## Part 01:

"start": "PORT=2000 react-scripts start",

A picture containing shape

Description automatically generated

Created a basic template with index.html & style.css.

## Part 02: Install tools

## Part 03: Design web template:

A picture containing shape

Description automatically generated

## Part 04: Display Products

Graphical user interface, application

Description automatically generated

Create Index.html., style.css file and created basic UI Design. i.e implemented a basic list of products using html tag i.e 5 div tags.

Graphical user interface, application, Teams

Description automatically generated

## Part 05: Create React App

Create basic react app and move main div tag of index.html to react application, at the end of this lesson we are able to see product list from static js array.

To create react application Npx create-react-app frontend

Cd frontend

Npm start

Copy paste content of body tag into app.js file. Remove certain files-app.css, logo-svg , copy style.css and paste it under body of index.css.

Copy images folder to public folder under frontend

All data in App.js is static data- hard coded 5 times to display 5 elements, and we need to load data from js array. To do that:

Go to src folder: create data.js file (which is source of products)

Data.js: recording all details of products

Text

Description automatically generated

Update App.JS file, and instead of having 6 card, we want to have only one cart div, we use map function in app.js file to convert data.js product items to html elements. And convert static name to product.name.

Hence just keep the first div and remove all other div’s.

We import data from data.js: import data from ./data

Text

Description automatically generated

Using map:

Data is a object, products is array of products stored in data.js file.

Text

Description automatically generated

So basically we load data of all the items from a javascript array into the main div, and use map function which iterates for n time(= no of products i.e for each product from products invoke everything written under () of map(()=>()))

## Part 07: Create Product components

What are components in react: [React Components (w3schools.com)](https://www.w3schools.com/react/react_components.asp)

The main concept of react is to use components- basically reuse code: It is a good practice to to write all the components into a component folder and use it whenever needed.

Graphical user interface, text, application

Description automatically generated

So basically we make the code more clean and reusable. So we copy the content of div in map function and paste in component file called - product.js. so we can now invoke the entire content of product.js file as <product></product>, also to display the content of each product object we have to pass an object as: product={product}, is an product object that stores 1 product

Graphical user interface, text

Description automatically generated

We write the code to display product inside react functional component:

Graphical user interface, text, application, chat or text message

Description automatically generated

How does the react component determine which value to view ? so for this we need to pass props

A screenshot of a computer screen

Description automatically generated with medium confidence

## Part 08: Create HomeScreen, ProductScreen, add routes from product details to app.js

Graphical user interface, text

Description automatically generated

Go to frontend fol: npm install react-router-dom

By installing router dom we can define multiple routes for our application and for each route specific screen response will be shown to user.

To implement route go to app.js and wrap the whole content inside return function to <Browser router> tag and we are ready to define routes

So 1st route is for homescreen and the components that respond to this is{Homescreen} when we set exact attribute it means, only if url=”/” then render this element.

Route 2: for product screen



When user enters address with this format we will render product screen. Id: contains the id of the products



Create a new screen folder and add all screens under screens folder.

Now how does routing work?

For every product image we have specified <a href =”product/${product\_id}”> hence this means when we click on this link - url will be localhost:3000/product/3. Here we get value=3 from product.\_id .

Now when we write this: 

It means when you are using url as mentioned above, it will redirect us to product screen component.

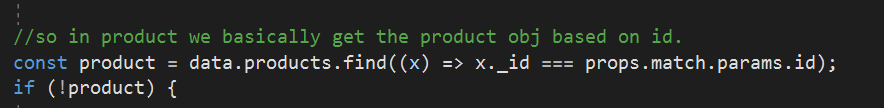
Eg:

Graphical user interface, text, application

Description automatically generated

Now for all the product id be it 1/2/3.. we get this same page. Now how do we ensure that particular product page opens up when we click that product : we write this code in product screen.

We fetch the product from array of products whose id is mentioned by user in the url. Now using this product object we can display all of its details on the screen.



find is an array function that finds an element based on the criterion that we define.

here we are searching for x.\_id which is the id for inside products array and comparing it with props.match.params.id

props.match.params.id = the value that user enter in id in route path="/product/:id" in URL

we compare this value with items in products array

Graphical user interface, text, application, email

Description automatically generated

then in return() we write the code to display details of particular product.

## Part 09:

## Here we start building Backend, we create a very basic node.js application that build an api for us that returns list of products to frontend – server.js

Text

Description automatically generated

So do npm init in amazone-clone folder, once we do that we will have package.json file in amozone-clone .

Next -Create a backend folder:

create server.js,it is the entry point of our application here we create an express server, express is a node package that helps creating a server very easily and powerful, install express: npm install express. This will add dependencies section and express will appear in package.json

In server.js-> import express, and create an app for express

Text

Description automatically generated

To open this:

Type in terminal-> node backend/server.js

Click on log link and u see :

A picture containing text

Description automatically generated

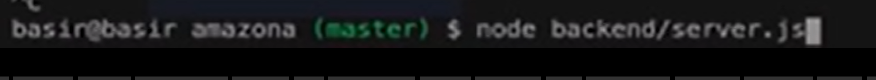
We successfully created our first server using node and express

Copy data.js folder from src to backend folder. So now we have all the products in backend.

Now in server.js we add a route: app.get(‘/api/products’) and its body we return data.products – list of products

So in frontend when we hit on api /products this route activates and sends list of products from data.js

To see this, Go to frontend:

Start the server 

In the URL, type: Graphical user interface, text, application

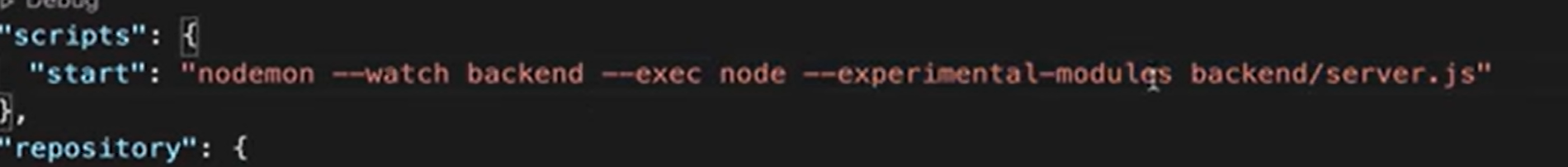
Description automatically generated

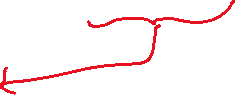
Result:

Text, letter

Description automatically generated

Install nodemon- to automatically re run your application when there is a change in your code (hende we wont have to re run server every time from terminal)





So when there is a change : this command will run.

## Part 10: Load products from Backend:

Text

Description automatically generated

Here we fetch products from backend, we have ajax request at \api\products and this return list of products to us. We have used this api using axios library. Have also created loading box component to show loading before fetching data from backend.

Step 1: Setting proxy to package.json file of frontend folder, so when you send a request to proxy it will be proxied/redirected to backend.

2. Install Axios. It is a library to send ajax request to backend.

3. And in home screen instead of fetching data from data.js from frontend we are fetching data from backend i.e /api/products .

4. Define react hook. To define hook state we use usestate, this hook is used to manage the state of this react component



Products is an empty array, and when we want to change value of products we use setproducts

We fil this products using useffect function .

It is another hook that happenes when your component didmount your webpage- that is after rendering the components this function will run.

It accepts 2 parameters. 1st : function , 2nd: array

In useEffect we send request to backend and fetch all the details.

Graphical user interface, application

Description automatically generated

This array accepts list of dependencies, by having this and after rendering your component this function will run only for 1 time and that is what we want – so here we will send ajax req to backend and fetch details from backend

This func is an assync fun becoz ajax req is an assyn req and in the body of this request we can fetch data from backend.

Graphical user interface, text

Description automatically generated

By passing this fun array in backend will be transferred to data in frontend.

Next step is to setting products that we are getting from backend.

Hence products array contains data from backend. And then we call getchData()

// You fixed your own error finallyyyyyyy. Fixed proxy settings 😊.

Here able to load data from backend

Application

Description automatically generated with medium confidence

Graphical user interface, application, Teams

Description automatically generated

Now we add loading component, initially we set loading as false as by default we are not loading anything.

We are going to use setLoading and set it true before sending ajax req (axios.get), and set it false after loading .

We use one more hook for error, we set the error to false by default as there are no error. To catch error we wrap the ajax req in try catch block. So in case if there is any error in ajax req. we catch it in catch block and set setError(err.message) and loading false

## Part 12: Add Redux to HomeScreen: Text Description automatically generated A picture containing text, clock Description automatically generated

Till now we have only implemented the View component. In the view section we are using view component. In prev lessons we have sent ajax request to backend to get list of products, but this time we are going to use redux to get things from backend and put them in redux state instead of component state.

So we dispatch an action in view part, and this action sends a request to redux store to make a change in the state of our application.

To make a state change we need to make an action and dispatch that action. In redux store we have 2 things- a state and a reducer

State: current state of our application

Reducer : is a function that takes current state make a change in state and return new state.

1st step to creating a redux is creating a store : store.js in src folder.

To create store: we need 2 things: 1 . Initial state 2. Reducer.

Text

Description automatically generated

To use redux store we need to go to index .js, which is the main entry point of our react application, here we wrap whole application inside provider and set store ad an property to this provider.

We need to add redux extensions to see redux store on inspect window.

This is what we get , when we add redux extension: No store found, to have reduc store we need to add some code in store.jsA screenshot of a computer

Description automatically generated

Install redux thunk using npm install redux-thunk in frontend.

Redux-thunk makes it possible to send ajax request to our redux actions, add it to your store.

Next step is adding redux to chrome developer tools, to show redux dev tool we need to update compose function

Text

Description automatically generated

Hence we define compose Enhancer, if it does not exist we use compose from import statement.

So when you will refresh you will be able to see redux store and its state A screenshot of a computer

Description automatically generated

## Text Description automatically generated

We can see list of products as reducer has returned them

### Defining product constants:

Now if you open a homescreen what we are gonna do is fetch products data from backend instead of having ajax request in component we are going to move it to action

So to create an action you need to create some constants. So under src folder create a new folder- constants and inside constants define productConstants.js

Inside productConstant.js create 3 constant , the reason to define 3 constants for getting list of products from backend is because of the async nature of ajax request

Graphical user interface, text

Description automatically generated

### Defining product actions:

Create another folder called actions , a new file productActions.js – here we define our very first action

Name of this action is ListProducts and action is a function, that returns another asyn function and accepts dispatch as a parameter, dispatch will be filled by redux think later. In the body of this function we need to dispatch product list dispatch

Dispatch accepts an object, that has a type and we will set the type to PRODUCT\_LIST\_REQUEST, which we need to import it.

Next step is fetching data from backend, which we wrap it into try catch block- so if there is any error while fetching details from backend we will dispatch PRODUCT\_LIST\_FAIL

Text

Description automatically generated

after fetching data from backend we dispatch another action, by dispatching another action we change the state of redux and based on that we update the home Screen and show products.

To dispatch an action we set the type of it and for sure type of this action is set as success and add payload – which contains data from backend.

Text

Description automatically generated

We implemented the listProduct actions , and by creating this actions we need product reducers to respond to these actions.

### Defining productReducers

Create a new folder called product Reducers, and file productReducers.js

Define a productListReducer, it’s a function that accepts 2 parameters state and action.

In the body of reducer we have a switch case and inside the switch we type action.type

Here we send loading:true in case: PRODUCT\_LIST\_REQUEST. Because when I dispatch , I am sending this ajax req to backend and waiting for response, and for default case we will return the current state, that is don’t change state at all and return previous state,

For PRODUCT\_LIST\_SUCCESS , we return : loading:false and set products to action.payload

Because in product action when we dispatch product\_list\_success, we set payload to data that we get from backend.

Hence in productListReducer we fetch data from which is in redux store by the data that we get from backend

We set the initial state to default one that is products =[], if we don’t mention this state it will show an error bcoz u want to products in hs, it should not be null, it should be an empty array

combineReducer(): will combine all the switch cases:

This is how based on the action and the type that we mention in action a reducer will be called.

Text

Description automatically generated

Diagram

Description automatically generated

Using dispatch() from homeScreen, we call an action which based on the type will call a reducer. And this reducer will store the state info of productList in redux store:

To display products on home screen:

Text

Description automatically generated

This function calls the list Products function:

A screenshot of a computer

Description automatically generated with medium confidence

Here we fetch data from backend, and add it to payload to store it in redux store.

Based on the type of dispatch : a reducer is called.

A screenshot of a computer

Description automatically generated with medium confidence

Here on PRODUCT\_LIST\_SUCCESS: we set the products in redux.

Diagram

Description automatically generated

Text

Description automatically generated

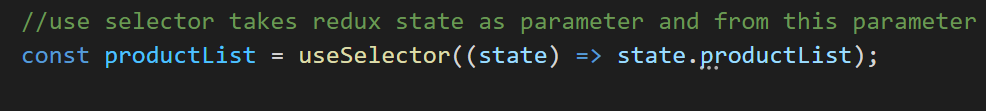
Here we initialize the productList to default value of state set in product reducer i.e:

Products: {}

Loading: true

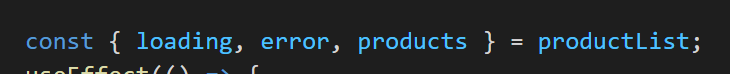
Error: null

Now we have everything stored in redux store: so we can fetch data from redux to component using useSelector.



Here from the state we extract only the value of product List.

And using value of productList , we fetch all three values.



And now we can use these products, error and loading on the frontend to display products. Based on loading and error.

## Part 13: Add Redux to Product Screen:

Text

Description automatically generated

## Defining constants:

Graphical user interface, text

Description automatically generated

So what happens is when on homescreen: we click on particular product , URL of that product is set to <a href=’product/${product.\_id}’></>

Text

Description automatically generated

props.match.params.id = the value that user enter in id in route path="/product/:id" in URL

hence the productScreen comes into the action:

here we first get the product\_id



Here we dispatch action to display detailsProduct and pass productid to get details for particular product:

Text

Description automatically generated

This will call up: detailsProduct()

Using this function we write an api to fetch details from backend for particular productId.

Text

Description automatically generated

From server.js using this API, we basically fetch the id from URL and find product with that id in data and save that product in product var and the backend sends the response of this data as response.

Text

Description automatically generated

After we receive data in frontend, we call dispatch({type: PRODUCT\_DETAILS\_SUCCESS, payload: data})

This in turn calls reducers : and sets data to payload to redux store.

Text

Description automatically generated

Now to view details from reducer: we use useselector in productScreen:

Text

Description automatically generated

So we had set values of productDetails using useSelector initially to :

products:

loading:

error:

Text

Description automatically generated

as per defined in reducer default state:

but using useEffect we add data to productDetails: and fill up all 3 values

products:

loading:

Graphical user interface, text, application, chat or text message

Description automatically generated

Hence we can fetch entire productDetails state on productscreen, and fetch all 3 values that

product: which has data for particular product. & 2 other.

Text

Description automatically generated

Now using product, we can display all the values on frontend.

## Part14: Handle Add to cart item:

Graphical user interface, text

Description automatically generated

We display add to cart button only if that particular product is available.

We add select box to select no of items :

For this we also create qty and setQty states to store values entered by the user

To display options: we use javascript array to view quantity upto count in stock

Doubt:

Graphical user interface, text

Description automatically generated

So if countInStock is 5, this funct returns value from 0 to 4.So we map each item to option and set value of option as x+1 (to display 1 to 5 instead of 0 to 4)

### Create Cart Screen:

Text

Description automatically generated

To get qty we need write props.location.search.

If props.location.search. does not exist we return 1.

If exists we split the url at = and take only the qty value present after ?

Add route of cart screen to app.js.



? means that mentioning id is optional , that is if we click on cart without id , still it should show up the cart.

## Part 15: Implement Add to cart:

Here we add cart items to redux store

Text

Description automatically generated

Dispatch and getstate are parameters in redux thunk, that makes it possible to dispatch an action and get access to that state of redux store.

### Defining CartActions:

Text

Description automatically generated

### Defining CartReducers:



We define initial state as empty cartItems.

Text

Description automatically generated

Find() finds product based on the criterion we decide. So here from all the item present in state.cartItems, if id of any product is same as id of product that I’m gonna add. Then return that product.

So if we already have product with this Id, we replace that item – existItem of cartItems with item we are gonna add (coz this one is knew and has updated value of quantity).

To replace we use map function:

…state means – we are not gonna change all the properties

And then we update cartItems : to update a val of one specific prod we use map function, and change item that already exists.

So using map if same product exists in cartItems we update it with ->item else return the old one-> x.

And if we are adding items for first time then we concatenate the new item with cartItems.

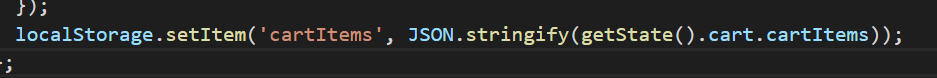
### Dispatch addToCart() in cartScreen:

Text

Description automatically generated

If productid exists, we call addToCart () and pass quantity and id

Also we change the <a> to <Link> of react router in product.js.



This will basically store your cart item to local storage hence when you refresh products would still persist.

Text

Description automatically generated

So in store.js we set the initial state as cart : cartItems (from local storage) cartItems exist in loacalstorage then we parse it and convert it to Array.

(note we use the same cartItem name that we used during setting the value).

### To show no of items besides cart icon:

So in app.js we use useselector to load cartItems that we had stored in redux . and len of cartItems would give us no items.

Text

Description automatically generated