**Batch: A-3 Roll No.: 16010122104**

**Experiment / assignment / tutorial No. 7**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| Title: Implementation of MongoDB, Node.js and Express js. |

**AIM:** Implementation of MongoDB, Node.js and Express js.

**Problem Definition:**

**Resources used:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment:**

**CO 4:** **Test the concepts and components of various front-end, back-end web app**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. Shelly Powers Learning Node O’ Reilly 2 nd Edition, 2016.

**Pre Lab/ Prior Concepts:**

 **MongoDB**: MongoDB is a NoSQL database that stores data in a flexible, JSON-like format known as BSON (Binary JSON). It is a document-oriented database that does not require a predefined schema, making it scalable and efficient for handling large volumes of unstructured data. MongoDB is widely used in modern web applications for its performance, scalability, and flexibility in handling complex data structures.

 **Connection using Node.js, Express.js, and MongoDB**: To connect MongoDB with a Node.js and Express.js application, you need to install the mongoose package, which simplifies interaction with MongoDB by providing an Object Data Modeling (ODM) layer. Here’s how to set it up:

1. Install the necessary packages:

bash

Copy code

npm install express mongoose

1. Set up MongoDB connection in your Express app:

javascript

Copy code

const express = require('express');

const mongoose = require('mongoose');

const app = express();

// MongoDB connection URL

const dbURI = 'mongodb://localhost:27017/mydatabase';

// Connect to MongoDB using mongoose

mongoose.connect(dbURI, { useNewUrlParser: true, useUnifiedTopology: true })

.then(() => console.log('MongoDB connected'))

.catch((err) => console.log('Error connecting to MongoDB:', err));

// Start Express server

app.listen(3000, () => {

console.log('Server running on http://localhost:3000');

});

**Methodology:**

App.js

import React from 'react'

import { Routes, Route } from 'react-router-dom'

import Home from './pages/Home'

import Collection from './pages/Collection'

import About from './pages/About'

import Contact from './pages/Contact'

import Product from './pages/Product'

import Cart from './pages/Cart'

import Login from './pages/Login'

import PlaceOrder from './pages/PlaceOrder'

import Orders from './pages/Orders'

import Navbar from './components/Navbar'

import Footer from './components/Footer'

import SearchBar from './components/SearchBar'

import { ToastContainer, toast } from 'react-toastify';

import 'react-toastify/dist/ReactToastify.css';

import Verify from './pages/Verify'

const App = () => {

  return (

    <div className='px-4 sm:px-[5vw] md:px-[7vw] lg:px-[9vw]'>

      <ToastContainer />

      <Navbar />

      <SearchBar />

      <Routes>

        <Route path='/' element={<Home />} />

        <Route path='/collection' element={<Collection />} />

        <Route path='/about' element={<About />} />

        <Route path='/contact' element={<Contact />} />

        <Route path='/product/:productId' element={<Product />} />

        <Route path='/cart' element={<Cart />} />

        <Route path='/login' element={<Login />} />

        <Route path='/place-order' element={<PlaceOrder />} />

        <Route path='/orders' element={<Orders />} />

        <Route path='/verify' element={<Verify />} />

      </Routes>

      <Footer />

    </div>

  )

}

export default App

main.js

import React from 'react'

import ReactDOM from 'react-dom/client'

import App from './App.jsx'

import './index.css'

import { BrowserRouter } from 'react-router-dom'

import ShopContextProvider from './context/ShopContext.jsx'

ReactDOM.createRoot(document.getElementById('root')).render(

  <BrowserRouter>

    <ShopContextProvider>

      <App />

    </ShopContextProvider>

  </BrowserRouter>,

)

**Implementation Details:**

**1. Entry Point (index.jsx)**

**Code Explanation:**

* In index.jsx, the React app is being initialized using ReactDOM.createRoot to render the main application component (App).
* **BrowserRouter**: It wraps the entire app to handle routing. It enables navigation between different views of the application using URL paths.
* **ShopContextProvider**: This is a custom context provider used to manage global state for the shop (e.g., products, cart). It wraps the entire app, making the state accessible throughout the app.

**Steps for Execution:**

1. Ensure that the necessary dependencies (react-router-dom and any other related packages) are installed:

bash

Copy code

npm install react-router-dom

1. The index.jsx file must be placed in the root directory of your React app, typically inside the src folder.
2. The App.jsx file is imported and rendered inside the ReactDOM.createRoot() method.
3. Open the terminal, navigate to your project folder, and run the following command to start the development server:

bash

Copy code

npm run dev

This will compile the app and open it in the browser at http://localhost:3000.

**2. Main App (App.jsx)**

**Code Explanation:**

* **Routes**: The app is divided into multiple routes using React Router. Each route corresponds to a specific page/component that is rendered when the path matches.
  + /: Displays the **Home** page.
  + /collection: Displays the **Collection** page.
  + /about: Displays the **About** page.
  + /contact: Displays the **Contact** page.
  + /product/:productId: Displays the **Product** page (dynamic route with a productId).
  + /cart: Displays the **Cart** page.
  + /login: Displays the **Login** page.
  + /place-order: Displays the **Place Order** page.
  + /orders: Displays the **Orders** page.
  + /verify: Displays the **Verify** page.
* **ToastContainer**: This component from the react-toastify package is used to display toast notifications globally.
* **Navbar, Footer, SearchBar**: These components are included at the top and bottom of every page to ensure consistency across the app.

**Steps for Execution:**

1. Ensure that the necessary dependencies (react-router-dom, react-toastify, etc.) are installed:

bash

Copy code

npm install react-router-dom react-toastify

1. In the App.jsx file:
   * **Navbar**: Displays a navigation bar on all pages.
   * **SearchBar**: Displays a search bar for users to search for products.
   * **ToastContainer**: Displays toast messages that may be triggered during various interactions (e.g., success or error messages).
   * **Routes**: Defines multiple routes for different pages.

To use dynamic routing with parameters (e.g., :productId), the Product component must be able to retrieve and use the productId parameter via useParams:

javascript

Copy code

import { useParams } from 'react-router-dom';

const Product = () => {

const { productId } = useParams();

// Fetch product details using productId

};

1. Create each of the page components (Home, Collection, About, Contact, Product, etc.) in the pages folder as referenced in the Routes.
2. For the **Verify** page, ensure that you define the Verify component similarly and include any logic to verify user actions (e.g., verify an email).
3. Make sure that the **Navbar**, **Footer**, and **SearchBar** components are designed and implemented to be consistent with the layout.
4. Start the app using the following:

bash

Copy code

npm run dev

1. Navigate to http://localhost:3000/ to see the application in action, with routing properly set up.

**General Execution Flow:**

1. The index.jsx file initializes and renders the application, wrapping it in both BrowserRouter for routing and ShopContextProvider for global state management.
2. The App.jsx file contains all route definitions. Each route corresponds to a page (or view) that is displayed when the path is matched in the browser URL.
3. When a user visits a specific URL path, React Router loads the corresponding component, and the navigation is handled seamlessly without page reloads.
4. Global state, like products or cart items, can be accessed in any component via context (ShopContextProvider).
5. The ToastContainer will display notifications for events triggered by actions like product add/remove or successful actions like form submissions.

**Conclusion: We implemented the MongoDB, Node.js and Express js components of our project.**