**Objective:**

This project aims to implement a web application to maintain visitor and information about COVID 19 testing facility, vaccinated percentage and to maintain covid tracks in India. The motivation behind the development of this project is a need to efficiently manage the footfall at the testing facility. The product aims to digitalize the record maintenance of COVID 19 in India.

**Users of the System:**

1.   Admin

2.   Public

**Functional Requirements:**

* Each side navigation icons have its own specific contents.
* Vaccinated and non-vaccinated are also mentioned with the percentages.
* Vaccine 1st and 2nd dose are vaccinated by 54.9% of the people in India currently.
* Affected people of COVID 19 are differentiated by Confirmed, Active, Deceased and Recovered.
* Total number of people vaccinated doses are also administrated.
* The tabular column referred here is mentioned the number of people affected by COVID 19 are differentiated by state-wise and differentiated by Confirmed, Active, Deceased and Recovered.
* India map mentioned here indicates the state with coloured effects mentions the COVID affected and delivers content of states.
* Here comes a online bot with a help-line number which provides service 24/7 and also provides mental health counselling services for a stipulated time.
* It provides a detailed list about the vacancies of beds and availability of ventilators in the hospitals.
* Pop-up messages to spread awareness of COVID among the people.

**Output/ Post Condition:**

* Records Persisted in Success & Failure Collections
* Standalone application / Deployed in an app Container

**Non-Functional Requirements:**

**Security**

* App Platform –User Name/Password-Based Credentials
* Sensitive data has to be categorized and stored in a secure manner
* Secure connection for transmission of any data

**Performance**

* Peak Load Performance (during Festival days, National holidays etc)
* eCommerce -< 3 Sec
* Admin application < 2 Sec
* Non-Peak Load Performance
* eCommerce < 2 Sec
* Admin Application < 2 Sec

**Availability**

       99.99 % Availability

**Standard Features**

* Scalability
* Maintainability
* Usability
* Availability
* Failover

**Logging & Auditing**

* The system should support logging(app/web/DB) & auditing at all levels

**Monitoring**

* Should be able to monitor via as-is enterprise monitoring tools

**Cloud**

* The Solution should be made Cloud-ready and should have a minimum impact when moving away to Cloud infrastructure

**Browser Compatible**

All latest browsers

**Technology Stack**

**Front End** Html, CSS and Java Script.

**Material Design**  Bootstrap / Bulma

**Server Side** Spring Boot / .Net WebAPI/ Node Js

**Database** MySQL or Oracle or MSSQL

**Key points to remember:**

1.   The id (for frontend) and attributes(backend) mentioned in the SRS should not be modified at any cost. Failing to do may fail test cases.

2.   Remember to check the screenshots provided with the SRS. Strictly adhere to id mapping and attribute mapping. Failing to do may fail test cases.

3.   Strictly adhere to the proper project scaffolding (Folder structure), coding conventions, method definitions and return types.

Adhere strictly to the endpoints given below.

**Application assumptions:**

1.   The login page should be the first page rendered when the application loads.

2.   Manual routing should be restricted by using AuthGuard by implementing the canActivate interface. For example, if the user enters as [http://localhost:8000/signup](http://localhost:4200/signup) or [http://localhost:8000/home](http://localhost:4200/home) the page should not navigate to the corresponding page instead it should redirect to the login page.

3.   Unless logged into the system, the user cannot navigate to any other pages.

4.   Logging out must again redirect to the login page.

5.   To navigate to the admin side, you can store a user type as admin in the database with a username and password as admin.

6.   Use admin/admin as the username and password to navigate to the admin dashboard.

**Validations:**

1.   Basic email validation should be performed.

2.   Basic mobile validation should be performed.

Password validations should be performed

**Project Tasks:**

**API Endpoints:**

**USER**

**Action URL Method Response**

Login /login POST true/false

Signup /signup POST true/false

Get All Products – Home /home GET Array of Products

Add to cart /home/{id} POST Item added to cart

Cart Items /cart/{id} GET Array of Cart Items

Delete cart Item /cart/delete POST Cart Deleted

Cart to Orders /saveOrder POST Cart items added to the Orders list

Orders list /orders POST Array of Orders

Place order directly /placeOrder POST Place items to orders directly

**ADMIN**

**Action URL Method Response**

Get All Products /admin GET Array of Products

Add Product /admin/addProduct POST Product added

Delete Product /admin/delete/{id} GET Product deleted

Product Edit /admin/productEdit/{id} GET Get All details of Particular id

Product Edit /admin/productEdit/{id} POST Save the Changes

Get All Orders /admin/orders GET Array of Orders

**Frontend:**

**Customer:**

1. **Auth**: Design an auth component (Name the component as ***auth***for angular app whereas ***Auth***for react app. Once the component is created in react app, name the jsx file as same as component name i.e Auth.jsx file) where the customer can authenticate login and signup credentials

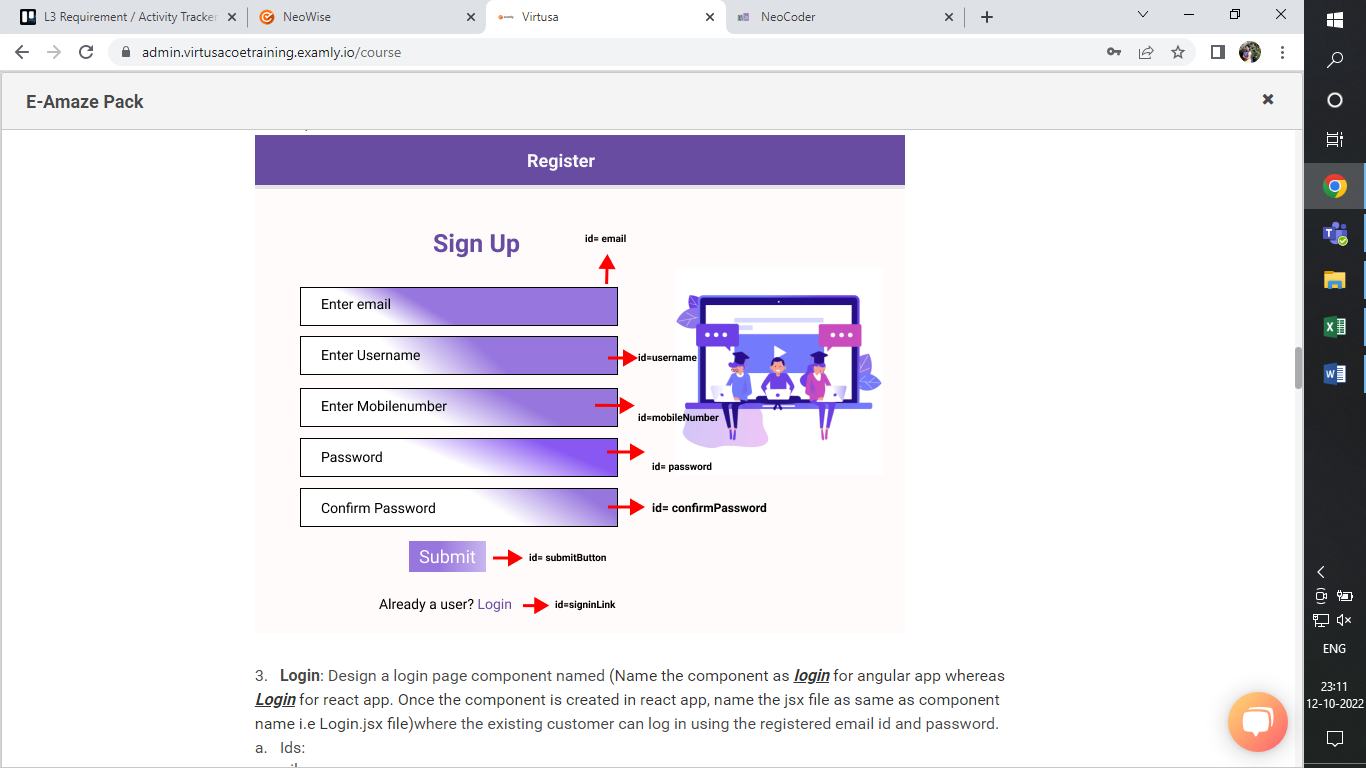
2.   **Signup**: Design a signup page component (Name the component as ***signup***for angular app whereas ***Signup***for react app. Once the component is created in react app, name the jsx file as same as component name i.e Signup.jsx file)where the new customer has options to sign up by providing their basic details.

a.   Ids:

* email
* username
* mobilenumber
* password
* confirmpassword
* submitButton
* signinLink
* signupBox

b.   API endpoint Url: [http://localhost:8000/signup](http://localhost:4200/signup)

c.   Output screenshot:



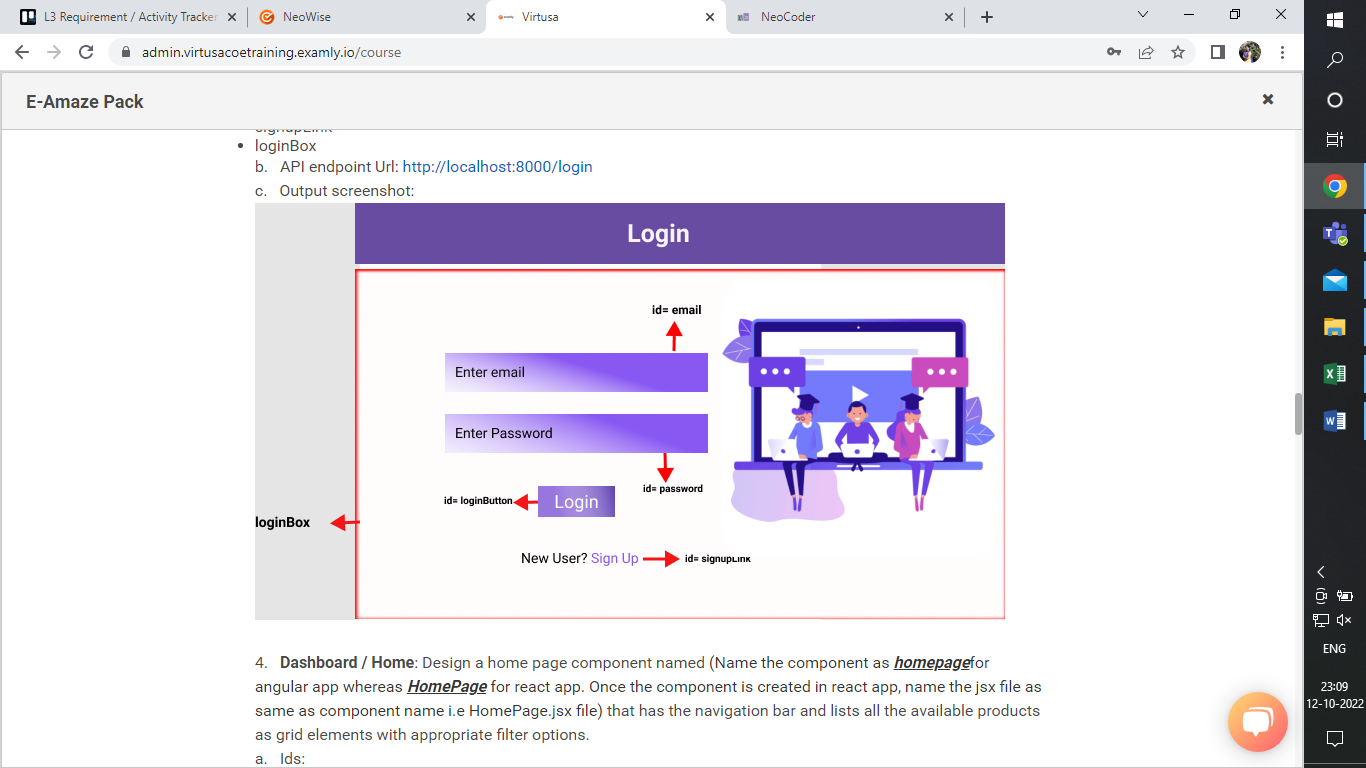
3.   **Login**: Design a login page component named (Name the component as ***login*** for angular app whereas ***Login*** for react app. Once the component is created in react app, name the jsx file as same as component name i.e Login.jsx file)where the existing customer can log in using the registered email id and password.

a.   Ids:

* email
* password
* submitButton
* signupLink
* loginBox

b.   API endpoint Url: [http://localhost:8000/login](http://localhost:4200/login)

c.   Output screenshot:



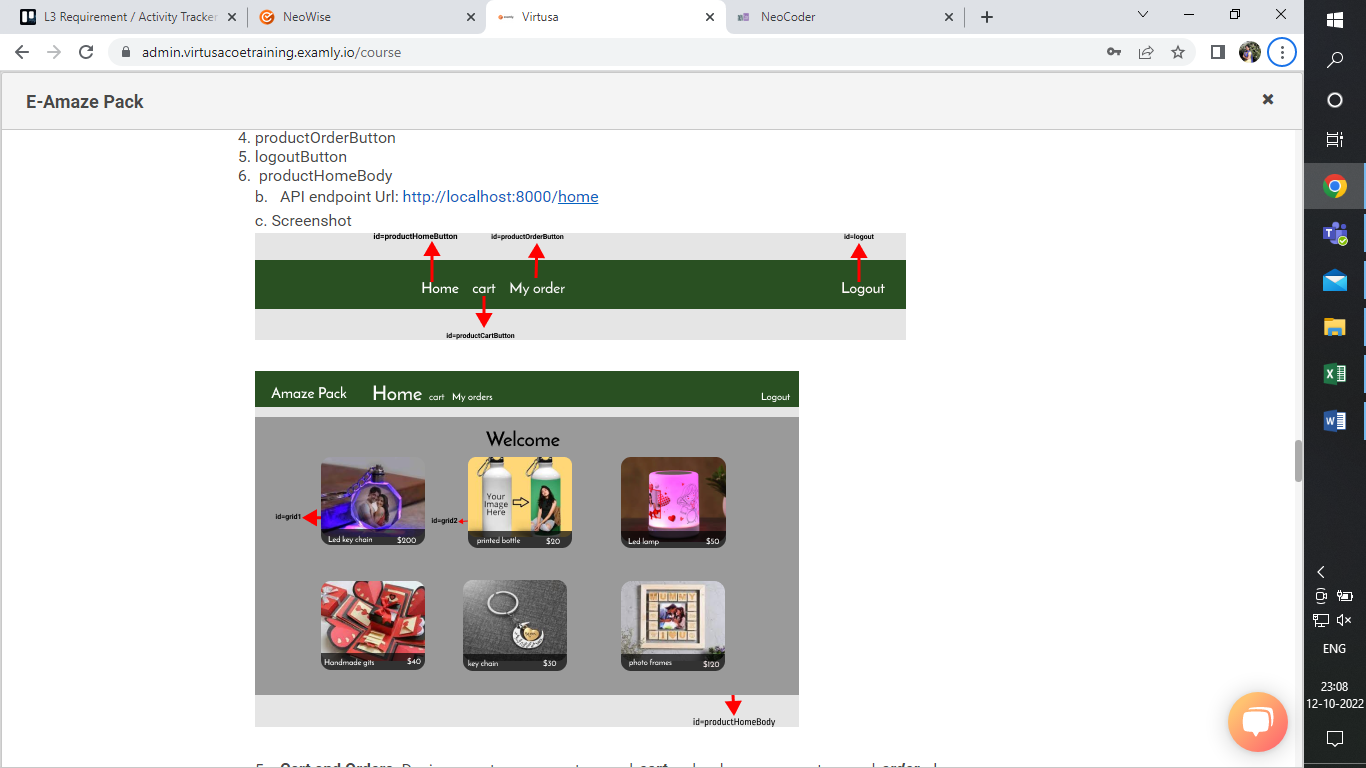
4.   **Dashboard / Home**: Design a home page component named (Name the component as ***homepage***for angular app whereas ***HomePage***for react app. Once the component is created in react app, name the jsx file as same as component name i.e HomePage.jsx file) that has the navigation bar and lists all the available products as grid elements with appropriate filter options.

a.   Ids:

1. userNavbar
2. productHomeButton
3. productCartButton
4. productOrderButton
5. logoutButton
6. productHomeBody

b.   API endpoint Url: [http://localhost:8000/](http://localhost:4200/login)home

c. Screenshot



5.   **Cart and Orders**: Design a cart component named ***cart***and order component named ***order***where we can see the cart items and see the items ordered after placing an order.

a.   Ids

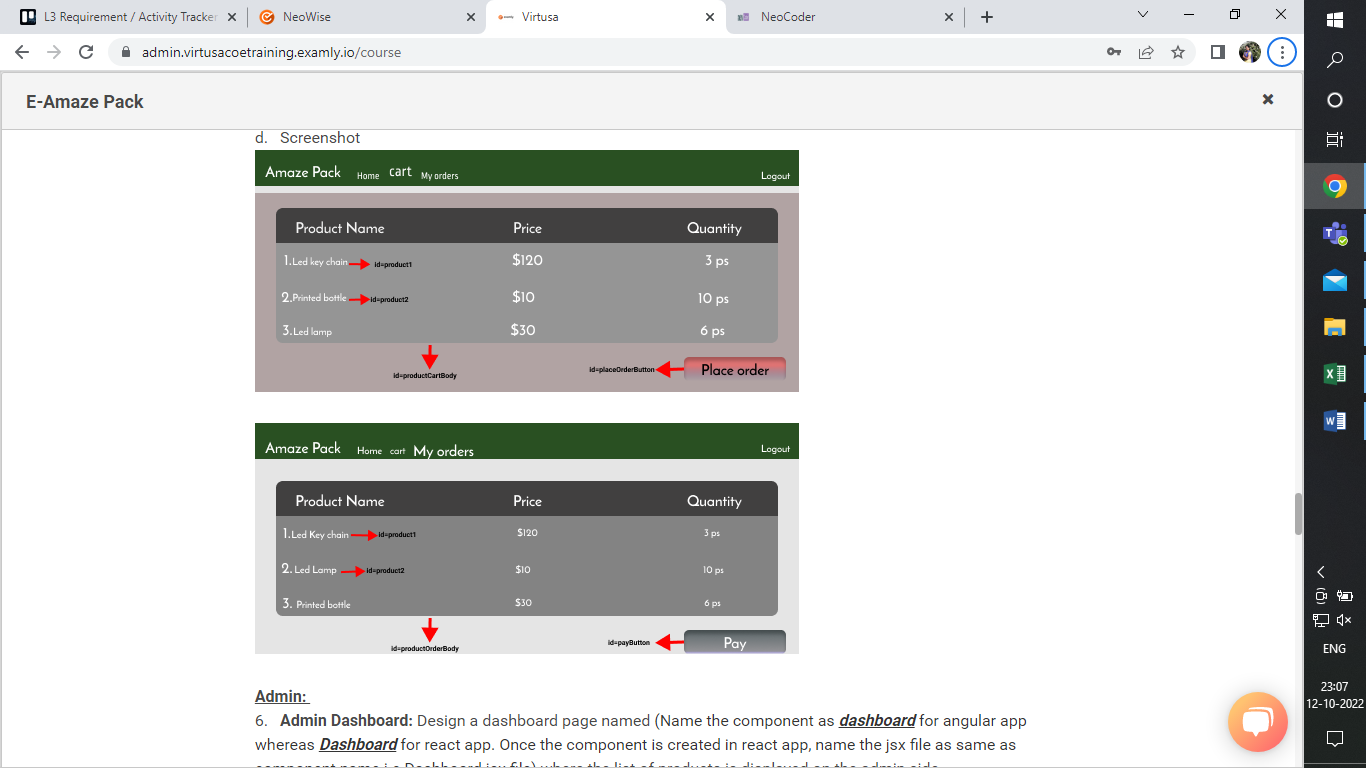
                              i.   productCartBody

                              ii.   productOrderBody

b.   API endpoint Url: [http://localhost:8000/](http://localhost:4200/login)cart

c.   API endpoint Url: [http://localhost:8000/orders](http://localhost:4200/orders)

d.   Screenshot



**Admin:**

6.   **Admin Dashboard:** Design a dashboard page named (Name the component as ***dashboard*** for angular app whereas ***Dashboard*** for react app. Once the component is created in react app, name the jsx file as same as component name i.e Dashboard.jsx file) where the list of products is displayed on the admin side.

a. **Admin Navigation**: Design a navigation component (Name the component as ***adminhomepage*** for angular app whereas ***AdminHomePage***for react app. Once the component is created in react app, name the jsx file as same as component name i.e AdminHomePage.jsx file) that can navigate to products and orders.

      i.Ids:

1. adminNavbar
2. adminProductButton
3. adminOrderButton
4. logoutButton

b. **Add Product**: Design an add product component (Name the component as ***addproduct*** for angular app whereas ***AddProduct*** for react app. Once the component is created in react app, name the jsx file as same as component name i.e AddProduct.jsx file) in which the admin can add new products to the inventory.

       i.Ids:

1.addProductBody

2.productName

3.productPrice

4.productDescription

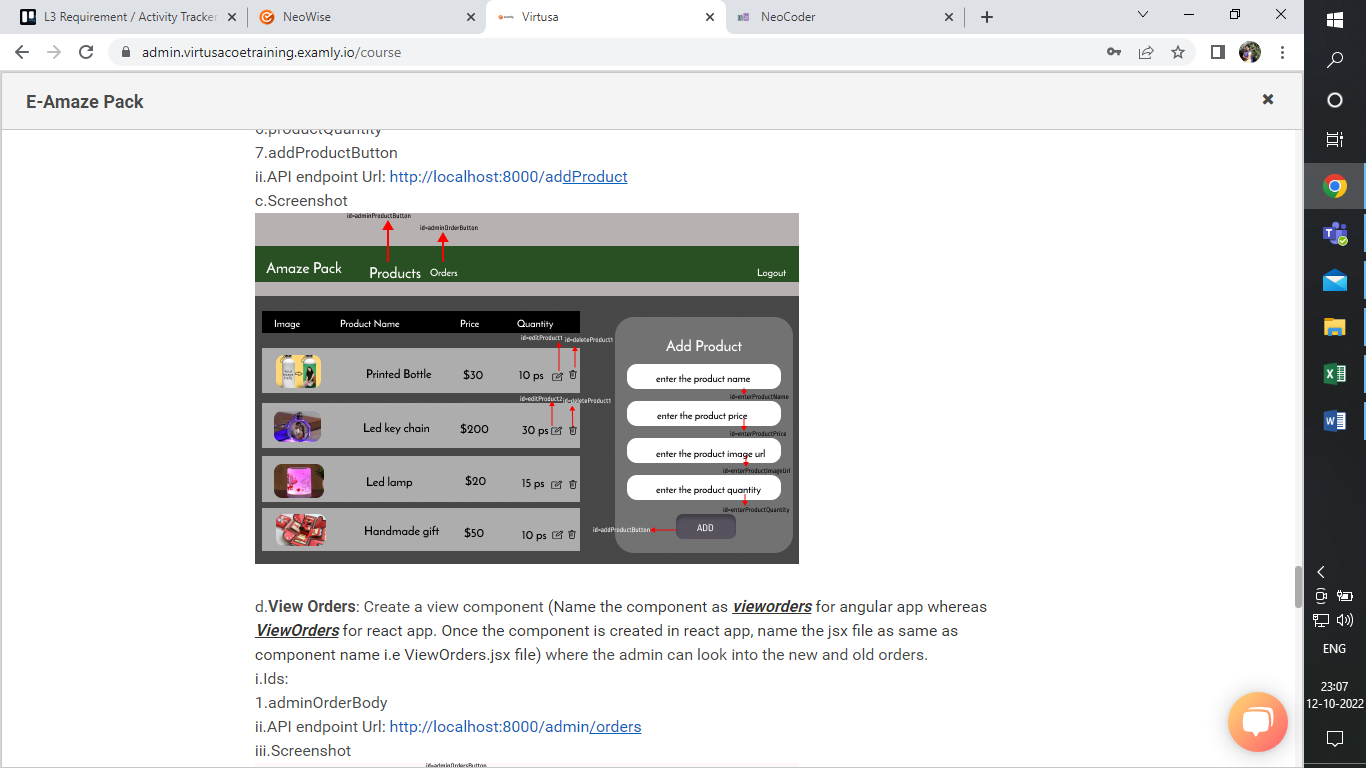
5.productImageURL

6.productQuantity

7.addProductButton

ii.API endpoint Url: [http://localhost:8000/ad](http://localhost:4200/ad)dProduct

c.Screenshot



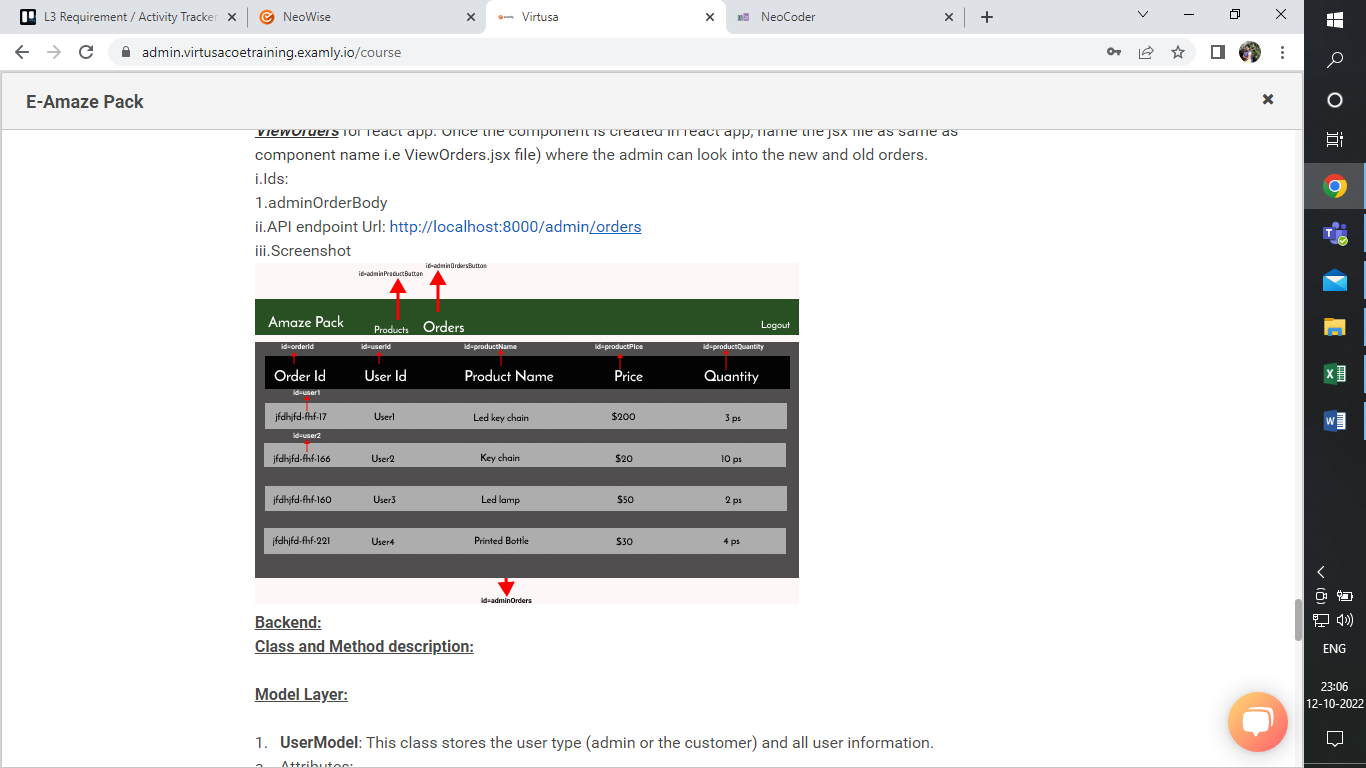
d.**View Orders**: Create a view component (Name the component as ***vieworders*** for angular app whereas ***ViewOrders*** for react app. Once the component is created in react app, name the jsx file as same as component name i.e ViewOrders.jsx file) where the admin can look into the new and old orders.

i.Ids:

1.adminOrderBody

ii.API endpoint Url: [http://localhost:8000/admin](http://localhost:4200/admin)/orders

iii.Screenshot



**Backend:**

**Class and Method description:**

**Model Layer:**

1.   **UserModel**: This class stores the user type (admin or the customer) and all user information.

a.   Attributes:

                                                         i.   email: String

                                                       ii.   password: String

                                                      iii.   username: String

                                                      iv.   mobileNumber: String

                                                       v.   active: Boolean

                                                      vi.   role: String

2.   **LoginModel**: This class contains the email and password of the user.

a.   Attributes:

                                                         i.   email: String

                                                       ii.   password: String

3.   **ProductModel**: This class stores the details of the product.

a.   Attributes:

                                                         i.   productId: String

                                                       ii.   imageUrl: String

                                                      iii.   productName: String

                                                      iv.   price: String

                                                       v.   description: String

                                                      vi.   quantity: String

4.   **CartModel**: This class stores the cart items.

a.   Attributes:

                                                         i.   cartItemID: String

                                                       ii.   userId: UserModel

                                                      iii.   productName: String

                                                      iv.   quantity: int

                                                       v.   price: String

5.   **OrderModel**: This class stores the order details.

a.   Attributes:

                                                         i.   orderId: String

                                                       ii.   userId: String

                                                      iii.   productName: String

                                                      iv.   quantity: int

                                                       v.   totalPrice: String

                                                      vi.   status: String

                                                     vii.   price: String

**Controller Layer:**

6.   **SignupController**: This class control the user signup

a.   Methods:

                 i.   saveUser(UserModel user): This method helps to store users in the database and return true or false based on the database transaction.

7.   **LoginController**: This class controls the user login.

a.   Methods:

              i.   checkUser(LoginModel data): This method helps the user to sign up for the application and must return true or false.

8.   **ProductController**: This class controls the add/edit/update/view products.

a.   Methods:

                 i.   List<ProductModel> getProduct(): This method helps the admin to fetch all products from the database.

                ii.   List<ProductModel> getHomeProduct(): This method helps to retrieve all the products from the database.

               iii.   ProductModel productEditData(String id): This method helps to retrieve a product from the database based on the productid.

            iv.   productEditSave(ProductModel data): This method helps to edit a product and save it to the database.

               v.   productSave(ProductModel data): This method helps to add a new product to the database.

               vi.   productDelete (String id): This method helps to delete a product from the database.

9.   **CartController**: This class helps in adding products to the cart, deleting the products from the cart, updating items in the cart.

a.   Methods:

 i.   addToCart(String Quantity, String id): This method helps the customer to add the product to the cart.

                 ii.   List<CartTempModel> showCart(String id): This method helps to view the cart items.

                 iii.   deleteCartItem(String id): This method helps to delete a product from the cart.

10. **OrderController**: This class helps with the orders such as save order/ place an order/ view order.

a.   Methods:

                   i.  List<OrderTemp> getUserProducts(String id): This method helps to list the orders based on the user id.

                   ii.  saveProduct(String id): This method helps to save the cart items as an order.

iii. placeOrder(OrderModel order): This method helps to place an order by the customer.