Installed package

1. Npm install react-router-dom react –router-bootstrap --save

React-router-dom is react router and react-router-bootstrap is with react bootstrap component like navbar

1. Npm i axios

Axios is used to call rest api

1. To convert node js script into ecma script forma, use latest version of node above 14.6 and add following line inside package.json

“type”:”module”

Used in routes

Module.exports = products

export default products;

include package

const express = require(‘express’);

import express from 'express';

any file we want to import

const products = require(‘./data/products’);

import products from './data/products.js';

1. To run client and server script concurrently, concurrently devdependencies is used

npm I –d concurrently

 "scripts": {

    "start": "node backend/server",

    "server": "nodemon backend/server",

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

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

  },

Add proxy in frontend/package.json to call backend api

{

  "name": "frontend",

  "proxy": "http://127.0.0.1:5000",

  "version": "0.1.0",

1. **Express async middleware handler (Node Package)**

npm i express-async-handler

express-async-handler is used place of try catch inside route that handle error

1. Bcryptjs is used to hash the password like md5 in php

npm i bcryptjs

import bcrypt from 'bcryptjs';

{

    name: 'Admin User',

    email: 'admin@example.com',

    password: bcrypt.hashSync('123456', 10),

    isAdmin: true,

}

1. **Error handling in node js**

Install package to handle async request, It is used as middleware

npm i express-async-handler

productRoute.js

import express from 'express';

import asyncHandler from 'express-async-handler'; //import

import Product from '../models/productModel.js';

const router = express.Router();

// @desc    Fetch all products

// @route   GET /api/products

// @access  Public

router.get(

  '/',

  asyncHandler(async (req, res) => { //asyncHandler - it remove try catch to handle error

    const products = await Product.find({});

    res.json(products);

  })

);

// @desc    Fetch single product

// @route   GET /api/products/:id

// @access  Public

router.get(

  '/:id',

  asyncHandler(async (req, res) => {

    const product = await Product.findById(req.params.id);

    if (product) {

      res.json(product);

    } else {

      res.status(404); //throw custom error

      throw new Error('Product not found');

    }

  })

);

export default router;

Handle error at server.js/app.js/index.js (Main file)

Middleware /errorMiddleware.js

const notFound = (req, res, next) => {

  const error = new Error(`Not Found - ${req.originalUrl}`);

  res.status(404);

  next(error);

};

const errorHandler = (err, req, res, next) => {

  const statusCode = res.statusCode === 200 ? 500 : res.statusCode;

  res.status(statusCode);

  res.json({

    message: err.message,

    stack: process.env.NODE\_ENV === 'production' ? null : err.stack,

  });

};

export { notFound, errorHandler };

import { notFound, errorHandler } from './middleware/errorMiddleware.js';

server.js

call after all routes

app.use('/api/products', productRoutes);

app.use(notFound);

app.use(errorHandler);

1. **React redux**

Middlware in redux

Npm i redux react-redux redux-thunk redux-devtools-extension

Redux-thunk is a middleware that handle async request

1. **React redux implementation [Part 1]**

**Step1: Create store; store.js**

import { createStore, combineReducers, applyMiddleware } from 'redux';

import thunk from 'redux-thunk';

import { composeWithDevTools } from 'redux-devtools-extension';

import {

  productDetailsReducer,

  productListReducer,

} from './reducers/productReducer';

import { cartReducer } from './reducers/cartReducer';

const cartItemsFromStorage = localStorage.getItem('cartItems')

  ? JSON.parse(localStorage.getItem('cartItems'))

  : [];

const initialState = {

  cart: {

    cartItems: cartItemsFromStorage,

  },

};

const reducer = combineReducers({

  productList: productListReducer,

  productDetails: productDetailsReducer,

  cart: cartReducer,

});

const middleware = [thunk];

const store = createStore(

  reducer,

  initialState,

  composeWithDevTools(applyMiddleware(...middleware))

);

export default store;

**Explanation:**

**createStore is used to create store**

**it take three arguments**

**1)reducer**

**2) initialState of store [optional]**

**3)Middlewares ([thunk] is used to handle async request of http)**

const store = createStore(

  reducer,

  initialState,

  composeWithDevTools(applyMiddleware(...middleware))

);

**CombinReducers** is used to combined multiple reducers into meaningful object store

const reducer = combineReducers({

  productList: productListReducer,

  productDetails: productDetailsReducer,

  cart: cartReducer,

});

**applyMiddleware** is used to apply multiple middleware

'redux-devtools-extension' is used to work with redux devtool chrome extension that help to know state and tree structure of data.

Step2: Wrap the entire app with store (index.js)

import React from 'react';

import ReactDOM from 'react-dom';

import { Provider } from 'react-redux';

import './bootstrap.min.css';

import './index.css';

import App from './App';

import \* as serviceWorker from './serviceWorker';

import store from './store';

ReactDOM.render(

  <Provider store={store}>

    <App />

  </Provider>,

  document.getElementById('root')

);

// If you want your app to work offline and load faster, you can change

// unregister() to register() below. Note this comes with some pitfalls.

// Learn more about service workers: https://bit.ly/CRA-PWA

serviceWorker.unregister();

Step3: Create action contant (constants/productConstant.js)

export const PRODUCT\_LIST\_REQUEST = 'PRODUCT\_LIST\_REQUEST';

export const PRODUCT\_LIST\_SUCCESS = 'PRODUCT\_LIST\_SUCCESS';

export const PRODUCT\_LIST\_FAIL = 'PRODUCT\_LIST\_FAIL';

export const PRODUCT\_DETAILS\_REQUEST = 'PRODUCT\_DETAILS\_REQUEST';

export const PRODUCT\_DETAILS\_SUCCESS = 'PRODUCT\_DETAILS\_SUCCESS';

export const PRODUCT\_DETAILS\_FAIL = 'PRODUCT\_DETAILS\_FAIL';

Step4: Create Reducer (reducers/productReducer/js)

import {

  PRODUCT\_LIST\_REQUEST,

  PRODUCT\_LIST\_SUCCESS,

  PRODUCT\_LIST\_FAIL,

  PRODUCT\_DETAILS\_REQUEST,

  PRODUCT\_DETAILS\_SUCCESS,

  PRODUCT\_DETAILS\_FAIL,

} from '../constants/productConstant';

export const productListReducer = (state = { products: [] }, action) => {

  switch (action.type) {

    case PRODUCT\_LIST\_REQUEST:

      return { loading: true, products: [] };

    case PRODUCT\_LIST\_SUCCESS:

      return { loading: false, products: action.payload };

    case PRODUCT\_LIST\_FAIL:

      return { loading: false, error: action.payload };

    default:

      return state;

  }

};

export const productDetailsReducer = (

  state = { product: { reviews: [] } },

  action

) => {

  switch (action.type) {

    case PRODUCT\_DETAILS\_REQUEST:

      return { loading: true, ...state };

    case PRODUCT\_DETAILS\_SUCCESS:

      return { loading: false, product: action.payload };

    case PRODUCT\_DETAILS\_FAIL:

      return { loading: false, error: action.payload };

    default:

      return state;

  }

};

Step5: Create actions (action/productAction.js)

import {

  PRODUCT\_LIST\_FAIL,

  PRODUCT\_LIST\_SUCCESS,

  PRODUCT\_LIST\_REQUEST,

  PRODUCT\_DETAILS\_REQUEST,

  PRODUCT\_DETAILS\_FAIL,

  PRODUCT\_DETAILS\_SUCCESS,

} from '../constants/productConstant';

import axios from 'axios';

export const listProducts = () => async (dispatch) => {

  try {

    dispatch({ type: PRODUCT\_LIST\_REQUEST });

    const { data } = await axios.get('/api/products');

    dispatch({ type: PRODUCT\_LIST\_SUCCESS, payload: data });

  } catch (error) {

    dispatch({

      type: PRODUCT\_LIST\_FAIL,

      payload:

        error.response && error.response.data.message

          ? error.response.data.message

          : error.message,

    });

  }

};

export const listProductDetails = (id) => async (dispatch) => {

  try {

    dispatch({ type: PRODUCT\_DETAILS\_REQUEST });

    const { data } = await axios.get(`/api/products/${id}`);

    dispatch({ type: PRODUCT\_DETAILS\_SUCCESS, payload: data });

  } catch (error) {

    dispatch({

      type: PRODUCT\_DETAILS\_FAIL,

      payload:

        error.response && error.response.data.message

          ? error.response.data.message

          : error.message,

    });

  }

};

We can also get store state inside the action using getState (Optional)

export const getUserDetails = (id) => async (dispatch, getState) => {

  try {

    dispatch({

      type: USER\_DETAILS\_REQUEST,

    })

    const {

      userLogin: { userInfo },

    } = getState()

    const config = {

      headers: {

        Authorization: `Bearer ${userInfo.token}`,

      },

    }

    const { data } = await axios.get(`/api/users/${id}`, config)

    dispatch({

      type: USER\_DETAILS\_SUCCESS,

      payload: data,

    })

  } catch (error) {

    const message =

      error.response && error.response.data.message

        ? error.response.data.message

        : error.message

    if (message === 'Not authorized, token failed') {

      dispatch(logout())

    }

    dispatch({

      type: USER\_DETAILS\_FAIL,

      payload: message,

    })

  }

}

Step6: Subscribe store by component

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

import {

  Button,

  Card,

  Col,

  Form,

  Image,

  ListGroup,

  Row,

} from 'react-bootstrap';

import { useDispatch, useSelector } from 'react-redux';

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

import Rating from '../components/Rating';

import Loader from '../components/Loader';

import Message from '../components/Message';

import { listProductDetails } from '../actions/productAction';

const ProductScreen = ({ history, match }) => {

  const productDetails = useSelector((state) => state.productDetails);

  const { product, loading, error } = productDetails;

  const [qty, setQty] = useState(1);

  const dispatch = useDispatch();

  useEffect(() => {

    dispatch(listProductDetails(match.params.id));

  }, [dispatch, match]);

  const addToCartHandler = () => {

    history.push(`/cart/${match.params.id}?qty=${qty}`);

  };

  return (

    <>

      <Link className='btn btn-light my-3' to='/'>

        Go Back

      </Link>

      {loading ? (

        <Loader />

      ) : error ? (

        <Message variant='danger'>{error}</Message>

      ) : (

        <Row>

          <Col md={6}>

            <Image src={product.image} alt={product.name} fluid />

          </Col>

          <Col md={3}>

            <ListGroup variant='flush'>

              <ListGroup.Item>

                <h3>{product.name}</h3>

              </ListGroup.Item>

              <ListGroup.Item>

                <Rating

                  value={product.rating}

                  text={`${product.numReviews} reviews`}

                />

              </ListGroup.Item>

              <ListGroup.Item>Price: ${product.price}</ListGroup.Item>

              <ListGroup.Item>

                Description: ${product.description}

              </ListGroup.Item>

            </ListGroup>

          </Col>

          <Col md={3}>

            <Card>

              <ListGroup variant='flush'>

                <ListGroup.Item>

                  <Row>

                    <Col>Price :</Col>

                    <Col>

                      <strong>${product.price}</strong>

                    </Col>

                  </Row>

                </ListGroup.Item>

                <ListGroup.Item>

                  <Row>

                    <Col>Status :</Col>

                    <Col>

                      <strong>

                        {product.countInStock > 0 ? 'In Stock' : 'Out of Stock'}

                      </strong>

                    </Col>

                  </Row>

                </ListGroup.Item>

                {product.countInStock > 0 && (

                  <ListGroup.Item>

                    <Row>

                      <Col>Qty :</Col>

                      <Col>

                        <Form.Control

                          as='select'

                          value={qty}

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

                        >

                          {[...Array(product.countInStock).keys()].map((x) => (

                            <option key={x + 1} value={x + 1}>

                              {x + 1}

                            </option>

                          ))}

                        </Form.Control>

                      </Col>

                    </Row>

                  </ListGroup.Item>

                )}

                <ListGroup.Item>

                  <Button

                    className='btn-block'

                    type='button'

                    disabled={product.countInStock <= 0}

                    onClick={addToCartHandler}

                  >

                    Add To Cart

                  </Button>

                </ListGroup.Item>

              </ListGroup>

            </Card>

          </Col>

        </Row>

      )}

    </>

  );

};

export default ProductScreen;

1. Implement Paypal Gateway

npm I react-paypal-button-v2 (check documetation)

Load paypal

import { PayPalButton } from 'react-paypal-button-v2';

 const [sdkReady, setSdkReady] = useState(false);

useEffect(() => {

    const addPayPalScript = async () => {

      const { data: clientId } = await axios.get('/api/config/paypal');

      const script = document.createElement('script');

      script.src = `https://www.paypal.com/sdk/js?client-id=${clientId}`;

      script.type = 'text/javascript';

      script.async = true;

      script.onload = () => {

        setSdkReady(true);

      };

      document.body.appendChild(script);

    };

    if (!order || successPay || orderId !== order.\_id) {

      dispatch({ type: ORDER\_PAY\_RESET });

      dispatch(getOrderDetails(orderId));

    } else if (!order.isPaid) {

      if (!window.paypal) {

        addPayPalScript();

      } else {

        setSdkReady(true);

      }

    }

  }, [orderId, dispatch, successPay, order]);

Add paypal button

{!order.isPaid && (

                <ListGroup.Item>

                  {loadingPay && <Loader />}

                  {!sdkReady ? (

                    <Loader />

                  ) : (

                    <PayPalButton

                      amount={order.totalPrice}

                      // shippingPreference="NO\_SHIPPING" // default is "GET\_FROM\_FILE"

                      onSuccess={successPaymentHandler}

                    />

                  )}

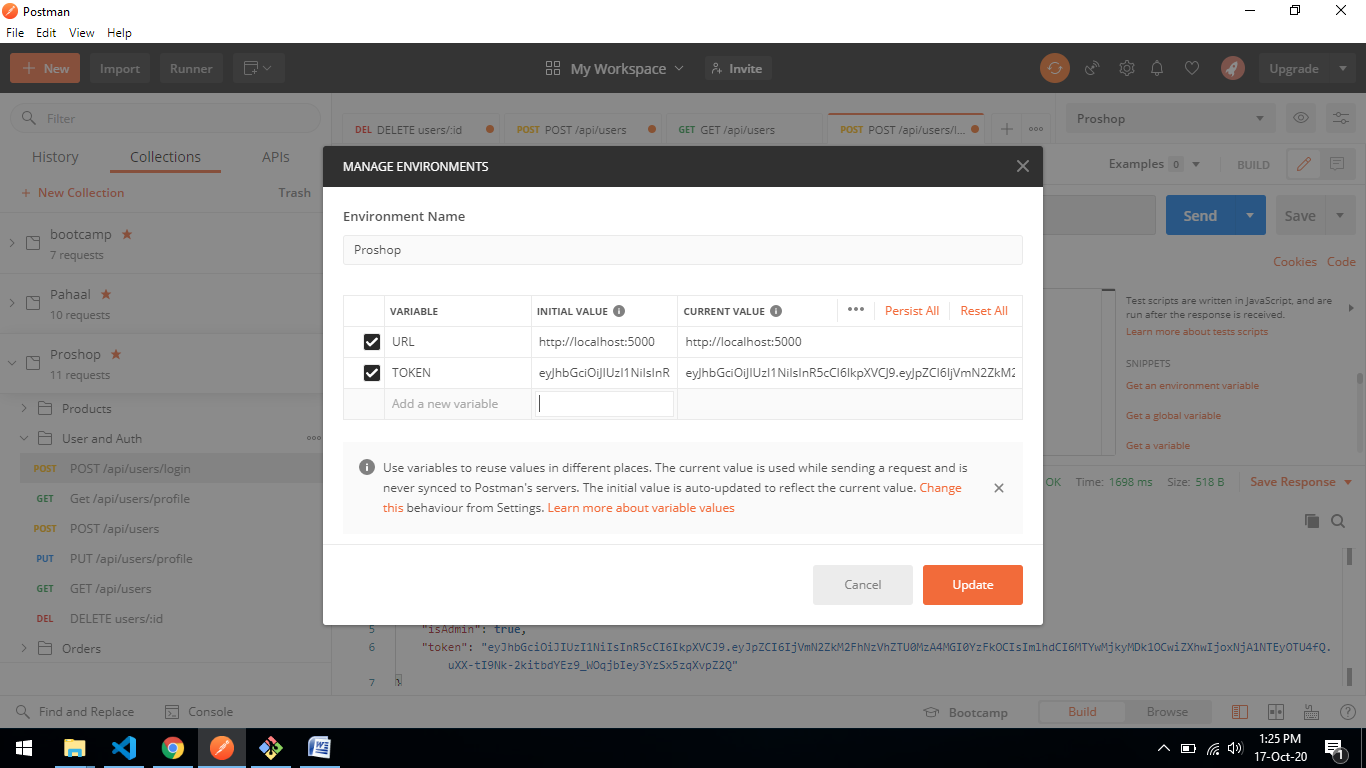
                </ListGroup.Item>

              )}

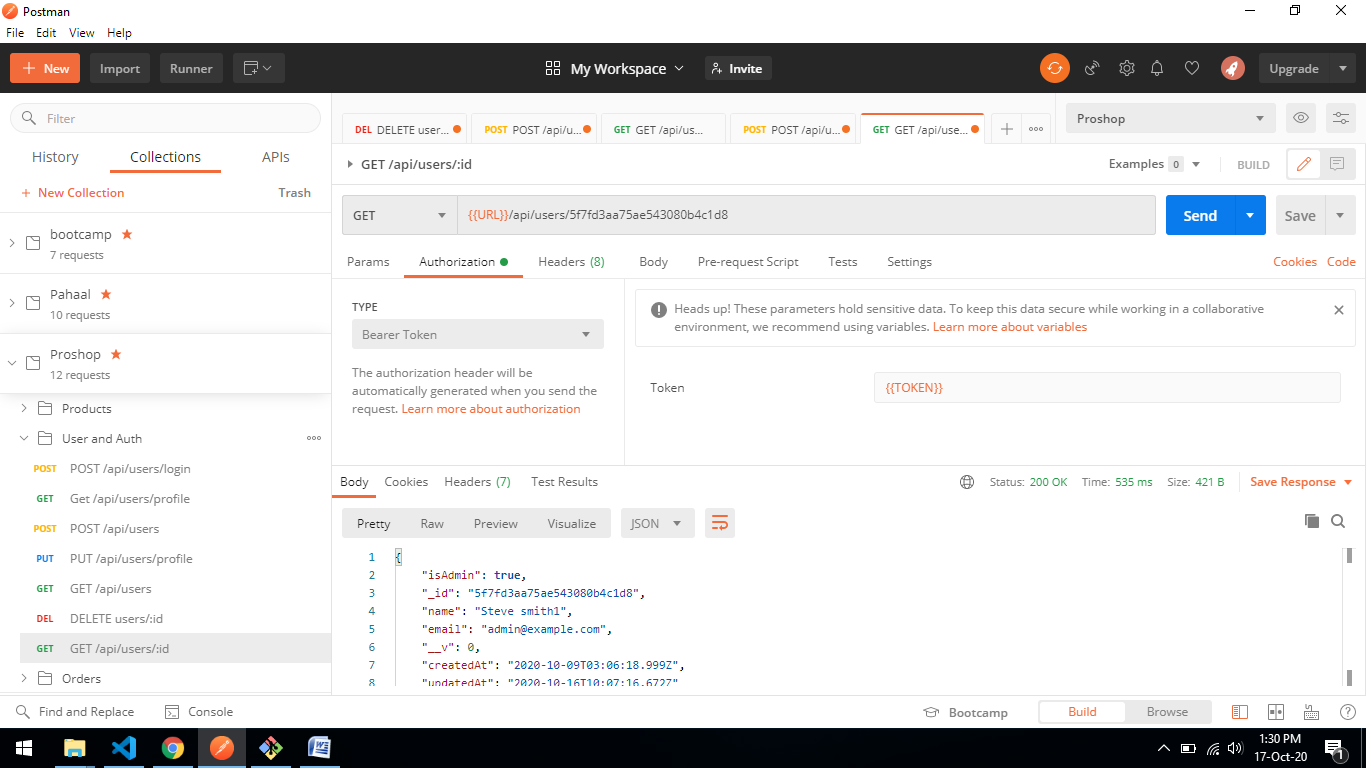
1. Tips and tricks for postman

Set this command inside postman tests tab. It automatically update token that we had set in environment variable.

pm.environment.**set**("TOKEN",pm.response.json().token)



Add token inside authorization



Error that I have faced yet

1) Api calling

‘api/users’ incorrect

‘/api/users’ correct

2) Order of api is important in backend or order of routes in frontend

\*backend

router.route('/myorders').get(protect, getMyOrders);

router.route('/:id').get(protect, getOrderById);

\*frontend

<Route path='/register' component={RegisterScreen} />

<Route path='/login' component={LoginScreen} />

<Route path='/' component={HomeScreen} exact />

Notice /myorders , /:id react same route

**IMP NOTE:** /:id must define at last

3) Always observe state of component inside useEffect, Because It is responsible for fetching data. If state of component is not change or reset then it will not call the fresh copy data.

4) If something goes wrong with render function then handle this error with add/remove code

5) use special operator of javascript

\* ?? nullish operator (It return second value if first value is null of undefined otherwise first value)

Const a = null ?? “aamir” //aamir

\* && if first value is true then return second value otherwise first value

Const a = true && “your are save” //you are save

\* || it check if first value is falsy value( null,undefined,0,’’,false) then return second value

const a = 0 || ‘test’ //test

6)when we get parameter from route it is in the form of string. If we want to match string with integer id then we have to convert string to integer

const Id = +this.props.match.params.id

or

const Id = Number(this.props.match.params.id)

7) Sometime by mistake we return different value from reducer to component.

Solution: Make alert to show state in reducer or inside component useSelector and mapStateToProps

8) component file name must start with capital letter

Eg:

CounterComponent.js valid

counterComponent.js invalid

9) custom hook start with use eg: useCounter

10)return array must be destructure with same as return

Eg: return [count,increment,decrement]

return const [count,increment,decrement] = useCounter();

File uploading n Nodejs using multer

\*create storage where we have to define file store location and file name pattern

\*and create middleware that accept two parameters = storage and filtertype

Server.js

import path from 'path';

import express from 'express';

import uploadRoutes from './routes/uploadRoutes.js';

const app = express();

const \_\_dirname = path.resolve();

app.use('/uploads', express.static(path.join(\_\_dirname, '/uploads'))); app.use('/api/upload', uploadRoutes);

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

app.listen(

  PORT,

  console.log(

    `Server running in ${process.env.NODE\_ENV} mode on port ${PORT}`.yellow.bold

  )

);

uploadRoutes.js

import path from 'path';

import express from 'express';

import multer from 'multer';

const router = express.Router();

const storage = multer.diskStorage({

  destination(req, file, cb) {

    cb(null, 'uploads/');

  },

  filename(req, file, cb) {

    cb(

      null,

      `${file.fieldname}-${Date.now()}${path.extname(file.originalname)}`

    );

  },

});

function checkFileType(file, cb) {

  const filetypes = /jpg|jpeg|png/;

  const extname = filetypes.test(path.extname(file.originalname).toLowerCase());

  const mimetype = filetypes.test(file.mimetype);

  if (extname && mimetype) {

    return cb(null, true);

  } else {

    cb('Images only!');

  }

}

const upload = multer({

  storage,

  fileFilter: function (req, file, cb) {

    checkFileType(file, cb);

  },

});

router.post('/', upload.single('image'), (req, res) => {

  res.send(`/uploads/${req.file.filename}`);

});

export default router;

Action function (get state inside function)

export const updateUserProfile = (user) => async (dispatch, getState) => {

  try {

    dispatch({

      type: USER\_UPDATE\_PROFILE\_REQUEST,

    });

    const {

      userLogin: { userInfo },

    } = getState();

    const { data } = await authAxios.put(`/api/users/profile`, user);

    dispatch({ type: USER\_UPDATE\_PROFILE\_SUCCESS, payload: data });

    dispatch({ type: USER\_LOGIN\_SUCCESS, payload: { ...userInfo, ...data } });

    localStorage.setItem('userInfo', JSON.stringify(data));

  } catch (error) {

    dispatch({

      type: USER\_UPDATE\_PROFILE\_FAIL,

      payload:

        error.response && error.response.data.message

          ? error.response.data.message

          : error.message,

    });

  }

};