {

useNewUrlParser: true,

useUnifiedTopology: true

}).then(() => {

console.log('Connected to MongoDB');

}).catch((err) => {

console.error('Failed to connect to MongoDB', err);

});

**Step 3: Set Up the Frontend with React**

**Create a React application:**

npx create-react-app client

**Move into the React application directory:**

cd client

**Start the React development server:**

npm start

The React app should be running on http://localhost:3000.

**Step 4: Integrate React with the Backend**

**Set up a proxy in React:**

In the client directory, add the following line to package.json:

"proxy": "http://localhost:5000"

**Fetch data from the backend in React:**

Modify App.js in the client/src directory to fetch data from the Express server:

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

import './App.css';

function App() {

const [message, setMessage] = useState('');

useEffect(() => {

fetch('/')

.then(response => response.text())

.then(data => setMessage(data));

}, []);

return (

<div className="App">

<header className="App-header">

<p>{message}</p>

</header>

</div>

);

}

export default App;

**Run both servers concurrently:**

Install concurrently to run both the backend and frontend servers at the same time:

npm install concurrently --save

**Modify the root package.json:**

In the root directory, update the scripts section in package.json:

"scripts": {

"start": "node server.js",

"client": "npm start --prefix client",

"dev": "concurrently \"npm run start\" \"npm run client\""

}

**Run the application:**

npm run dev

This command will start both the Express server and the React development server concurrently.