Topic : The Full Stack MERN E-Commerce Website

By : Abhyanshu Pandey

Welcome this is a full stack ecommerce website of MERN stack

FOR BETTER EXPLANATION YOU CAN REFER TO MY README MD ITS NOT COMPLETE BUT CAN HELP YOU – BUT I TRIED MY BEST TO GIVE THE COMMENTS AND THE EXPLANATION IN THE APP AND HERE ,

The Start

THE BACKEND

This will have the package file in the root directory

1 – The Server File

- This will be our main server connection to frontend file
- We will install some packages mentioned in the instructions for the website
- It will take the app , cloudinary , the data base connection and the errors that are used to help us find and stop any kind of unknown activity
- The env will be used when we go to the hosting
- On any errors occurs between connection of the site it will automatically stop the server

2 – The app file

- This will hold our routing and the use of the app
- Created the app file
- Then added the required packages and use app.use to use it
- Then created the routes main after the port (/api/v1)
- And added the file upload and the error handler for future use

3 – Utilities Folder

1- The Api Features folder

- This will hold our features sections for the product
- Created the class of apifeatures
- Created a constructor which will take the query and do the implementation based on it
- Created the search for search product based on the keyword
- Created filter based on the price and category , the keyword , page and the price limit is been removed to search products based on price too
- Gte for greater than , lte for lower than price

- Category will be same as the frontend it will be defined in the product schema
- The last will be the pagination it will show the products and the pagination based on the total products on the page
- The pagination with the filters doesn't worked for me it was giving the error so had to remove it from the all products controller

2 – Error Handler

- It will extends upon the error class of the express class
- Created a class named error handler to give the message and the status code based on the error is giving in the process
- It will be used in the next part of the controller to give back response if the condition not fulfilled usually "if" condition

3 – The Json Web Token file

- Creating a send token function to run when the token is passed to saved when the user wants it – usually on login or register and on the forgot password
- First we will take the user web token generated this will be done by the function in the user modal
- Created an different option based on the user entry of the token
- Created a response based on the conditions fulfilled
- Cookie expire will expire the cookie after the recommended date , user have to re login

4 – Send Email file

- This will use the node mailer to send the mail from our main owner id of the website to the user
- Careated the node mailer function with the service – what gmail or any other , then mail our , and the password of our id to send it to user
- Mail option will handle the the different constituents of the mail
- Then the mail will be send through the transporter

4 – Middleware folder

1 – Auth file

A - the token validation function

- This will take the require modules and the jwt package
- Created the authenticator function
- Taken the token from the cookie we saved in from the user register or the login
- Then decode data and verify its credentials if its true by the jwt.verify
- Then finding the user by the decoded token id
- This will also be used so the user as he login or register then he directly go to his profile page

B – The authorize roles

- This will handle the role being admin or not for having the admin routes access restricted
- Thes will take the role from the backend route which will access the role by the frontend redux library
- And if good to go then use the next() to go to the next step of the routing

2 – Catch Async Error file

- This will our try and catch block for the controller functions
- It will have the promise and if resolved then only it will continue the process

3 – Error File

- This will take the error handler to work with
- Created the status code and message function based on different condition of the errors
- It will have cast error , duplicate , Jwt token error , jwt token expire error

5 – Config folder

1 – The config env file

- This will have our environment variable this will use throughout our backend and the stripe for frontend]
- The variable will be

```
# The cloud server url
DB_URI
# temparary url for the frontend to reset the password in the controller of the forgot password
FRONTEND_URL
# this is for the jwt
JWT_SECRET
JWT_EXPIRE
COOKIE_EXPIRE
# This is for the mail functionality
SMPT_SERVICE
SMPT_MAIL
SMPT_PASSWORD
SMPT_HOST
SMPT_PORT
# This is for the image upload functionality
CLOUDINARY_NAME
CLOUDINARY_API_KEY
CLOUDINARY_API_SECRET
# This is for the payment functionality
STRIPE_API_KEY
STRIPE_SECRET_KEY
```

2 – The Database file

- This will be our file for the database connection to the mongo db
- It will use the mongoose to have the better experience and handle our app better
- Created the connect function using mongoose which will take the db url of the mongodb

6 – The Modals , Routes , And the Controller folders all together for better understanding

a) – The Product Functionality

1 - The Modal

```
const productSchema = new mongoose.Schema({
    name: {
        type: String,
        required: [
            true, "Please Enter Product Name"
        ],
        trim: true
    },
    description: {
        type: String,
        required: [true, "Please Enter Description"]
    },
    price: {
        type: Number,
        required: [
            true, "Please Enter Product Price"
        ],
        maxLength: [8, "Price Cannot Exceed 8 Figures"]
    },
    // stars given - this is the overall review rating average of all
users
    ratings: {
        type: Number,
        default: 0
    },
    // for multiple images
    images: [
        {

            public_id: {
                type: String,
                required: true
            },
```

```
            url: {
                type: String,
                required: true
            }
        }
    ],
    category: {
        type: String,
        required: [true, "Please Enter Product Category"]
    },
    // product in stock
    Stock: {
        type: Number,
        required: [true, "Please Enter Product Stock"],
        maxLength: [4, "Stock cannot exceed 4 Characters"],
        default: 1
    },
    // count no of reviews added
    numOfReviews: {
        type: Number,
        default: 0
    },
    // this is to add reviews to the product
    reviews: [
        {
            user: {
                type: mongoose.Schema.ObjectId,
                ref: "User",
                required: true
            },
            name: {
                type: String,
                required: true
            },
            // this is the rating withing the review of the product
            rating: {
                type: Number,
                required: true
            },
            comment: {
                type: String,
                required: true
            }
        }
    ],
```

```
    // this is to check who created the product
    user: {
        type: mongoose.Schema.ObjectId,
        ref: "User",
        required: true
    },

    createdAt: {
        type: Date,
        default: Date.now
    }
})
```

1. Created a mongoose schema and exported it with the name “Product”
2. In the reviews section the use ref is used to make a reference to the user and its id for the future user to get the reviews from the user id on the product

2 – The Product Routes

```
// To get all Products
router.route('/products').get(getAllProducts);

// To get product Details
router.route('/product/:id').get(getProductDetails);

// To get the Reviews Or Update the review
router.route("/review").put(isAuthenticatedUser, createProductReview);

// To Get All Reviews on The Product - no need to login because you wont have to
login to see the - anyone can see the review
router.route("/reviews").get(getProductReviews);

//to Delete A Review On A Product
router.route("/reviews").delete(isAuthenticatedUser, deleteProductReview );


// Admin Routes
// to get all products - for admin without  any filter
router.route('/admin/products').get(isAuthenticatedUser , authorizeRoles("admin")
, getAdminProducts);


// TO create Product - it will require the auth and the admin route
router.route('/admin/product/new').post(isAuthenticatedUser,
authorizeRoles("admin"), createProduct);
```

```
// To Update a Product
router.route("/admin/product/:id").put(isAuthenticatedUser,
authorizeRoles("admin"), updateProduct);


// To delete A Product
router.route("/admin/product/:id").delete(isAuthenticatedUser,
authorizeRoles("admin"), deleteProduct);
```

1. Taken the router from the express then created the routes the different routes handle the different functionality
2. The isAuthenticator and the authorized role is used to have the restriction on the route that require the validation of the user profile and the user role respectively

3 – the Product controller

Every function will be linked to its route as show in above route part

It will take the Product from the modal and the errorhandler , apifeatures ,cloudinary , catchasync error for the future use

1. The create product
    - It will take the images then push it into the array of strings to have it on the cloudinary to use
    - Then it will create product based on the images
    - Then used the product schema to use the create function on it to create product
    - Then sended the response back to the user
2. Get all products
    - This will get all the products from the data base
    - It will have our filters from the apifetaures file if the features will be applied it will find the product accordingly
    - It will have the result per page to show only the results restricted to page to have the pagination do its works
    - In this the pagination with the filter doesn't work for me giving the error so have to remove it
    - Then send back the user the response – the response defined here will be useful when we created the action and the reducer for the frontend to take the required data
3. The Get all Admin products
    - Finding all the products by the find method of mongodb
    - Sending back res of the products finded
4. Get product details
    - Finding product by id then sending he single product find in the response
    - This is for the single product
5. Update product – by admin

- Finding product by id
- Taking the images of before uploaded
- Then destroyed old ones if new ones are being sotred
- Then created the new product
- Sended the response back the product that been just created

6. Delete Product – by admin only
    - Finding by id
    - Delete the product and the images in it from the cloudinary too
    - Then sending res of the delete product message
7. Create product review
    - The review will be linked to the product schema
    - It will take the user id , name , the rating entered and the comment string
    - Finding the product if it exists
    - to check if the id of the user who input the review matched the id of the user who is logged in
    - if review is added then checking if the review already exits if yes then update on it else create a new one
    - the review of the user will be limited to the product he put it into
    - and the review deleted can be only on the product not all the reviews entered by the user
    - created an average of the reviews to show the average rating of the products
    - sended the res of success
8. Getting all Product reviews
    - It will have all the reviews on the product
    - It will be shown to the user or admin by the product id
    - It will send res back the product reviews
9. Delete Product review
    - First find the product
    - Then filter all the reviews that not matched the entered review
    - Then if the review matched then change the average
    - Then see if the review is zero then no change in ratings else the average will be the avg / reviews on the product
    - Then get the no. of the reviews
    - Then find the reviews again
    - Then give back the response of the success.

b) – The Order Functionality

1. Order Modal

```
    const orderSchema = new mongoose.Schema({

  // This is for the Shipping Address
```

```
    shippingInfo: {
        address: {
            type: String,
            required: true,
        },
        city: {
            type: String,
            required: true,
        },
        state: {
            type: String,
            required: true,
        },
        // for international - if want for just one then use the by
default  Country name
        country: {
            type: String,
            required: true,
        },
        pinCode: {
            type: Number,
            required: true,
        },
        phoneNo: {
            type: Number,
            required: true,
        },
    },

    // This Is for the Order Item
    orderItems:[
        {
            name:{
                type: String,
                required: true,
            },
            quantity:{
                type: Number,
                required: true,
            },
            price:{
                type: Number,
                required: true,
            },
            image:{
```

```
            type: String,
            required: true,
        },
        product:{
            type:mongoose.Schema.ObjectId,
            // Product Here is referencing to the Product Schema
            ref:"Product",
            required:true,
        },
    },
],

user:{
    type:mongoose.Schema.ObjectId,
    // User Here is referencing to  the  User schema
    ref:"User",
    required:true,
},

paymentInfo:{
    id:{
        type: String,
        required:true,
    },
    status:{
        type: String,
        required:true,
    },
},

paidAt:{
    type:Date,
    required:true,
},

// this Will be calculated in the front end
itemPrice:{
    type:Number,
    required:true,
    default:0,
},

taxPrice:{
    type:Number,
    required:true,
```

```
        default:0,
    },

    shippingPrice:{
        type:Number,
        required:true,
        default:0,
    },

    totalPrice:{
        type:Number,
        required:true,
        default:0,
    },

    orderStatus:{
        type:String,
        required:true,
        default:"Processing",
    },

    deliveredAt:Date,
    createdAt:{
        type:Date,
        default:Date.now,
    },
});
```

1. crated the mongoose schema
2. then exported it with the Name “Order” to use elsewhere
3. the use ref is used in the product to attach to the product for the reference of its id
4. the ref is used in the user to reference to the user who created the order

2 – Order Rotues

```
// To Create An Order
router.route("/order/new").post(isAuthenticatedUser , newOrder);

// To get the All Orders done by a User
router.route("/orders/me").get(isAuthenticatedUser , myOrder);

// To Get The single Order - User Should be able to see his single order
router.route("/order/:id").get(isAuthenticatedUser , getSingleOrder);
```

```
// to Get All Order -- Admin
router.route("/admin/orders").get(isAuthenticatedUser , authorizeRoles("admin") ,
getAllOrders);

// to Update An order Status - Admin
router.route("/admin/order/:id").put(isAuthenticatedUser ,
authorizeRoles("admin") , updateOrder);

// To Delete an order -- Admin
router.route("/admin/order/:id").delete(isAuthenticatedUser ,
authorizeRoles("admin") , deleteOrder);
```

10. it will usually take a isauthenticator for the verification to crated order if the user exists so the non logged in user cannot make the order and also the purchase and the payment this is because we cannot locate an unknow user
11. it will require the authorize roles to validate the role of the user admin to have him work on the process after the order is been made.

- The Order Controller

  It will take the order from the modal schema

  The product for using it

  And the error handler and the catch async error (our self try and catch block)

  Every function will be linked to its route as show in above route part

  It will take the Product from the modal and the errorhandler , apifeatures ,cloudinary , catchasync error for the future use

  1. New Order
     - It will take some data from the user to create an order

```
const {
        shippingInfo,
        orderItems,
        paymentInfo,
        itemPrice,
        taxPrice,
        shippingPrice,
        totalPrice
    } = req.body;
```

     - Then it will create the new order based on the given data

- It will create the order by the order.create method it will also take the login user id and the current data to create the order
- Then it will send the created order in response to show the user the order that been created

2. Get single order
    - This will take the order by the user id and populate the order object with it email and the user
    - Then it will just simply return the order that been find by the user id on the order id and send the order in response
3. My order
    - It will find the order by the user id in the order schema
    - It will find all the orders created and send back the response to the user the all order is been created by the user
4. Get all orders – admin
    - This will find all the orders on the site
    - Then also calculate the total amount been earned by adding all the prices of the orders been created into total orders
    - Then it will send the orders in response
5. Update order
    - This will update the order status based on the process it in
    - First find the order by id of the order
    - Then if the order is not been shipped then shipped and update the stock since when the product is shipped then we have to update our stock
    - Update stock will take the product from the order helped by the reference to refer to the product by the id and then quantity of the products that's been ordered
    - Then take the order status
    - Then save the new order status
6. Update stock
    - This is the function used in the update product to update the stock when the order is been shipped
    - Finding product by id
    - Then minus quantity of the product that been shipped
    - Then save the product
7. Delete order
    - Finding order by the id
    - Then delete the order only one
    - Sending response the success if done

C) – The User Functionality

1. The User Modal

```
const userSchema = new mongoose.Schema({

    name:{
        type:String,
        required:[true,"Please Enter Your Name"],
        maxLength:[30,"Name Cannot Exceed 30 Characters"],
        minLength:[4,"Name Should have More Than 4 Character"]
    },
    email:{
        type:String,
        required:[true,"Please Enter Your Email"],
        unique:true,
        validate:[validator.isEmail,"Please Enter A valid Email"],
    },
    password:{
        type:String,
        required:[true,"Please Enter Your Password"],
        minLength:[8,"Password Should be greater Than 8 Characters"],
        select:false,
    },
    // not an array because it will be only one image - will be on
cloudanary
    avatar:{

        public_id: {
            type: String,
            required: true
        },
        url: {
            type: String,
            required: true
        }
    },
    role:{
        type:String,
        default: "user" // till we make him a admin
    },
    createdAt:{
        type:Date,
        default:Date.now,
    },
    resetPasswordToken: String,
    resetPasswordExpire: String,

});
```

2. Created the mongoose schema
3. Then export it with the name of “User” to use in the different files
4. This user modal file will have some inbuild functions to not write those functions in the controller files

   The all will use the userSchema to work with as the object to store or work with the data

   a) The pre save function
      - This will be the function which will save the function in the data base
      - This function will but the password rehashing can be a problem do it will have a condition if the password is being reset or update only then rehash the password otherwise don’t
      - The function also save the password the hashed password

   b) The get jwt token function
      - This will return the token
      - Then token will be mix of the user id and the jwt secrect we have from our environment variable
      - The jwt secret is very sensitive so we don’t share it with anybody other wise anyone can create the fake users in our data base
      - This will also take the jwt expire to expire the token after a given amount of time

   c) The method of compare password
      - This will compare our password to login only if the password matched with the created one
      - The only bcrypt can compare the passwords it hashed so we use bcrypt compare method to compare

   d) The method of get reset password token
      - This will create an hex to get the reset password token that been send to the user for his password reset
      - It will do the hash and take the new password and the expiry of how long before the reset generated token will be expired
      - Then it will return the reset token to the function of reset password

2. The User Routes

```
// to register a user - to sign up
router.route("/register").post(registerUser);


// to login a user - for sign in
router.route("/login").post(loginUser);
```

```
// route for Forgot password
router.route("/password/forgot").post(forgotPassword);

// reset password - token will be the parameter we provide
router.route("/password/reset/:token").put(resetPassword);


// to logout the user
router.route("/logout").get(logout);


// To get the User Details
router.route('/me').get(isAuthenticatedUser, getUserDetails);

// to update password
router.route('/password/update').put(isAuthenticatedUser,
updatePassword);

// update profile - it was me/update , but i can change it later
router.route('/me/update').put(isAuthenticatedUser, updateProfile);

// delete account by the user itself
router.route('/me/delete/:id').delete( isAuthenticatedUser ,
deleteProfile);

// Admin Routes
// to get all users (admin)
router.route('/admin/users').get(isAuthenticatedUser,
authorizeRoles('admin'), getAllUsers);

// to get a specific user (admin)
router.route('/admin/user/:id').get(isAuthenticatedUser,
authorizeRoles('admin'), getSingleUser);

// To update user role (admin)
router.route('/admin/user/:id').put(isAuthenticatedUser,
authorizeRoles('admin'), updateUserRole);

// To Delete a User From the Websites Database (admin)
router.route('/admin/user/:id').delete(isAuthenticatedUser,
authorizeRoles('admin'), deleteUserProfile);
```

1. Taken router from the express
2. Tekan the isAuthenticated for the user authentication to be used only when logged in

3. The authorizes roles to have the restricted routes for the admin

3. The User Controllers

   This will take the crypto for reset password to decrypt it

   User from the Schema

   The Error Handler , the Catch Async Errors

   Then the token from the jwt to authenticate the user

   And send email to send the mail in the required places (only using in the forgot password for currently

   Every function will be linked to its route as show in above route part

   It will take the Product from the modal and the errorhandler , apifeatures ,cloudinary , catchasync error for the future use

   1. Register User
      - This will take the user email, password , name that Is required
      - It will take the avatar also that is also necessary
      - Then it will send back the token in the response
      - In here the sendToken function will be use to authenticate the user and If good to go then provide him a token for future use
   2. Login user
      - This will take the email and password from the user
      - Finding the user from the user data base by the email and password
      - If user exists then compare the passwords
      - If all clear then the send token function to validate and give the user the login token
   3. Logout User
      - This will take the cookie and it will null the token matches the user id
      - The expiry date will be current
      - Then send the response of the logout
   4. Forgot Password
      - Finding the user by the email
      - Then getting the reset token to be send in the response from the function resettokenpassword

- Saving the user
- It will take the host and the id for non hosting for right now
- The message will be send with the reset password url
- Then send email function will be used to send the mails to the user from the site manager email it will be auto generated
- Then we will reset the password token and expire to undefined when the task is done in the user modal to have the user to do the next task other wise the old will not take the new token and expire
- Then saving the user

5. Reset Password
    - Taking the password token from the url and creating a hash the sha256 hash I used here , then update the hash and then digest it with hex
    - Then the crypto is used to decrypt it
    - Then finding the user with the reset password token and setting the expire of the token
    - Then checking the password and confirm password matched
    - If yes then saving the password entered in the user.password
    - The password saved will be hashed because of the modified condition in the user modal
    - Then reset password token and expire will be undefined to add the next set of the reset for the user
    - Then saving the user
    - And sending the token for authentication and giving user the password
6. Get user details
    - Taking the details of the user who is logged in
    - Then sending his data in the response
7. Update Password
    - Finding user and selecting the password of the user also
    - Comparing the old password with entered one
    - Checking if the new password matched the confirm password then
    - Giving the password in users password and saving the user
    - Then sending the token
8. Update profile
    - Taking the user name and email
    - Then taking the image of the user too
    - Then finding and updating the user with his id and the new user data object (the image object)
    - Sending the response of the success
9. Delete profile -
10. Get all users – admin
    - This is to take all users on the website
    - Just user find to find all the users
    - Sending the response of success and the users object

11. Get single user - admin
    - Finding user by the id in the parameter
    - Then sending the response of success and the find user object
12. Update user role - admin
    - This is to update the user role from admin to user or vice versa
    - This will take the name , email , and role of the user from the data of the user body
    - Then it will find and update the user find form the parameter id and then
    - Change the data of the user to new user data
    - Res send the success
13. Delete User - admin
    - This is to delete the user from the admin side
    - This will take the user by id
    - Then it will destroy the images of the user from cloudinary
    - Then delete the user by the delete one function
    - Then sending the message in response of user deleted successfully

d) – The Payment functionality

1. Payment Route
    - Created two routes first for checking the process if on the payment
    - Taking the stripe key back to the user in frontend to to set the payment
2. The Payment controller
    - We will use the api key in the frontend but we are calling it here so we can have a secure api key
    - Taken the stripe and the secret key
    - Then created the payment function
    - Then taken the body by the stripe created function
    - The amount and sended Back the response of the client secret and success meassge
    - It also has the stripe api key function theis will just give back in response the stripe api key

BACKEND IS DONE HERE ……..

THE FRONTEND

Initial steps

- First created the react app with the named frontend
- Used the npx create-react- app appname
- Then deleted some unnecessary files
- Installed the every required packages (refer to the instruction readme)
- And then started the code

MAIN FILES

1. App file
    - This will hold our main all the pages and its routing
    - It will take the users data also to work with
        1. The stripe api function
            - This will take the api key from the backend and set the api state to data.stripe api key
            - This will then used in the use effect function to call
            - The use effect function to load user get stripe api key and get the fonts used in this app
        2. The routing
            - This will have the routing and also the protected routes
            - It will have the routing with same basic router , routes and route
            - The non restricted routes will be open
            - The restricted routes will have the authenticated user to have
            - The admin restricted routes will also check the role of the user
            - Basic route will look like this
            - ```
              <Route element={<Dashboard/>} path='/admin/dashboard' exact/>
              ```
            - The stripe will be check the stripe key in the protected route to make the payment.
        3. The Protected route component
            - Created the component then taken the user authentication from the redux state
            - if user is authenticated then to profile otherwise login
            - if user is admin then dashboard other wise profile so people cannot access the admin dashboard to modify it only the admins can
2. Index file
        1. It will use the store so it will be wrapped in the provider to work with it

```
<Provider store={store} >
    <ToastContainer toast={toast} />
    <App />
  </Provider
```
```

3. Store file
    1. This will take all the reducers and combine it to store in the redux state
    2. It will take the product reducers
        - newProductReducer,
        - newReviewReducer,
        - productDetailsReducer,
        - productReducer,
        - productReviewsReducer,
        - productsReducer,
        - reviewReducer
    3. the user reducers
        - allUsersReducer,
        - forgotPasswordReducer,
        - profileReducer,
        - userDetailsReducer,
        - userReducer
    4. The cart reducer
        - cartReducer
    5. the order reducers
        - allOrdersReducer,
        - myOrdersReducer,
        - newOrderReducer,
        - orderDetailsReducer,
        - orderReducer
    6. Then it will combine all through the combine reducer
    7. Then it will have the intial state for the cart functionality
    8. Then for middle ware we use the thunk
    9. And then it will create the store with all the combined reducer , initial state , and apply the middleware
    10. Then we export the store to use in different file mainly index.js for now
4. The basic structure of the redux functionality
    - The constants this will take all the constants for the given component required to perform the action , it is used to not have error in getting the reducer with the string names
    - Action it will take the data from the user or the api call when the dispatch is used then it will request the data from the backend with the api call and then it will pass that data into the reducer to change the state
    - Reducer will take the state and change it accordingly to the functionality and the data required and passed from the action
    - At then end all will be stored in the store file for app to use

THE COMPONENTS SECTION

1 – The layout Files

- The About us
  - This will have our layout of the about us page simple page with the image and some links
- Banner
  - This will have a banner with the image and a button to link to the path we define (to product page in our case)
- Contact us page
  - This will have a link that will take you to the mail of yours to write a mail to us
- The feature
  - This section will show our features provided by our app
- Loader
  - This will be our page to show the loading screen when the app is loading
  - It will use the loading state from the redux work properly
- Meta Data
  - This will have our heading render dynamically with the help of helmet package
- Header
  - Created the navbar using the bootstrap
  - Added all the functionality with the links
  - The profile and login linking will be done by the authentication if the user is login then profile and other options for the user , if the user is not then show him the login or register
  - The dashboard only when the user is the admin
  - This will also have the search functionality
    a. This will have the submit handler
    b. The function takes in the keyword state the key word will be taken from the parameters query section to search
    c. Then if the keyword exits then it will go to the products and the products present with that names
    d. If the product does not preset then it will go to the products page
  - the logout functionality
    - the dispatch of logout will be send
    - if the user is logout then we will navigate to the login page
- the footer page
  - it will be in the bottom of our page

2 – The Home Files section

1. Home file
   a. This will be our main landing page
   b. It will have the banner and the features products section
   c. The feature products will be taken by the product card
   d. Mapping the product from the state and then the product will be show based on the id
2. Product card

- This will be our product card which will show the product
- This will have the following to show
    a. The product link to single product
    b. The image of the product
    c. The name of product
    d. The options for the rating of the product – the starts is used from the material ui to have it
    e. The product review section
    f. Then the product price

3 – The Product section

This will have the use of all the reducer and actions and the constants for the utilization so will write in the format

1. The Products page
    - This will show all the products in our website with the filters section
    - Imported all the required modules
    - Used the material ui carousel for use
    - Created the categories to choose from
    - Then the main function
    - Use states in function
        a. The current page for pagination
        b. The price for the price filter
        c. The ratings for the rating
        d. Category for the category selection
        e. Taking in keyword to go to the page when the search happen
    - The use effect this will show any error occurred
    - The use effect will dispatch a function of get product in the action which will take the data from the api url
        a. The url will be taken by the filter applied so there will be two different url used one with category applied one with not
        b. The action will take the data and pass it to reduce
        c. The products reducer will take the product array
        d. Then it will set the
        products: the no of proudcts
        productsCount: the count of the products on website
        resultPerPage: the result shown per page
        filteredProductsCount: filter the products based on the given situation
    - Products page will show the products it take
    - Then the filters will be shown the price , rating and the category
    - The page will also have a pagination at the bottom
2. Product details page
    - Created the page

- Taking the user id and the product details state and the new review for creating the review
- Review form handler for giving the review
- The two functions for the quantity and one for the add to cart the quantity to the add to cart
- The new review reset is used to stop the reset after one successful attempt
- The main function
    1. It will have the one div with the image and the slider / carousel
    2. Then the next div will be the products details
    3. There will be the add to cart button and the submit review button
    4. The add to cart will check the number of product that can be added based on the stock info from the product in the redux
    5. For submitting review used the material ui dialog box
    6. Then at the bottom there will be a review section which will show all the review on the product

3. Review Card
    - This will be in the product details page it will show the review of the user
    - It will have the profile image and the ratings and the comment done by the user

4 – The User folder section

The actions and the reducer will be change and the api call will be to but the main dynamic of the function will be same the request , success or fail of the request only the data will be send in the function is writing here

1. Profile page
    - This will have the profile image and the update profile in first div
    - The update profile will have the ability to change the every thing in the profile except the password
    - The profile data will be taken from the user state after the login
    - Then there will be second div which will have the user details and the update password and my orders section
2. The Login signup page
    - This will be both login and signup page in the same component it will be handled by a switch tab function
    - For login It will take the
        a. Email and the password
        b. On event it will call a function then it will dispatch the login function to the dispatch will take the login email and password
        c. That will go to the action login and post the login request and send the data to the reducer to change the state
    - For register
        a. It will take the name , email , password , avatar of the user to register
        b. Then sends the from to the action of register user

- If the conditions are matched then we will redirect to the profile page otherwise we will go in to the error

3. Update Profile page
    - This will take the data same as register and give a post request to the backend only difference here it will take the only already existing user
    - And you cannot enter the password here the update password will be on the different page
4. The update password
    - This will take the user form as register the difference here will be the that it will only take the password form the user
    - The password will be passed in the action and then if success then pass the data and change the state in the reducer
    - This will take the current password , new password and the confirm password
    - Then we will send the form in the action and then usually the change in the user state
5. The forgot and the reset password
    - The forgot password will take in the email id of the user
    - Only the non logged in user can make this call
    - Pass it to the action as form and then the
    - Backend will send the auto generated message with the reset token
    - Then the reset password token will be taken and used to go to the reset page of password of the user
    - The reset password will take the new password and the confirm password

5 – The Cart Folder section

1. Cart file
    - This will show us the data of add to cart
    - This will show the image , quantity (can be updated here) , total price a remove button to remove the item from the cart and at the bottom a proceed to payment
    - The cart will only got to the shipping details page if the user is logged in other wise it will go to the login page to make the next step work
    - The cart will be store the items in the instant storage to use
    - The add to cart action will take the id and quantity of the product to work with
2. The cart item
    - This will be our card to show the cart items in the product
3. The Shipping page
    - Shipping page will take all the states of our data and save it into the state for the later use as of the shipping address
    - It will take the address , city , pincode , phone no,
    - The package of country- state-city is been used here to have the state select based on the country
    - Then at the end it will have the submit that will submit our form for the next step which is confirm order

4. The checkout step
    - It's a line above that shows the step of our order process
    - The stepper is been used by the material ui
    - It will have an active step
5. The Confirm order page
    - This will show us the details of our order and the items we have before the payment process
    - If every thing is fine accordingly to us then we will go to the payment page
    - This will navigate to the stripe payment method page the process/payment
    - The order info will be saved in the session storage
6. The payment page
    - The payment page will handle our payment with the stripe it will take the order info and the payment method
    - Then it will create a payment
    - On the success the navigate to the order success page
7. The order success
    - This is just a page to show the process is successful and we can see more products

6 – The Order Folder Section

The data grid is been used here it is easier to use than to create the whole table by scratch and it also include the filters this is from the material ui

1. My order file
    - This page will show all the orders done by us it will take the all the order by our id
    - It will dispatch the get order details action which will take the id
    - The reducer will change the state of page with the details
    - We will take the order details from the orderDetails state
2. Order details page
    - This will show us the details of the single order
    - It will take the order detail by the order id

7 – The Admin Folder components

This will be our restricted pages only accessible to the admin – its backend routes will take the role and the login to continue to it

1. The dashboard
    - This will be our main landing page on the dashboard from the user profile or the navigation
    - It will have 2 div one with the side bar and second one is the main div with the info of the page
    - The <Sidebar/> Component will be constant on all the pages
    - The main dashboard page will have the details of our total product , amount earned , orders made and the users on our website

- The dashboard will have two charts taken by the js chart and react js chart
- The line chart will show the amount earned and the donut will show the products that are in stock or out of the stock

2. The sidebar
    - It will be constant throughout every page
    - It will have the link to dashboard , the product which will include the create and all products , the order , the users and the reviews page
3. The product and update and all product
    - The all product will show all the products that are in the app with the product id it will find with the action of all the products without any argument
    - It will have functionality to update and delete the product the update and delete will be done by the order id
    - The create product will take , the new product file
        a. Name
        b. Price
        c. Description
        d. Category
        e. Stock
        f. And the images of the product to create
        - After creating the product it will go to the dashboard page
        - The action function will be the createProduct to create it which will take the form of the data and also append the images urls in it.

    a. The update product will take same as the new product but it will take the exiting product and you can change the product fields there
4. The Order list and the Process payment
    - The order list will show us all the order done on the app
    - The order list will be taken by the order id and the order is to get all the orders action with no arguments
    - The order list will have two functionality the update the process and the delete one
    - The delete order will delete the order
    - The update will update the order process of the order
    - If the order is processing we can update it to shipping and if shipped then to delivered
    - the order will be taken by the id to take the single product to update
5. User list and update the user
    - This will take the all the user on the website we can update their role and also delete from the website
    - Update user will take the user id and the form to update his role and nothing else
6. Product review
    - In here we can find all the reviews on the product by its id
    - The all reviews will be related to the specific order id
    - We can delete the review here from the user id that has entered and the product id

FRONTEND HERES

I Know this explanation will not be that great but I tried my best to explain what is what ,

You can refer to the readme if you want it has in depth detail but it is incomplete for now

Thanknyou

THINGS LEARNED DURING THIS

- Learned how to create the routes with protected and normal
- Learned how to use the outlet
- How to create the backend routes and what kind of data to be send according to your requirement
- Learned how to create the full stack big project
- Learned how to use the different packages of the different things
- How the reducers works and how to call the action functions
- How to give the popup messages by the toast
- How to get the user data from the frontend and how to use the destructuring of the object
- How to call the api and change the state according to need
- How to use the reducer in our app
- AND MANY MORE SMALL THINGS

SOME THINGS CHANGED

- The react toastify is used instead of the alert
- The material ui is used for the things used instead of the bootstrap in many occasions
- The navigation is used instead of history
- The use params is used instead of the match
- The page filter is used without the pagination change
- Many package name is changed and the different things added

RESOURCES USED

- The app is been successful by the help of the abhishek sings youtube cannel ecommerce webstie you can refer to him for more
- The different you tube videos
- The reference to the internshala instaclone for the things

THANK YOU NOTE

- I have been thankful to the teachers that taught me how to do the routing well and how to interact with the different parts of the languages in the stack
- Great full of the community that helped me in cases that I needed
- The abhishek singh you tube cannel that I tooked the inspiration from to create this app.

MISTAKES AND THE INSUFICCIENCY IN THE PROJECT

- The payment gate way is used but not the paypal - didn't know how to send the address with it
- The hosting didn't happed – it was getting late to submit the project before the viva I tried my best but I also had to go further to make the progress hope you'll understand
- The context api doesn't used don't know how to create it
- The project is not that self made that I would like or you will recommend but I tried my best to learn from it more than I tried to created it all by my self