Amazona Ecomerce app -🡪Details

1. Instalation of the react app .
2. Home Screen

* List products 🡪 inside frontend folder 🡪in src🡪 create data.js and inside that create a object name data which contains details of the products.

Then in the App.js file in the main section we are importing the data.js file and using map function traverse through the object and display …

Apply css to that products name ..and used <a> tag in images and names.

* Add Routing 🡪

First need to install the react router dom (npm i react-router-dom)

Then import BrowseRouter in the App.js page..

Then create a Screens folder 🡪 then HomeScreen.js🡪

Inside that move all the content of the Products that was previously present in app.js then using route we can get the deltails of the HomeScreen.js file.

Then we added ProductScreen in App.js and inside ProductScreen we use UseParams HOOK to get the ProductSlug value using params hook in react router-dom.

To resolve the page refress issue when moving from one page to another we have to use the Link tag of react instead of using the <a> tag.

Change the <a> tag in present in the HomeScreen to <Link> tag of react and href 🡪to

* Create NodeJS Server 🡪

Now we are creating a new folder Backend to perform the backend operations

Install the NodeJS in it (npm init) and then update the package.json file {type : “module”}

Now, we have to create server.js in backend folder..Then install express dependencies..

Then we have to copy the data.js file from the frontend folder to the backend folder

And install nodemon to auto compile and load the file.

* Fetch Products From Backend 🡪

1. Set proxy in package.json – in the frontend folder..

This proxy is done to get access of the backend in the frontend

1. npm install axios : To get data from backend we ar using axios libraby… to know more about this axios visit([axios-http.com](https://axios-http.com/docs/intro)).

Install the axios in the frontend folder

1. use state hook
2. use effect hook
3. use reducer hook

* **Manage State By reducer HOOK**

Here we are replacing useReducer in state of useState bcz when we want to send an ajex request to backend state is complex and nextState depends on the previous one

Link for useState ,useReducers -🡪 <https://reactjs.org/docs/hooks-reference.html>

🡪action.payload contains all produts from backend

🡪To define useReducer we need to define an array that contains two values the first one is object and the second parameters are dispatch

* In package.json we import (logger) use-reducer-logger
* In HomeScreen replace useState with useReducers

1. React Bootstrap

* **ADD BOOTSTRAP**

Install react-bootstrap bootstrap

In app.js Converting Simple header main and footer to reactBootstrap

Navbar Container, LinkContainer,

* In package.json we install bootstrap react-bootstrap ans react-router-bootstrap
* In app.js we change the simple header into react-bootstrap header main and footer
* In index.css we remove the header design and give the class name of div used in app.js site-container
* In index.js we import bootstrap css from bootstrap components
* Create Product Component

Using react BootStarap desing the products in <Row> and <Col> tags …

1. We are creating a components folder and inside that we are creating product.js file where we are using the code of the products details…

This is Done usings props which helps in reusability of code

In the product.js file further we desing the code using react -Bootstarap using different tags like 🡪 Card, card.Body,Card.Title,Card.Text,Button

1. Then in the components folder we created rating.js file and done the rating and numreviews operatons in that . we import the fontowsum font in the index.html page inside the public folder ans done Some changes in the index.css folder .

Create Product and Rating Components..

* At first go to the HomeScreen and to change the product list from simple div to Bootstrap styles 🡪changes ar using tags(Row , Col,)
* Use of props ---- to use product items in multiple places we create a product component …Which is placed inside components folder and in the Prodcuct.js file . then import the details in the HomeScreen
* Now in the Product.js file we are changing the div into the Card tags card.Body , Card.Text, Card.Title all are from react-Bootstrap

Changes Done --Simple div to Card components of react-bootstrap which is also responsive

🡪Resolve the error of 🡪 each child in a list should get a unique ‘key’

Error resolve is done 🡪 go to the HomeScreen and after the map function inside the Col add the key Attribute { key ={product.slug}}

* In the HomeScreen.js HE

🡪Rating Components -----

In Product.js file add Rating components .

Inside components folder add The Rating.js file inside which keep the details of the rating and numReviews..

Then go to index.html in the public folder to import fontawesome

1. Product Screen

* Add Product Screen---

1 . Create Prodcut Details Screen – Open the ProductScreen.js inside the screens folder. Here we have to fetch the product from backend

2. Open server.js and then change app.get('/api/products/slug/:slug', (req, res) => {

  const product = data.products.find((x) => x.slug === req.params.slug);

  if (product) {

    res.send(product);

  } else {

    res.status(404).send({ message: 'Product Not Found' });

  }

});

In the ProductScreen we have use many react-Bootstrap tags like Row,Col,ListGroup,ListGroup.Item

Checking countInStock value we are also checking the avaivility of the product .

* To show the product name in the page title We ar using a package name

React Helmet async.. Inside the frontend folder install (npm I react-helmet-async) and then go to index.js and then wrap the whole App inside HelmetProvider . and then go to productScreen and before heading

 <Helmet>

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

              </Helmet>

Also do the change in the HomeScreen.js

* **Create-Loading-and-Message-Component**
* In the HomeScreen.js replace the loading div to a component name LoadingBox

Inside the component folder create a LoadingBox.js.

By this we are creating a universal loading and use it in every part of the project

In the LoadingBox page we used a Spinner icon from react-bootstrap

Same create a MessageBox component in the component folder

And inside messageBox use Alert from react-Bootstrap

* Create utils.js fle to resolve the error message of product not found

And then go to the productScreen and go to the try catch and as a payload return getError(err).

1. Cart Screen
2. Create React Context.

Implement Add to cart -- to implement add to cart features here we are using React Context 🡪 [link](https://reactjs.org/docs/context.html)

First create a store.js file inside src folder…

There we create a StoreProvider ..

We Use StoreProvider in index.js

Then in the ProductScreen .js we use the context and dispatch CART-ADD-Items.

And then in the App.js we added the cart option inside the container in a Nav tag

1. Complete Add To cart Functionality.

First in the data.js in backend we added \_id to every product.

Then in the server.js we created new api based on the product id

In App.js we replace the cartItems.length with the reduce function

Reduce((a,c)=> a+ c.quantity,0)

In the store.js we fix the issue in the CART\_ADD\_ITEM action

And in the productScreen we improve add to cart when user click add to cart multiple time.

1. Create Cart Screen.

In App.js we added new route for CartScreen

In index.css we added new class for image thumbnail.

We Created new screen name CartScreen.js to show list of items in the cart.

And inside the productScreen.js we use useNavigate() hook to navigate to redirect user to the cart screen after adding a new item to the cart.

1. Complete Cart Screen.

In Store.js we store cartItems in LocalStorage.we also used case ‘CART\_REMOVE\_ITEM’ action.

In Product.js we added stock availability

In CartScreen.js we implemented updateCartHandler to increase and decreasethe items number ..and revomeItemHandler for the trash icon to delete the item and also checkOutHandler for proceed to checkout button

6.Sign-in-Screen

1. Create Signin Screen..

Create a signinScreen.js page insinde screen folder

There we use useLocation hook and other different tags from react-bootstrap like(Container, Form, Form.Group,Form.Label,Form.Control)

Then in the App.js we add new route SigninScreen.

2. Connect To MongoDB.

Making connection to the mongodb Atlas .

Install Two packages dotenv and mongoose In backend.

Then in server.js connect mongoose.connect and connect to the database.

3. Generate Sample Products:

* Created a models folder inside backend.. Inside that created prodcutsModel.js.
* In prodcutModel.js we are creating mongoose model .. importing mongoose.Schema ..It accepsts an object as parameter and this objects defines the fields of product.

Mongoose.Schema generally accepts two objects one is the field for the Schema,and the second one is options where we are using timestamps : true.

Then we are creating a model based on the above schema named as Product = mongoose.model(‘Product’, productSchema).

* To seed or generate sample products in the database ..

For that inside backend folder create a routes folder ( it is made to make the api in the backend project modulers).

Inside that create

* seedRoutes.js – we are importing express and using s

const seedRouter = express.Router();

seedRouter.get('/', async (req, res) => {

await Product.remove({});

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

res.send({ createdProducts });

});

export default seedRouter;

and then in the Server.js file call app.use function and call seedRouter.

app.use('/api/seed', seedRouter);

* Then in the data.js remove the \_id bcz mongodb is going to assign id automatically.
* productRoutes.js – almost same as seedRoutes.js.

after creating this in the server.js we are using

app.use(‘/api/products/’,productRoutes)

const product = data.products.find((x) => x.slug === req.params.slug);

This above arrow function is creating error in productRoutes.js file while running . so it changed to an object . like below

const product = await Product.findOne({ slug: req.params.slug });

Generate Sample Users:

Almost same as sample product…

Created userModel.js inside models folder ..

Then in seedRoutes.js routes the User model same as product Model

And then in data.js created two sample user one as admin and other as user… There we install bcryptjs to secure password.

1. Implement Signin API.

First we are creating userRouters.js in route folder in backend

There inside post function we are using expressAsyncHandler which catch error in the async function. Inside backend folder we install express-async-handler ..

So if we have an error in this function we can handel it in server.js..

In server.js we are defining error handler for express.

app.use((err, req, res, next) => {

res.status(500).send({ message: err.message });

});

* It is like a middleware. When expressAsyncHandler is having some error then the above middleware will run and show the error message to the user.
* In package.json we install two new packages express-async-handler and jsonwebtoken.
* In Server.js we use express.json to convert fork data I post request to javaScript object in request.body.

Also we implemented errorHandler using express middleware.

* In utils.js we created generateToken function to generate a token.
* And in userRoutes we created new routes to singnin users

1. Complete Signin Screen. Important Section video 25.

🡪In package.json we install new package react-toastify to show error message in designed way.

🡪 In App.js we imported ToastContainer and ToastCss. Also used userInfo in the reactContext and if userInfo does exists then we show a drop down that shows username userprofile OrderHistory and signout.. Using react-bootstrap..

* In the Signin Out Handler we dispatch USER\_SIGNOUT
* In store.js we read the userInfo from localStorage and we added two new action in the reducers one is USER\_SIGNIN AND USER\_SIGNOUT.
* **Most imprt work is done in the signinScreen.js page**

Here we imported :

import Axios from 'axios';

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

import { Store } from '../Store';

import { toast } from 'react-toastify';

import { getError } from '../utils';

here we get the user Email and Password on submitHandler we pass it onto the signin api and if the data is correct we save the user in the localStorage and in the reactContext and redirect user to the redirect url.

1. Checkout Wizard
2. Create Shipping Screen.

In app.js we added new ShippingAddressScreen to get the address From the users.

In components folder we created CheckoutSteps.js to show the result of the steps we are in.

In index.css we style the checkout-steps.

In Store.js we get the shippingAddress from the localStrorage .. Also we have added a new action for SAVE\_SHIPPING\_ADDRESS in the reducer and if user enter new address we save it in the store

1. Create Sign Up Screen.

In the backend we created siginUp api .. created new userModel save it and then return the token along with the user information.

In frontend inside App.js we created new routes for signup screen.

And we created SignupScreen and get user information that create an account for that user.

1. Implement select Payment Method Screen.

In the App.js we add the routes for PaymentMethodScreen and in the signOutHandler we remove peymentmethod from localStorage.

In store.js we set a default value for payment method from localstorage.and we added an action SAVE\_PAYMENT\_METHOD inside that we update the paymentMethod inside the cart.

Also we created a PaymentMethodSCreen where we can select different payment method.

1. Create Place Order Screen.

In App.js we added PlaceOrderScreen.js

And we created PlaceOrderScreen we created preview of the order including shippingAddress and Items in the cart also here we show the order Summary by Calculating orderPrice taxPrice shippingPrice and order Total.

1. Implement Place Order Action. Important section.

In server.js we add new route for order.

In utils.js in backend we add isAuth middleware to authenticate request based on the Bearer token.

We Created orderModel to save an order in the database.

We created orderRoute we create an order in the backend.

In store.js we added cart clear action to make cart items empty after an order is placed.

**And in the PlaceOrderScreen we implemented Placing order..(This should be see in details after video 30.)**

1. Create Order Screen.

In backend site we created an Api to return the order information based on the orderId.

In App.js we added new Route name OrderScreen.js

In screens folder we Created OrderScreen and inside that we define a reducer to manage state and inside useEffect we define fetchOrder function inside that we send an ajax request to get the order details from the backend. And in the return part we show all information about the order

**OrderScreen can be go through for more detailing in video 31**

1. Pay Order By PayPal.

In server.js we added new get Api for the paypal to return the client\_id.

In orderRoutes.js we implemented a put Api to update the order status after payment.

In packaje.json we install react-paypal-js.

In index.js we wrap the <App /> in a Higher order Component <PaypalScriptProvider>

And in the orderScreen we Implemented Payments using PayPal Buttons.

1. Customer Area.
2. Display Order History.

Here in some Parts toFIxed(2) is used.

1. The toFixed() method converts a number to a string.
2. The toFixed() method rounds the string to a specified number of decimals.

In the orderRoutes we Created a new Routes To return list of orders of current users.

In App.js we added orderHistoryScreen.

And we Imlemented OrderHistory Screen

1. Create User profile Screen.

In App.js we are adding new routes ProfileScreen.js

And fix the issue og signOut option with

window.location.href = '/signin';

In Backend userRoute added new route to update user details.

Created new ProfileScreen in screen folder where we can update our profile details change our password .

1. Publish To Heroku.

In the server.js we added a middleware to serve all files inside build folder in frontend.

🡪app.use(express.static(path.join(\_\_dirname, '/frontend/build')));

And when user enter any address redirect user to the index.html. it’s the production version of our react app.

1. Create Sidebar And Search Box. IMP

We Created a sidebar in App.js and a SearchBox component and we used it in app.js

Also in the productRoutes we implemented an Api to return categories of product.

1. Create Search Screen. More Important

In the productRoute we implemented an Api to return products based on filtered and sort criteria and sort order.

In App.js we implemented the SearchScreen.js

In rating.js we changed the span of numreviews to caption.

And created searchScreen to filter products.

Need to go trough again : 🡪 SearchScreen , productRoutes.js

9. Admin Area.

1. Create Admin Menu
   1. define protected route component
   2. define admin route component

we created two components to protect the routes for login users like customers and admin users in the components folder ..

* 1. add menu for admin in header.

We use them in App.js to protect routes from accessing auouthorized user.

Also we created the DashboardScreen to test the result.

1. Create Dashboard Screen
   1. create dashboard ui
   2. implement backend api
   3. connect ui to backend

In the utils.js in the backend folder we added isAdmin middleware.

In orderRoutes.js we implemented an Api to return summary data of the ecom website.

In Package.json we install react google charts.

And in the Dashboard Screen we show the info box the charts of the ecom website.

1. Manage Products
   1. create products list ui
   2. implement backend api
   3. fetch data

Created a new Api in the productRoutes in backend . for /admin to get all products based on the selected page.

In the App.js we implemented the productListScreen.

And in the productListScreen we created a page to show the list of products from the backend for the admin.

1. Create Product
   1. create products button
   2. implement backend api
   3. handle on click

In the productRoute added a new route for creating a product.

In the productlistScreen created a new button and an handler to create the product.

1. Create Edit Product

In App.js we added a new route for productEditScreen

We created a new page for product edit screen.

And in the ProductListScreen.js we change the navigate from the creataHandler and make the new data added to the payload when Create is success.

We Created a Action column in the table with Edit option which navigate to the ProductEditScreen.js

* 1. create edit button
  2. create edit product ui
  3. dispaly product info in the input boxes.

1. Implement Update Product:

In the productRoutes.js we added a new Api to update product.

And in the product edit screen added submitHandler for the update button. And added new action for updating product in the reducer.

* 1. create edit product backend api
  2. handle update click.

1. Upload Product Image 🡪 IMP:

In package.json we install 3 packages { Cloudinary, multer, streamifier }.

In Server.js we added a new route upload from the routes folder (uploadRoutes.js)

We created a new routes uploadRoutes.js in the routes folder in backend to upload the image through cloudinary.

In the productEditScreen we implemented upload button and handel it using a backend api.

In the .env file in the backend we added some details about the cloudinary like cloudinary name, secrect key, id etc..

* 1. create cloudinary account
  2. use the api key in env file
  3. handle upload file
  4. implement backend api to upload.

1. Delete Product

In productRoutes.js we implemented an Api to delete a product.

And in the product List Screen we implemented actions to delete product and handel the state of the component after deletion.

* 1. show delete button
  2. implement backend api
  3. handle on click.

1. List Orders

In the orderRoutes we created an api to return all orders for ADMIN

In App.js we implemented new routes for the OrderListScreen.js

And we created a OrderListScreen.js to view the details of all the orders only for Admin .

* 1. create order list screen
  2. implement backen api
  3. fetch and display orders

1. Deliver Order

In orderRoutes we added an Api to change the state of the order to deliver.

In order screen we added a button to deliver order only for admin user

* 1. add deliver button
  2. handle click action
  3. implement backen api for deliver

1. Delete Order

In this we added a button on the orderListScreen in admin view to delete an order and a deleteHandeler to handel the deletion and send ajax request to backend

In the orderRoutes.js we created an Api to delete an record from backend.

* 1. add delete button
  2. handle click action
  3. implement backen api for delete

1. List Users

In userRoutes.js we added a new Api to return all users for admin

In App.js we implemented a new route UserListScreen for admin users

We created a userListScreen to see all users for Admin users.

* 1. create user list screen
  2. implement backen api
  3. fetch and display users

1. Edit User

In the userRoutes we added to new Api ..One for to get users by id from the database and Another is to update the user details in the database by Put request.

In the App.js we added userEditScreen.js

In the UserListScreen we added a button to edit the user details

And we created a userEditScreen to edit the details of the users.. which can be done only by the Admin users.

* 1. create edit button
  2. create edit product ui
  3. dispaly product info in the input boxes
  4. implement backend api
  5. handle edit click

1. Delete User:

In the userRouter.js we added a new route to delete a user.

And in the userListScreen we added a new button to delete a user and added a deletehandler.

* 1. add delete button
  2. handle click action
  3. implement backen api for delete