1. First we will creata a Product component and will move the product related code there

2. Create a Rating Compnent and move the rating based code there

3. Add the Product Component in App.js and Rating component in product.js

<Product product={product} key={product.\_id}/> in App.js

<Rating rating={product.rating} numReviews={product.numReviews} /> in Product.js

4. **ADDING REDUX TO THE PROJECT**

FRST CREATE A STORE FOLDER AND IN THERE CREATE A REDUCER AND ACTION FOLDER

CREATE A STORE.JS FILE and in there

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

import thunk from 'redux-thunk';

import { productListReducer } from './reducers/Productreducer';

const initialState={}

const reducer = combineReducers ({ // COMBINING THE REDUCER

productList:productListReducer

})

const composeEnhancers = window.\_\_REDUX\_DEVTOOLS\_EXTENSION\_COMPOSE\_\_ || compose; // ADDING REDUX DEV TOOLS

const store= createStore( //CREATING A STORE

reducer,

initialState,

composeEnhancers(applyMiddleware(thunk))

)

export default store;

NOW IN CREATE A REDUCER.JS FILE IN REDUCER FOLDER

export const productListReducer = (

state = { loading: true, products: [] },

action

) => {

switch (action.type) {

case PRODUCT\_LIST\_REQUEST:

return { loading: true };

case PRODUCT\_LIST\_SUCCESS:

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

case PRODUCT\_LIST\_FAIL:

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

default:

return state;

}

};

NOW CREATE A ACTIONS FILE IN ACTIONS FOLDER WHERE YOU WILL SEND AJAX REQUEST

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

dispatch({

type: PRODUCT\_LIST\_REQUEST,

});

try {

const { data } = await Axios.get('/api/products'); //EXTRACTING DATA FROM THE REQUEST

dispatch({ type: PRODUCT\_LIST\_SUCCESS, payload: data });//SETTING PAYLOAD EQUALS TO data

} catch (error) {

dispatch({ type: PRODUCT\_LIST\_FAIL, payload: error.message });

}

};

NOW IN INDEX.JS FILE

import { Provider } from 'react-redux';

import store from './store/store';

AND WRAP <APP> WITH <PROVIDER>

NOW IN LANDING.JS FILE ADD THE REDUX

const dispatch = useDispatch(); //THROUGH THIS YOU CAN USE DISPATCH

const productList = useSelector((state) => state.productList); //used in the place of mapstatetoProps

const { loading, Error, products } = productList;EXTRACTING VARIABLES FROM DISPATCHED PRODUCTS

useEffect(()=>{

dispatch(listProducts());

}, [dispatch]); //SENDING REQUEST IN USEEFFECT

**Add Redux to The Products page**  
Create action for product details page

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

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

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

Then Create a function to send a request   
  
Pass the productid as a parameter and extract data about that product

export const ProductDetails =(productId)=> async(dispatch)=>{

  dispatch({

    type:PRODUCT\_DETAILS\_REQUEST,

    payload:productId

  })

  try{

    const {data}= await Axios.get(`/api/products/${productId}`) //Extracting data of the product

    dispatch({

      type:PRODUCT\_DETAILS\_SUCCESS,

      payload:data

    })

  }catch(err)

  {

    dispatch({

      type:PRODUCT\_DETAILS\_FAIL,

      payload: err.response && err.response.data.messsage

      ? err.response.data.message

      : err.message

    })

  }

}

Then in Reducer.js File the function is same as it was for Landing page.js  
  
  
Now in the main Product.js page   
  
  
Add the Redux there by using useSelector and UseDispacth

const productId =  props.match.params.id //Extract id from the params

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

 const dispatch =useDispatch();

 const {product ,Error , loading} = productDetails;

 useEffect(()=>{

     dispatch( ProductDetails(productId)) //Pass the product id

 },[productId,dispatch])

After this add the reducer in the store too in the combine reducer

const reducer = combineReducers ({

    productList:productListReducer,

    productDetails: productDetailsReducer,//Add this in the combine reducer

Now we have a product detail page but right now we don’t have the option for the add to cart and the quantity choose.   
  
For the Quantity and the button   
  
In this code add a select method in there we have to convert the object of CountInstock into an array it will give the array from 0-n and then we have to add the 1 into the values to make it simple

Add AddTocartHAndler there  
use history.push method to redirect to cart

And also initialize Qty using Setstate

 const AddtoCartHandler=()=>{

        props.history.push(`/cart/${productId}?Qty=${Qty}`)

    }

                product.countInStock > 0    &&  (

                    <div>

                        <li>

                            <div className = 'row'>

                                <div>Quantity</div>

                                <div>

                                    <select className='select' value={Qty}

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

 {[...Array(product.countInStock).keys()].map( //This function will return an array from 0-4 if the countinStock=5

                                (x) => (

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

                                    {x + 1}

                                  </option>

                                )

                              )}

                                    </select>

                                </div>

                            </div>

                        </li>

                     <li>

                    <button

                    onClick={AddtoCartHandler}

                     className="primary block">Add to Cart</button>

                   </li>

                   </div>

                )

            }

AT THIS STAGE YOU HAVE A NICE LANDING PAGE AND A GOOD LOOKING PRODUCT DETAILS PAGE  
  
NOW LETS MOVE TOWARDS CART

**CART ADDING REDUX**

IN THE ACTION FOLDER CREATE A ACTION FOR CART IN THERE SEND GET REQUEST AND EXTRACT DATA FROM THERE IN

PAYLOAD PASS THE NAME, PRICE AND OTHER ATTRIBUTES

The Actions Folder for the Cart contains actions.js and the CartActions.js  
  
Actions.js

export const CART\_ADD\_ITEMS='CART\_ADD\_ITEMS';

Now in the CartActions.js   
  
We are taking the parameters productid and the Qauntity and we are using getState in order to get the current status of the state so that we can compare the items in the cart

export const CartAddItem=(productId,Qty)=>async(dispatch,getState)=>{   //we are getting getState to update the cart

    const {data}= await Axios(`/api/products/${productId}`); //extracting the data of the Product

    dispatch({

        type: CART\_ADD\_ITEMS,

        payload:

        {

            name: data.name,

            image:data.image,

            price:data.price,

            countInStock:data.countInStock,

            product:data.\_id, // we are using product instead of productid here for the database

            Qty

        }

    })

Right Now we have the name,image,price,count and id in the payload of the added Item

Now Lets Move to the Reducer Folder there create a File of CartReducer.js

In there first set item to the payload and then check if the item already exist if yes then update it else add the the item

export const CartReducer=(state = { CartItems: [] }, action)=>

{

    switch(action.type)  {

        case  CART\_ADD\_ITEMS:

        const item = action.payload;

     const existeditem= state.CartItems.find((x)=> x.product === item.product)

        if (existeditem) {

            return {

                ...state,

    CartItems: state.CartItems.map((x)=>  x.product   === existeditem.product ?   item:   x)   //If the item exist then copy the state and in

                //cart update the existing product

        }}

        else{

            return {

                ...state,

                CartItems:[...state.CartItems,item] //this will concatenate the two elements as it will add the elements to the existing elements in cart

            }

        }

        default:

                    return state;

    }

}

Now Lets Create the Cart Page   
In there we have to extract the productid and the Qantity from the params which was passed in the Product.js Page AddTocart handler()

const dispatch=useDispatch();

    const productId = props.match.params.id;

    const Qty   =   props.location.search  //This will search for the qty in the url

     ?   Number(props.location.search.split('=')[1])  //this will get the value of qty which is after = sign in the url

     :   1;

     //We want to check if the productu id exist then add to cart that product

    const Cart= useSelector((state)=>state.Cart)

     const {CartItems}= Cart;

     useEffect(()=>{

         if(productId)

         {

         dispatch(CartAddItem(productId,Qty))

     }},[dispatch,productId,Qty])

Now that we have the quantity and the product id but we have to store the cartITems locally on the browser so that it does not get lost whenever we refresh the page. For that   
  
Add this in the CartAction.js file inside the CardAddItem funtion

localStorage.setItem('CartItems',JSON.stringify(getState().Cart.CartItems))

}

STORE.JS

Now move into store.js file   
  
const initialState={

    Cart:{

        CartItems: localStorage.getItem('CartItems')

        ?   JSON.parse(localStorage.getItem('CartItems')) //Converting the cart into an array

        : []

    }

}

And ADD the Cart reducer in the combine Reducer Function  
  
  
Right Now we have all the functionalties of the Cart and have all the data stored on the localstorage now we have to design the Cart Page   
  
The below code will update the price and paste the other code from the product screen

  </div>

            <div className="col-1">

        <div className="card card-body">

          <ul>

            <li>

              <h2>

                Subtotal ({CartItems.reduce((a, c) => a + c.Qty, 0)} items) : $

                {CartItems.reduce((a, c) => a + c.price \* c.Qty, 0)}

              </h2>

            </li>

            <li>

              <button

                type="button"

                onClick={CheckoutHandler}

                className="primary block"

                disabled={CartItems.length === 0}

              >

                Proceed to Checkout

              </button>

**Delete From Cart**Create a function of Delete Item in the cartaction.js file where just get the product id and get the localstorage stored on the browser

export const CartRemoveItem =   (productId)=> async(dispatch,getState)=>{

    dispatch({

        type:CART\_DELETE\_ITEM,

        payload:productId

    })

    localStorage.setItem('CartItems',JSON.stringify(getState().Cart.CartItems));

}

We will always use filter() function when we have to delete something

case CART\_DELETE\_ITEM:

            return{

                ...state,

                CartItems:state.CartItems.filter((x)=> x.product    !== action.payload)  //This will filter out that element which has the same product id

            }

**In the Cart.js Page**

 const removeCartHandler=(id)=>{

         dispatch(CartRemoveItem(id))

         console.log('Deleted',id)

     }

**BACKEND**

Create a Model and Routes Folder and in the models folder create a user.js file there add name,email and password all the user’s info and then export it   
  
Create another file product.js and insert product’s info there  
  
  
In the index.js file   
  
Require those routes and connect to the mongo db

mongoose.connect(process.env.MONGODB\_URL || 'mongodb://localhost/artgallery', {

  useNewUrlParser: true,

  useUnifiedTopology: true,

  useCreateIndex: true,

}

).then(() => {

    console.log('Database connected');

});

Create routes there

app.use('/api/user', userRoutes);

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

Now create the routes folder there add the userRoutes and placesRoutes files in them check for the   
dummy data

**DUMMY**

router.get('/seed',asyncexpressHandler(async(req,res)=>{

    //await Product.remove({})

    const createdProduct= await Product.insertMany(data.products);

    res.send({createdProduct})

}))

**For An Specific Product with an id**

router.get('/:id',asyncexpressHandler(async(req,res)=>{

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

    if(product)

    {

        res.send(product)

    }

    else{

        res.status(404).send({message:"Product not found"})

    }

}))

**HOME**

router.get('/seed',asyncexpressHandler(async(req,res)=>{

    //await Product.remove({})

    const createdProduct= await Product.insertMany(data.products);

    res.send({createdProduct})

}))

**SIGN IN**

First we will find the email if the email is found then we will check if the passwords match using bcrypt.compareSync if the password match then we will send the res.send (user.id,user.email,user.name,user.isAdmin and the user.token ) We will generate token for every user token is used to track the sign in time the user is signed into the website. For the token we will use json web token.

router.post(

    '/signin',

    expressAsyncHandler(async (req, res) => {

      const user = await User.findOne({ email: req.body.email });

      if (user) {

        console.log("user",user)

        if (bcrypt.compareSync(req.body.password, user.password)) {

          res.send({

            \_id: user.\_id,

            name: user.name,

            email: user.email,

            isAdmin: user.isAdmin,

            token: generateToken(user),

          });

          return;

        }

      }

      console.log(req.body)

      res.status(401).send({ message: 'Invalid email or password' });

    })

  );

Now we have to generate the token using json web token for that create a new folder util.js and in there create a new file of util.js in there. In there create a function of generateToken in there return jwt.sign () we have to create a token with the user info so pass that in the function and add a secret key with that then add the expiration time in which the token will be expired in . Now import that function in userRoutes.js file

const jwt= require('jsonwebtoken')

const generateToken=(user)=>{

    return jwt.sign(

        {

        \_id: user.\_id,

        name: user.name,

        email:user.email,

        isAdmin: user.isAdmin,

        },  process.env.JWT\_SECRET || 'somethingsecret',

        {

        expiresIn:'1d'

    })

}

module.exports = generateToken

SIGN IN SCREEN

In Pages folder create a signin.js file in there create a form with email and password and a submit button

 <div>

      <form className="form" onSubmit={submitHandler}>

        <div>

          <h1 className='H1'>SIGN IN</h1>

        </div>

        {loading && <Spinner/>}

        {error && <MessageBox variant="danger">{error}</MessageBox>}

        <div>

          <label htmlFor="email">Email address</label>

          <input

            type="email"

            id="email"

            placeholder="Enter email"

            required

            onChange={(e) => setEmail(e.target.value)} //Wil set the value of email to the entered value

          ></input>

        </div>

        <div>

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

          <input

            type="password"

            id="password"

            placeholder="Enter password"

            required

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

          ></input>

        </div>

        <div>

          <label />

          <button className="primary" type="submit">

            Sign In

          </button>

        </div>

        <div>

          <label />

          <div>

            New customer? <Link to="/register">Create your account</Link>

          </div>

        </div>

      </form>

    </div>

  );

}

Now you have a sign in screen but you cant signin right now you have to add sign in in redux

For that create a new folder Signin in the actions folder in there add a action.js file and a signin.js file

Action.js

export const SIGN\_IN\_REQUEST    =   'SIGN\_IN\_REQUEST';

export const SIGN\_IN\_SUCCESS    =   'SIGN\_IN\_SUCCESS';

export const SIGN\_IN\_FAIL    =   'SIGN\_IN\_FAIL';

export const SIGN\_OUT   =   'SIGN\_OUT';

Now create a Signin.js file in action folder where you will do a post request with email and password if the request is success then extract the data and store it locally on the browser

Else show an error .

export const SignIn=(email,password)=>async(dispatch)=>{

dispatch({

    type:SIGN\_IN\_REQUEST,

    payload: {email,password}

})

  try{

      const {data}= await Axios.post("/api/user/signin", {email,password}); // data contain the id token email and password of the user

dispatch({

    type:SIGN\_IN\_SUCCESS,

    payload: data

})

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

}catch(error)

{

    dispatch({

        type:SIGN\_IN\_FAIL,

         payload:

        error.response && error.response.data.message  //payload is checking if the err.response exist if it does then show that else display err.message

          ? error.response.data.message

          : error.message

})}}

Create a reducer file of SigninReducer.js in there

export const SigninReducer=(state={}, action)=>{

    switch(action.type)

    {

        case SIGN\_IN\_REQUEST:

            return {loading:true};

        case SIGN\_IN\_SUCCESS:

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

        case SIGN\_IN\_FAIL:

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

        case SIGN\_OUT:

            return{};

        default:

            return state;

    }}

Now add the Sign in Into the Signin page

const submitHandler = (e) => {

    e.preventDefault(); //Will not refresh the page after clicking the signing button

    dispatch(SignIn(email,password))

  };

Now you can signin to the web now in App.js file show the Signin link if you are not authenticated else show Signout for this you have to access the userInfo check if userInfo exist show Sign in else show Signout

const SignIn= useSelector((state)=> state.Signin);

      const {userInfo}  = SignIn;  //userInfo name should be same as in the reducer

{userInfo ? (  <Link to="#signout" onClick={SignOutHandler}>

                      Sign Out

                    </Link> ) : (

              <Link to="/signin">Sign In</Link>

            )}

Now you can Signin and the Sign in and Signout will be displayed add the Sign in reducer to store.js file to and do the steps as were done in the cart for the local storage

**STORE.js**

 Signin:{

  userInfo : localStorage.getItem('userInfo')

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

  :[]

  },

**Now lets Implement Sign Out**

In Signin actions file create another action for Signout in there remove the items from the localstorage of both cart and signin

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

localStorage.removeItem('userInfo')

localStorage.removeItem('CartItems')

dispatch({

    type: SIGN\_OUT

})}

Now add the Signout () in the Sign in page

 const redirect = props.location.search

  ? props.location.search.split('=')[1]

  : '/';

useEffect(()=>{

    if(userInfo){

      props.history.push(redirect);

    }

  },[props.history,userInfo,redirect])

This will redirect the user to the shipping if he clicked proceed to checkout after adding the products to the cart else it will redirect the user to the home page

Now In **APP.js file**  add the Signout handler

const SignOutHandler=()=>{

    dispatch(SignOut());

  }

**Place Order Backend Model**

We will create a models of order in which we store the info if the order ,where the product id will be used from the product. We store the shipping info and payment . We also store the user id from user and delivery and status.

Create a order router profile there create an order for the user we have to extract it from the authorization headers. Create a middle ware to extract the user id from the header

The authorization is in the form like Bearer Token we have to check of the headers has token then we will extract the info of user from the token using jwt verify

const Auth =(req,res,next)=>{

    const authorization = req.headers.authorization;

    if(authorization){

        const token = authorization.slice(7,authorization.length)

        jwt.verify(token,process.env.JWT\_SECRET || 'somethingsecret',

        (err,decode)=>{     //decode cintains the user info

            if(err)

            {

                res.status(401).send({message :"Invalid Token"})

            }

            else{

                req.user=decode;

                next();

            }

        })

    }

    else{

        res.status(401).send({message:"No token found"})

    }

}

Now in the rotes file make the post request

router.post('/orders',

Auth,

asyncexpressHandler(async(req,res)=>{

    if(req.body.orderItems.length === 0)

    {

        res.status(404).send({message :"Cart is Empty"})

    }

    else{

    const order = new Order({

        orderItems:req.body.orderItems,

        Shippinginfo : req.body.Shippinginfo,

        PaymentMethod: req.body.PaymentMethod,

        ItemsPrice: req.body.ItemsPrice,

        ShippingPrice: req.body.ShippingPrice,

        TotalPrice: req.body.TotalPrice,

        user:req.user.\_id                   //We are extracting id from teh middleware Auth

    })

    const createrOrder =await Order.save();

    res.status(201).send({message:"New order created " , order:createrOrder})

}})

)

Error face of Router.post() got a string this is because I forgot to add require in index.js while importing oredeRoute. Next error was faced I was importing {isAuth} like this and it should be like isAuth.