Neo Musicals

Objective:

Neo Musicals is an online web application to be built as a product that can be catering to various customers who require purchasing musical instruments.

Users of the System:

1. Admin
2. Customer

Functional Requirements:

- Build an application that customers can access and purchase musical instruments online.
- The application should have signup, login, profile, dashboard page, and product page.
- This application should have a provision to maintain a database for customer information, order information and product portfolio.
- Also, an integrated platform required for admin and customer.
- Administration module to include options for adding / modifying / removing the existing product(s) and customer management.
- **Based on month(odd/even) return the discount price.**

While the above ones are the basic functional features expected, the below ones can be nice to have add-on features:

- Filters for products like Low to High or showcasing products based on the customer's price range, specific brands etc.
- Email integration for intimating new personalized offers to customers.
- Multi-factor authentication for the sign-in process
- Payment Gateway

Output/ Post Condition:

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

Non-Functional Requirements:

Security	● App Platform –UserName/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	● IE 7+ ● Mozilla Firefox Latest – 15 ● Google Chrome Latest – 20 ● Mobile Ready

Technology Stack

Front End	Angular 7+ Google Material Design Bootstrap / Bulma
Server Side	Spring Boot Spring Web (Rest Controller) Spring Security Spring AOP Spring Hibernate
Core Platform	OpenJDK 11
Database	MySQL or H2

Platform Pre-requisites (Do's and Don'ts):

1. The angular app should run in port 8081. Do not run the angular app in the port: 4200.
2. Spring boot app should run in port 8080.

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.

4. 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:4200/signup or 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.

Project Tasks:

API Endpoints:

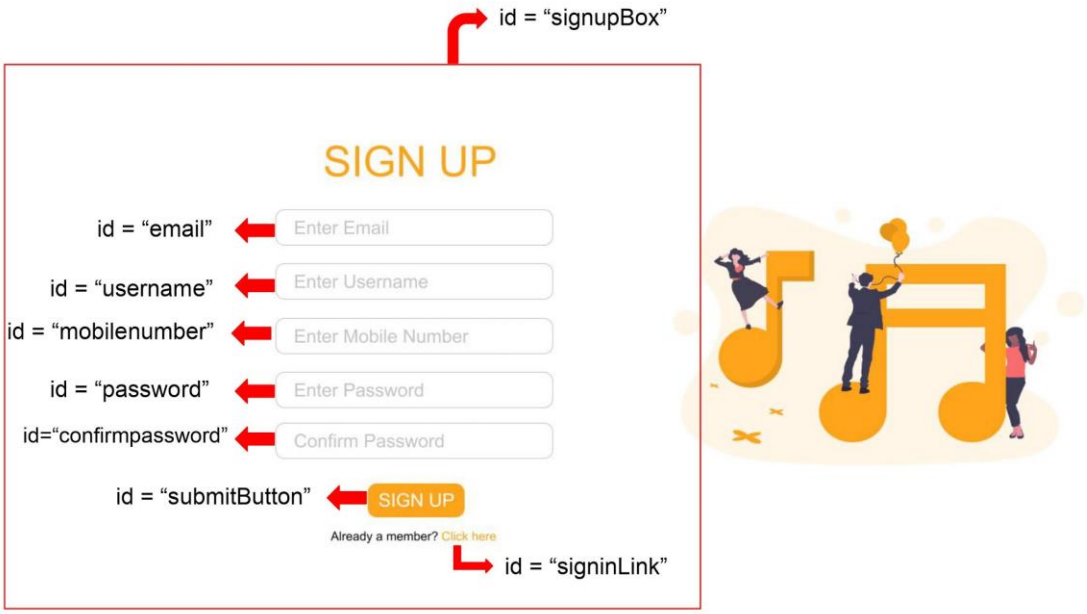
USER			
Action	URL	Method	Response
Login	/login	POST	true/false
Signup	/signup	POST	true/false
Get All Products – Home	/home	GET	Array of Products
Add to cart	/home/{id}	POST	Item added to cart
Cart Items	/cart/{id}	GET	Array of Cart Items
Delete cart Item	/cart/delete	POST	Cart Deleted
Cart to Orders	/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:

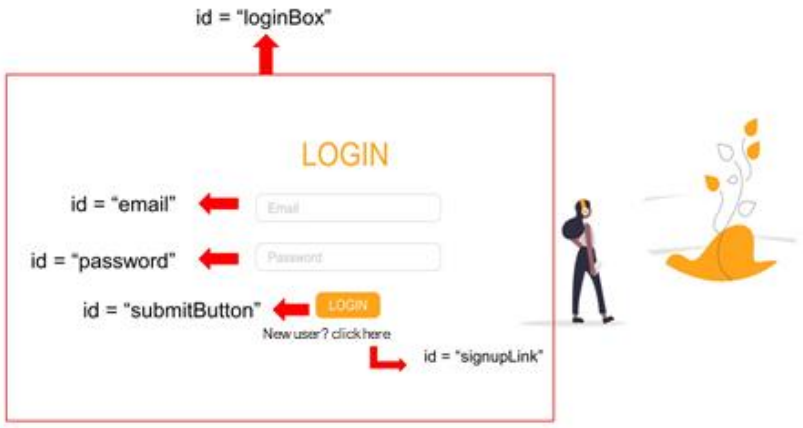
Customer:

1. Signup: Design a signup page component where the new customer has options to sign up by providing their basic details.
    a. Ids:
        i. email
        ii. username
        iii. mobilenumber
        iv. password
        v. confirmpassword
        vi. submitButton
        vii. signinLink
        viii. signupBox
    b. API endpoint Url: http://localhost:4200/signup
    c. Output screenshot:

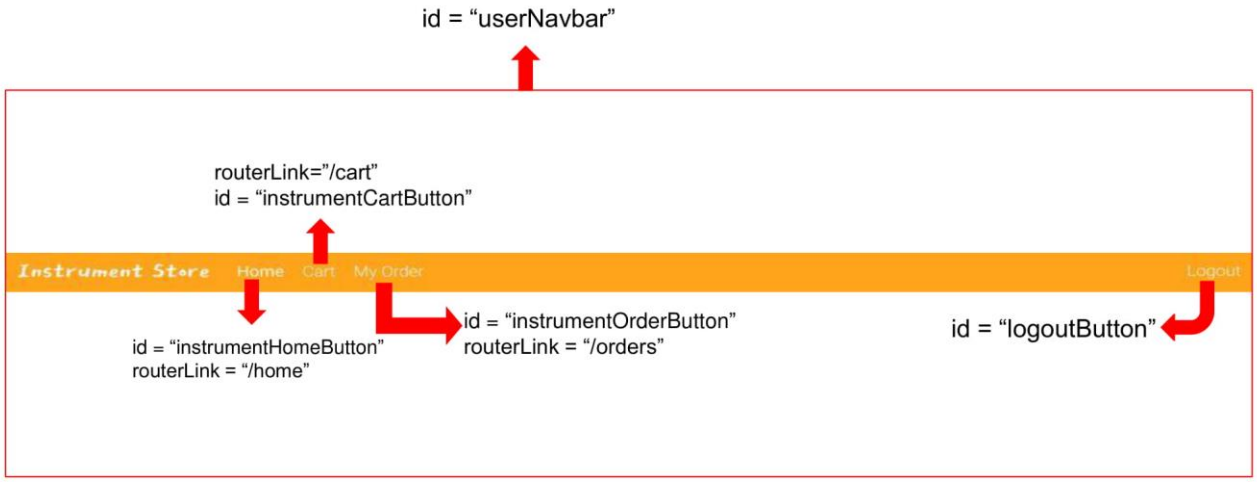


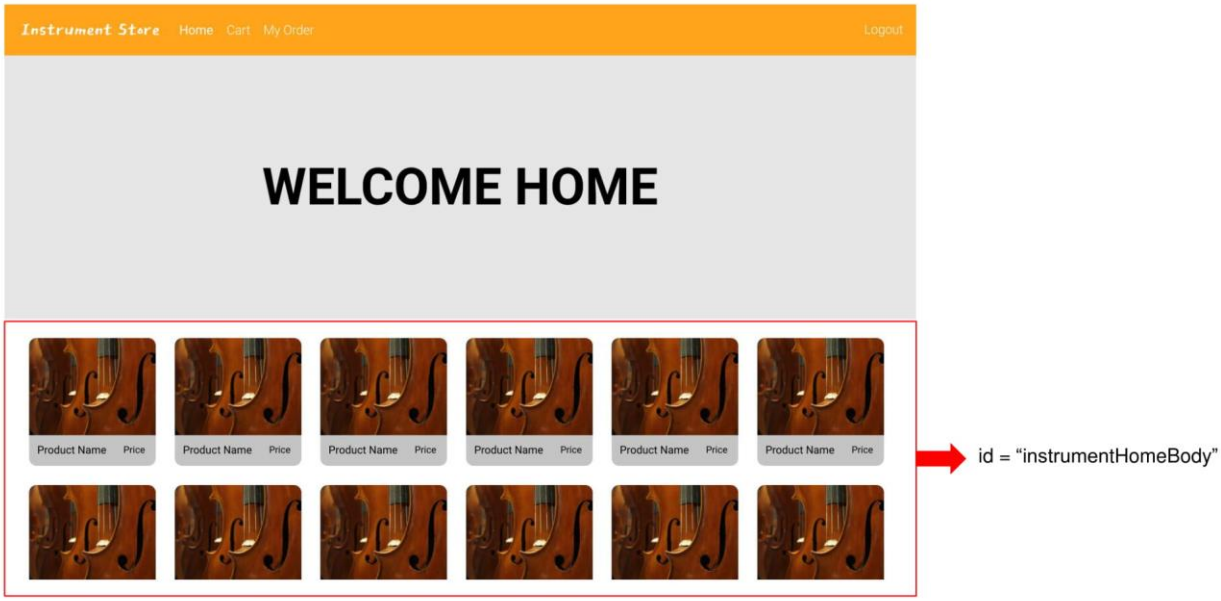
2. Login: Design a login page component where the existing customer can log in using the registered email id and password.

a. Ids:
    i. email
    ii. password
    iii. submitButton
    iv. signupLink
    v. loginBox
b. API endpoint Url: http://localhost:4200/login
c. Output screenshot:

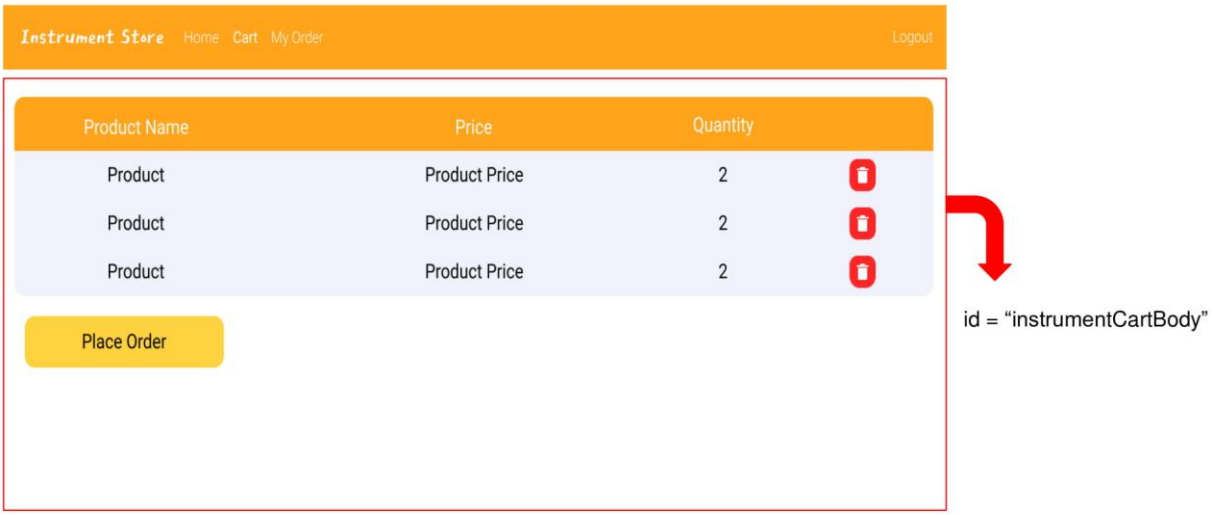


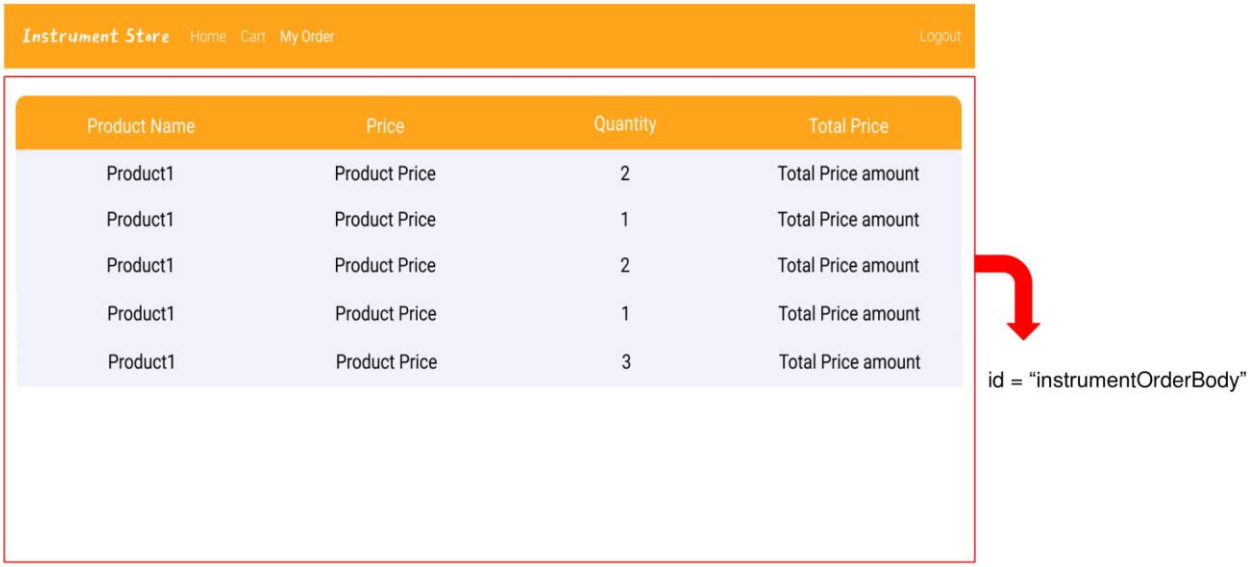
3. Dashboard / Home: Design a home page component that has the navigation bar and lists all the available products as grid elements with appropriate filter options.
    a. Ids:
        i. userNavbar
        ii. instrumentHomeButton
        iii. instrumentCartButton
        iv. instrumentOrderButton
        v. logoutButton
        vi. instrumentHomeBody
    b. API endpoint Url: http://localhost:4200/home
    c. Screenshot





4. Cart and Orders: Design a cart component and order component where we can see the cart items and see the items ordered after placing an order.
    a. Ids
        i. instrumentCartBody
        ii. instrumentOrderBody
    b. API endpoint Url: http://localhost:4200/cart
    c. API endpoint Url: http://localhost:4200/orders
    d. Screenshot

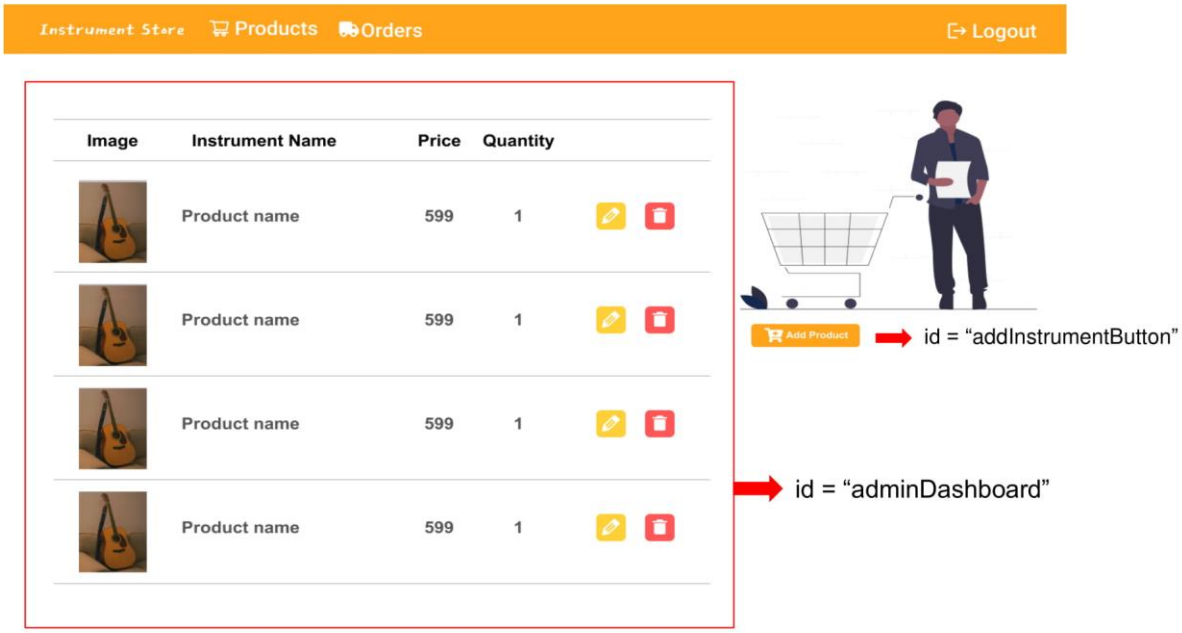




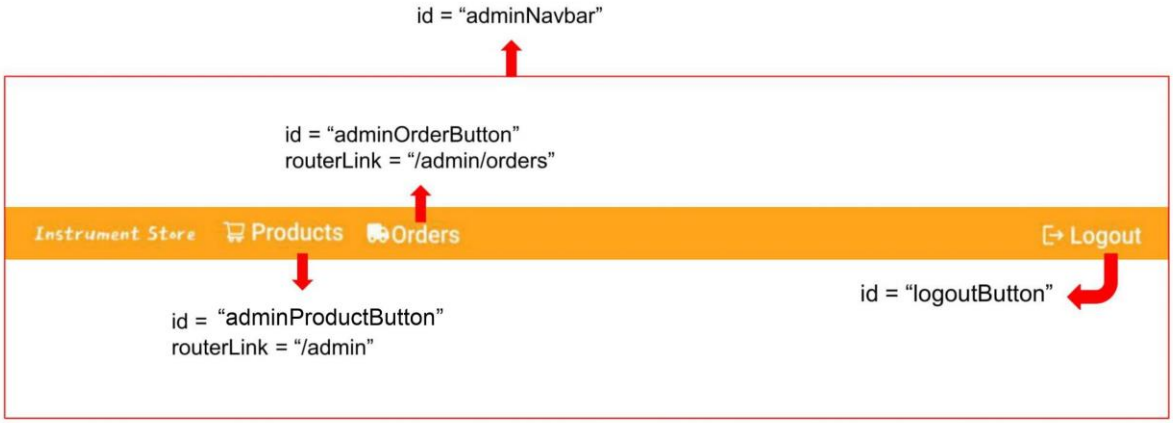
Admin:

5. Admin Dashboard: Design a dashboard page where the list of products is displayed on the admin side.
    a. Ids
        i. addInstrumentButton
        ii. adminDashboard
    b. API endpoint Url: http://localhost:4200/admin

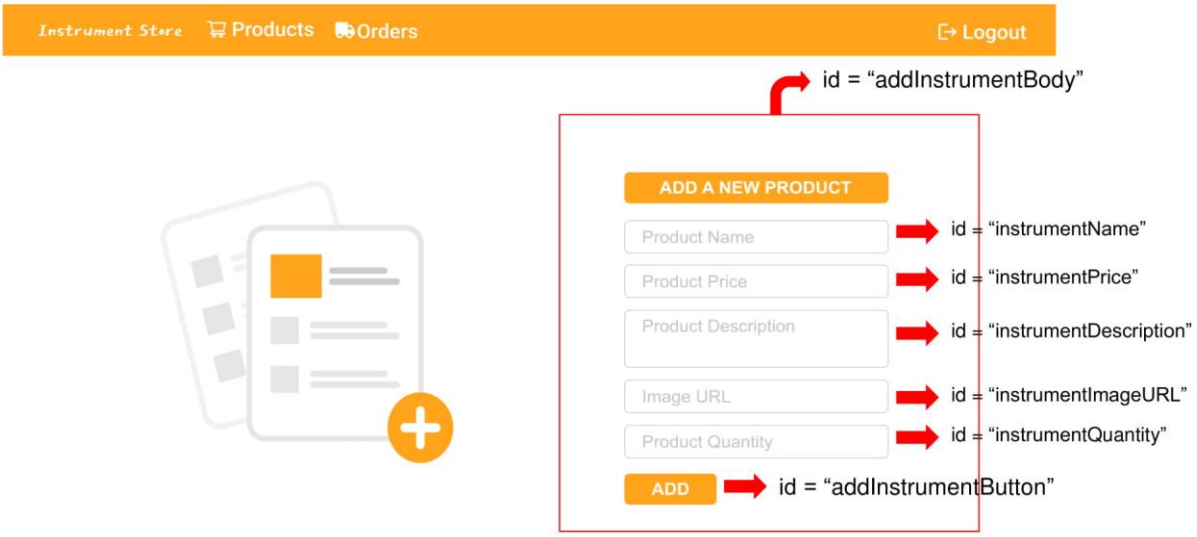
c. Screenshot



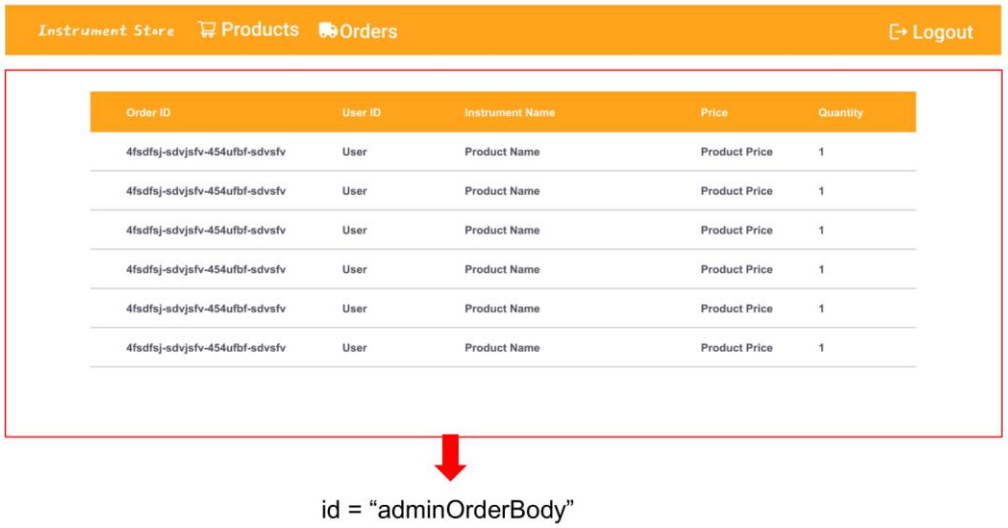
6. Admin Navigation: Design a navigation component that can navigate to products and orders.
    a. Ids:
        i. adminNavbar
        ii. adminProductButton
        iii. adminOrderButton
        iv. logoutButton
    b. Screenshot:



7. Add Product: Design an add product component in which the admin can add new products to the inventory.
    a. Ids:
        i. addInstrumentBody
        ii. instrumentName
        iii. instrumentPrice
        iv. instrumentDescription
        v. instrumentImageURL
        vi. instrumentQuantity
        vii. addInstrumentButton
    b. API endpoint Url: http://localhost:4200/addProduct
    c. Screenshot



8. View Orders: Create a view component where the admin can look into the new and old orders.
    a. Ids:
        i. adminOrderBody
    b. API endpoint Url: http://localhost:4200/admin/orders
    c. 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
        vii. cart: CartModel
        viii. ordersList: List<OrderModel>

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.

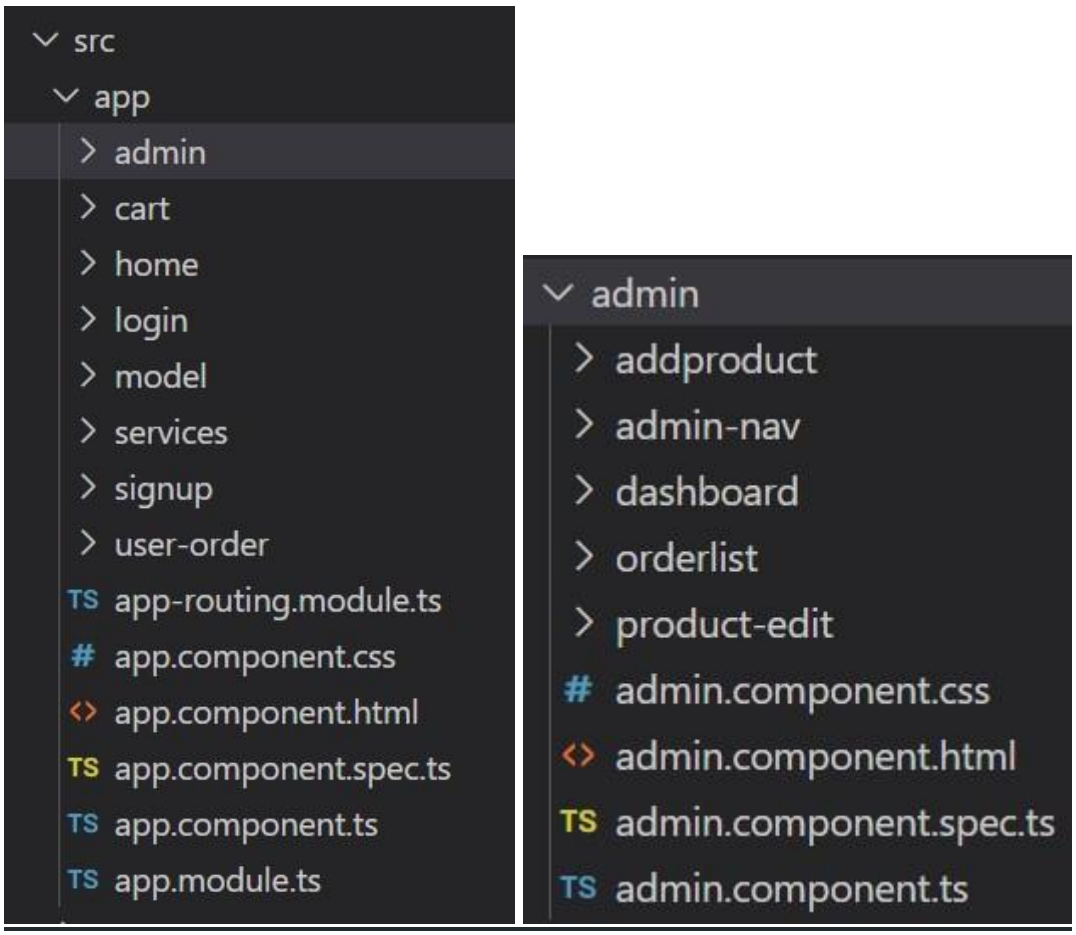
v. productSave(ProductModel data): This method helps to add a new product to the database.

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.

Angular Folder Structure:



NOTE:

You should create the above folder structure mandatorily to pass the test cases and you can also create extra components if you need.

Workflow Prototypes:

Admin Flow
https://www.figma.com/proto/KUXrU8hz3tRaWNiGQEAoqR/Music-Instrument-Store-Admin-Flow?node-id=1%3A2&viewport=439%2C389%2C0.15327335894107819&scaling=scale-down

User Flow
https://www.figma.com/proto/NjDlEqLaTxnqQ1Uwl5I4Yk/Music-Store-User-Flow?node-id=1%3A2&viewport=383%2C-83%2C0.27936112880706787&scaling=min-zoom