Yarn init

Yarn global add typescript

Tsc –init

Uncomment these - A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

Before –

A screenshot of a computer program

Description automatically generated

After- strict false

"outDir": "./build",

/\* Modules \*/

"module": "commonjs", /\* Specify what module code is generated. \*/

"rootDir": "./src", /\* Specify the root folder within your source files. \*/

"moduleResolution": "node10",

\

yarn add typescript express cors mongoose ts-node jsonwebtoken bcrypt nodemon

yarn add @types/express @types/node

"scripts": {

"start": "node ./build/index.js",

"build": "tsc -p .",

"dev": "nodemon ./src/index.ts"

},

Connect to database – create user.ts in /src/models

import { Schema, model } from "mongoose";

export interface IUser {

\_id?: string;

username: string;

password: string;

availableMoney: number;

purchasedItems: string[];

}

const UserSchema = new Schema<IUser>({

username: { type: String, required: true, unique: true },

password: { type: String, required: true },

availableMoney: { type: Number, default: 5000 },

purchasedItems: [

{ type: Schema.Types.ObjectId, ref: "product", default: [] },

],

});

export const UserModel = model<IUser>("user", UserSchema);

Create collection in mongoose

Create user.ts file –

import express from "express";

import jwt from "jsonwebtoken";

import bcrypt from "bcrypt";

import { IUser, UserModel } from "../models/user";

import { UserErrors } from "../routes/errors";

const router = express.Router();

router.post("/register", async (req, res) => {

const { username, password } = req.body; // Get the username and password from the request body

try {

const user = await UserModel.findOne({ username }); // Check if the user already exists

if (user) { // If the user exists, return an error

return res.status(400).json({ type: UserErrors.USERNAME\_ALREADY\_EXISTS }); // Return a 400 status code and the error type

}

const hashedPassword = await bcrypt.hash(password, 10); // Hash the password

const newUser = new UserModel({ username, password: hashedPassword }); // Create a new user

await newUser.save(); // Save the user to the database

res.json({ message: "User registered successfully" });

} catch (err) { // If an error occurs, return a 500 status code and the error

res.status(500).json({ type: err });

}

});

export {router as userRouter}; // Export the router

Test api –

A screenshot of a computer

Description automatically generated

Create router.post(“/register”)

router.post("/register", async (req, res) => {

const { username, password } = req.body; // Get the username and password from the request body

try {

const user = await UserModel.findOne({ username }); // Check if the user already exists

if (user) { // If the user exists, return an error

return res.status(400).json({ type: UserErrors.USERNAME\_ALREADY\_EXISTS }); // Return a 400 status code and the error type

}

const hashedPassword = await bcrypt.hash(password, 10); // Hash the password

const newUser = new UserModel({ username, password: hashedPassword }); // Create a new user

await newUser.save(); // Save the user to the database

res.json({ message: "User registered successfully" });

} catch (err) { // If an error occurs, return a 500 status code and the error

res.status(500).json({ type: err });

}

});

Create router.post(“/login”)

router.post("/login", async (req, res) => { // Create a new route to handle user login

const { username, password } = req.body;

try {

const user: IUser = await UserModel.findOne({ username }); // Find the user by the username

if (!user) {

return res.status(400).json({ type: UserErrors.USER\_NOT\_FOUND });

}

const isPasswordValid = await bcrypt.compare(password, user.password);

if (!isPasswordValid) {

return res.status(400).json({ type: UserErrors.WRONG\_CREDENTIALS });

}

const token = jwt.sign({ id: user.\_id }, "secret"); // Create a token with the user ID and the secret key "secret" (this should be a long and complex string)

res.json({ token, userID: user.\_id }); // Return the token and the user ID in the response body

} catch (err) {

res.status(500).json({ type: err });

}

});

**Create Middleware –**

export const verifyToken = (req:Request, res:Response, next: NextFunction) => { // Create a middleware to verify the token

const authHeader = req.headers.authorization; // Get the authorization header from the request headers

if (authHeader) { // If the authorization header exists

jwt.verify(authHeader, "secret", (err) => { // Verify the token with the secret

if (err) {

return res.sendStatus(403); // If the token is invalid, return a 403 status code and 403 means forbidden

}

next(); // If the token is valid, call the next function in the middleware chain

});

} else {

res.sendStatus(401); // If the authorization header doesn't exist, return a 401 status code and 401 means unauthorized

}

};

Client Side –

yarn create react-app . --template typescript

yarn add axios react-cookie react-router-dom @fortawesome/react-fontawesome @fortawesome/free-solid-svg-icons @fortawesome/fontawesome-svg-core

**@fortawesome/react-fontawesome, @fortawesome/free-solid-svg-icons, @fortawesome/fontawesome-svg-core:**

* **Purpose:** These packages are part of the Font Awesome icon library, which provides a collection of scalable vector icons.
* **Usage:** You can use Font Awesome icons to enhance the visual appearance of your React components.

Yarn start

Create a Navbar and App.tsx – basic routes to all pages –

import React from 'react';

import { BrowserRouter as Router, Routes ,Route} from 'react-router-dom';

import { Navbar } from "./components/navbar";

import "./App.css";

import { AuthPage } from './pages/auth';

import { CheckoutPage } from './pages/checkout';

import { PurchasedItemsPage } from './pages/purchased-items';

import { ShopPage } from './pages/shop';

function App() {

return (

<div className='App'>

<Router>

<Navbar />

<Routes>

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

<Route path='/auth' element={<AuthPage/>} />

<Route path='/checkout' element={<CheckoutPage/>} />

<Route path='/purchased-items' element={<PurchasedItemsPage/>} />

</Routes>

</Router>

</div>

);

}

export default App;

Auth/index.tsx –

export const AuthPage = () => {

return (

<div className="auth">

<Register />

<Login />

</div>

);

};

const Register = () => {

return <div>Register</div>

};

const Login = () => {

return <div>Login</div>

};

Basic setup for register –

const Register = () => {

const [username, setUsername] = useState("");

const [password, setPassword] = useState("");

return (

<div className="auth-container">

<form>

<h2>Register</h2>

<div className="form-group">

<label htmlFor="username">UserName : </label>

<input type="text" id="username" value={username} onChange={(event) => setUsername(event.target.value)}/>

</div>

<div className="form-group">

<label htmlFor="password">Password : </label>

<input type="password" id="password" value={password} onChange={(event) => setUsername(event.target.value)} />

</div>

<button type="submit">Register</button>

</form>

</div>

);

};

Geeting error for err –

const Register = () => {

const [username, setUsername] = useState("");

const [password, setPassword] = useState("");

const handleSubmit = async (event: SyntheticEvent) => {

event.preventDefault();

try {

await axios.post("http://localhost:3001/auth/register", {

username,

password,

});

alert("Registration Completed! Now login.");

} catch (err) {

if (err.response.data.type === UserErrors.USER\_NOT\_FOUND) {

alert("ERROR: No user found");

} else {

alert("ERROR: Something went wrong");

}

}

};

Make strict as false –

A screen shot of a computer

Description automatically generated

Register function –

const Register = () => {

const [username, setUsername] = useState("");

const [password, setPassword] = useState("");

const handleSubmit = async (event: SyntheticEvent) => {

event.preventDefault();

try {

await axios.post("http://localhost:3001/user/register", {

username,

password,

});

alert("Registration Completed! Now login.");

} catch (err) {

if (err.response.data.type === UserErrors.USER\_NOT\_FOUND) {

alert("ERROR: No user found");

} else {

alert("ERROR: Something went wrong");

}

}

};

Finalize login for now –

const Login = () => {

const [\_, setCookies] = useCookies(["access\_token"]); // it

const [username, setUsername] = useState<string>("");

const [password, setPassword] = useState<string>("");

const navigate = useNavigate();

const handleSubmit = async (event: SyntheticEvent) => {

event.preventDefault();

try {

const result = await axios.post("http://localhost:3001/user/login", {

username,

password,

});

setCookies("access\_token", result.data.token);

localStorage.setItem("userID", result.data.userID);

// setIsAuthenticated(true);

navigate("/");

} catch (err) {

let errorMessage: string = "";

switch (err.response.data.type) {

case UserErrors.USER\_NOT\_FOUND:

errorMessage = "User doesn't exists";

break;

case UserErrors.WRONG\_CREDENTIALS:

errorMessage = "Wrong username/password combination";

break;

default:

errorMessage = "Something went wrong";

}

alert("ERROR: " + errorMessage);

}

};

return (

<div className="auth-container">

<form onSubmit={handleSubmit}>

<h2>Login</h2>

<div className="form-group">

<label htmlFor="username">UserName : </label>

<input type="text" id="username" value={username} onChange={(event) => setUsername(event.target.value)}/>

</div>

<div className="form-group">

<label htmlFor="password">Password : </label>

<input type="password" id="password" value={password} onChange={(event) => setPassword(event.target.value)} />

</div>

<button type="submit">Login</button>

</form>

</div>

);

Now Create front end –

Create shop page for that create custom hook – useGetProducts –

Use effect hook can not be made async …so we can fetch details from a function that is async

**useGetProducts.ts -**

import axios from "axios";

import { useEffect, useState } from "react";

export const useGetProducts = () => {

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

const fetchProducts = async () => {

const products = await axios.get("http://localhost:3001/products",{headers:{Authorization:localStorage.getItem("token")}});

setProducts(products.data.products);// products.data.products is the array of products from the server

};

useEffect(() => {

fetchProducts();

}, []);

return { products, fetchProducts };

};

**useGetToken –**

import { useCookies } from "react-cookie";

export const useGetToken = () => {

const [cookies, \_] = useCookies(["access\_token"]);

return {

headers: { authorization: cookies.access\_token },

};

};

Product.tsx -

import { ShopContext } from "../../context/shop-context";

import { IProduct } from "../../models/interfaces";

import React, { useContext } from "react";

// interface Props is a TypeScript feature that allows you to define the shape of an object.

interface Props {

product: IProduct;

}

// The Product component is a functional component that takes in a product object as a prop and returns a div with the product details.

export const Product = (props: Props) => {

const { \_id, productName, description, price, stockQuantity, imageURL } =

props.product;

const { addToCart, getCartItemCount} = useContext(ShopContext);

const count = getCartItemCount(\_id);

console.log(count);

return (

<div className="product">

<img src={imageURL} /> {" "}

<div className="description">

<h3>{productName}</h3>

<p>{description}</p>

<p> ${price}</p>

</div>

<button className="add-to-cart-bttn" onClick={() => addToCart(\_id)}>

Add to Cart {count > 0 && <> ({count}) </>}

</button>

<div className="stock-quantity">

{/\* // If the stockQuantity is 0, display "OUT OF STOCK" \*/}

{stockQuantity === 0 && <h1> OUT OF STOCK</h1>}

</div>

</div>

);

};

Shop-context.tsx –

import { ReactNode, createContext, useEffect, useState } from "react";

import { useGetProducts } from "../hooks/useGetProducts";

import { IProduct } from "../models/interfaces";

import axios, { AxiosError, AxiosResponse } from "axios";

// import { ProductErrors } from "../models/errors";

import { useNavigate } from "react-router-dom";

import { useCookies } from "react-cookie";

import { useGetToken } from "../hooks/useGetToken";

export interface IShopContext {

getCartItemCount: (itemId: string) => number;

addToCart: (itemId: string) => void;

updateCartItemCount: (newAmount: number, itemId: string) => void;

removeFromCart: (itemId: string) => void;

}

const defaultVal: IShopContext = {

addToCart: () => null,

removeFromCart: () => null,

updateCartItemCount: () => null,

getCartItemCount: () => 0,

};

// Create a context with the default value of an empty object

export const ShopContext = createContext<IShopContext>(defaultVal);

export const ShopContextProvider = (props) => {

const [cartItems, setCartItems] = useState<{string:number} | {}>({}); // Create a state to store the cart items

const addToCart = (itemId: string) => {

if (!cartItems[itemId]) {

setCartItems((prev) => ({ ...prev, [itemId]: 1 }));

} else {

setCartItems((prev) => ({ ...prev, [itemId]: prev[itemId] + 1 }));

}

};

const getCartItemCount = (itemId: string): number => {

if (itemId in cartItems) {

return cartItems[itemId];

}

return 0;

};

const removeFromCart = (itemId: string) => { // Add an item to the cart

};

const updateCartItemCount = (newAmount: number, itemId: string) => { // Add an item to the cart

};

const contextValue: IShopContext = {

addToCart,

removeFromCart,

updateCartItemCount,

getCartItemCount,

};

return (

<ShopContext.Provider value={contextValue}>

{props.children}

</ShopContext.Provider>

);

}