# Lab 6 [Building a Product App with React, Node, Express and MongoDB]

|  |  |
| --- | --- |
| Lab Title: | Building a Product App with React, Node, and MongoDB |
| **Expected duration** (hours): | **1 hour 45 mins** |

|  |
| --- |
| Objectives |
| Understanding how to integrate with MongoDB from a NodeJS app |
| Learn how to integrate with a ReactJS app with a NodeJS app |
| Learn how to create a Product with React, Node, Express and Mongo (MERN) |

|  |
| --- |
| Requirements |
| A laptop or a desktop with Windows or Mac as an operating system |
| Visual studio code or something similar as a text editor |
| A modern web browser like chrome |

## Lab Instructions:

### **1. Integrate MongoDB from a NodeJS application**

* Install MongoDB Node.js Driver:
  + First, ensure you have the MongoDB Node.js driver installed. You can install it using npm:

npm install mongodb

* Set Up MongoDB Connection:
  + Create a connection to your MongoDB database in your NodeJS app. Here’s an example of how to do it:

import { MongoClient } from 'mongodb';

const uri = '<mongo db atlast connection url for nodejs>;

const client = new MongoClient(uri);

* Replace Connection String:
  + Replace 'mongodb://localhost:27017' with your actual MongoDB connection string. Since we are using MongoDB Atlas, your connection string will be different.
  + Navigate to MongoDB atlas and click on the connection.
  + Based on your connection type, it will provide you a connection URL. Remember to update your URL with the password you setup on Mongo DB lab.
* Handle Errors:
  + Ensure you handle any potential errors in the connection process and operations.
* Test the Connection:
  + Run your NodeJS app to ensure it connects to MongoDB and performs the desired operations (e.g., read all documents).

### **2. Update Product Listing & Product Creation endpoint**

* Update Get List of Products section of the code in your NodeJS app with the below details -

let products = [];

try {

// Connect to the MongoDB cluster

await client.connect();

// Confirm connection

console.log('Connected to MongoDB');

// Specify database and collection

const database = client.db('productApp');

const collection = database.collection('products');

// Fetch all products

products = await collection.find({}).toArray();

// Log the list of products

console.log('List of Products:', products);

res.json(products);

} catch (error) {

console.error('Error connecting to MongoDB', error);

} finally {

// Close the connection

await client.close();

}

* Update Create new product endpoint section of the code in your NodeJS app with the below details -

// Validation middleware

[

body('name').isString().withMessage('Name must be a string'),

body('description').isString().withMessage('Description must be a string'),

body('price').isFloat({ gt: 0 }).withMessage('Price must be a positive number')

],

async (req, res) => {

const errors = validationResult(req);

if (!errors.isEmpty()) {

return res.status(400).json({ errors: errors.array() });

}

const { name, description, price } = req.body;

const newProduct = {

name,

description,

price

};

try {

// Connect to the MongoDB cluster

await client.connect();

// Confirm connection

console.log('Connected to MongoDB');

// Specify database and collection

const database = client.db('productApp');

const collection = database.collection('products');

// Example operation (insert a document)

const result = await collection.insertOne(newProduct);

console.log('Document inserted with \_id: ', result.insertedId);

res.status(201).json(newProduct);

} catch (error) {

console.error('Error connecting to MongoDB', error);

} finally {

// Close the connection

await client.close();

}

}

* Test your application using Postman -
  + <http://localhost:3000/products> [GET]
    - Should return all products created so far in MongoDB atlas.
  + <http://localhost:3000/products> [POST]
    - Should create a new product with 201 status code.
      * Sample payload to create a new product

{

"name": "Product 4",

"description": "This is a test product",

"price": 400

}

Congrats!! Now you have an end-to-end product application working using NodeJS and MongoDB, which can list all products and create a new product.

### **3. Integrate ReactJS application with NodeJS application**

* Configure Axios (or Fetch API) in React App to Consume APIs
  + In your React app, use Axios (or Fetch API) to make HTTP requests to the API endpoints exposed by your NodeJS app.

import axios from 'axios';

const BASE\_URL = 'http://localhost:3000';

// function to fetch products

export async function fetchProducts() {

try {

const response = await axios.get(`${BASE\_URL}/products`);

return response.data;

} catch (error) {

console.error('Error fetching products:', error);

return [];

}

}

// function to create a new product

export async function createProduct(productData) {

try {

const response = await axios.post(`${BASE\_URL}/products`, productData);

return response.data;

} catch (error) {

console.error('Error creating product:', error);

return null;

}

}

* Update react component to display product data
  + Navigate to ProductList.jsx component and update the code as below

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

import { fetchProducts } from './productApi';

function ProductList() {

const [products, setProducts] = useState([]);

useEffect(() => {

async function fetchData() {

const productsData = await fetchProducts();

setProducts(productsData);

}

fetchData();

}, []);

return (

<div className="container">

<h1>Available Products</h1>

{products.map(product => (

<div className="product" key={product.\_id}>

<h2>{product.name}</h2>

<p>{product.description}</p>

<p className="price">{product.price}</p>

</div>

))}

</div>

);

}

export default ProductList;

* Update react component to create product data
  + Navigate to NewProduct.jsx component and update the code as below -

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

import { createProduct } from './productApi';

const NewProduct = ({ addProduct }) => {

const [name, setName] = useState('');

const [description, setDescription] = useState('');

const [price, setPrice] = useState('');

const handleSubmit = async (e) => {

e.preventDefault();

const newProduct = { id: Date.now(), name, description, price };

await createProduct(newProduct);

setName('');

setDescription('');

setPrice('');

};

return (

<div className="container">

<h1>Create New Product</h1>

<form onSubmit={handleSubmit}>

<div>

<label>Product Name:</label>

<input

type="text"

value={name}

onChange={(e) => setName(e.target.value)}

required

/>

</div>

<div>

<label>Description:</label>

<input

type="text"

value={description}

onChange={(e) => setDescription(e.target.value)}

required

/>

</div>

<div>

<label>Price:</label>

<input

type="text"

value={price}

onChange={(e) => setPrice(e.target.value)}

required

/>

</div>

<button type="submit">Add Product</button>

</form>

</div>

);

};

export default NewProduct;

* Test the application now for listing Product data.
  + Navigate to react application folder and run the below command

npm run dev

* Navigate to node js application and run the below command

npm start

* When navigate to the browser to view Product listing page – see if it returns all the available products created so far.

Congrats - you have managed to run a react application in coordination with a node js application to return data from a collection and also create data into a collection of a MongoDB end to end. So, this is a perfect example of a MERN stack application to retrieve data from database.