Sports Store Project

Step by Step

- After making the navigation bar and installing JSON server
- Download the json file from the link given
- run the json server
    - npm run json
    - check the json content using "localhost:3500"
    - It will show the products array and orders (empty)

- Now make a folder "**model**" -> It will describe the type of data that it contains
    - Which means classes inside Model folder will describe the products that are sold in this store and the orders placed by the customers
    - Inside "model" make a simple file "product.model.ts"
        - This file will contain the Product class and a constructor with some arguments
    - Next step is to Create a data source
        - Since the application is going to get the data from this data source created by us (This will be called as Static Data Source)
        - Create a file static.datasource.ts
            - It define a method called **getProducts()** which will return the product from our data source
            - The return type of **getProducts()** is an array of Product objects of type **Observable**
                - It is a class that supports message passing b/w **publisher** and **subscriber**
                - **Publisher:** someone who creates an observable instance by defining a subscriber function
                    - In this case, ***getProducts()*** is the subscriber function
                - **Subscriber:** It is the function that gets executed when someone calls the subscribe() method
                - This subscriber() is a predefined function inside Observable class
            - This Observable class is imported from RXJS packages and supports asynchronous programming (as we've discussed in the class)
            - It also contain one operator/function "***from()"*** which will return an array from the observable object in a sequence
            - So the Observable instance represents the asynchronous task that'll be execute at some point the future
                - Here that asynchronous task will be used to generate HTTP requests (Later when we integrate it with JSON server)

        - Now, this static.datasource.ts class is a service so we're using a decorator called **@Injectable()** to tell every component that this is a service class and use this class you need to call it using Dependency Injection technique.
            - We need to add the @Injector() decorator in a service in case where our services have some dependency itself. [see product.repository.ts class]
- Next step is to create a **Model Repository**
    - The data source (static.datasource.ts) will provide data to our application but access to that data should go through the data **repository**
    - The data repository is responsible for distributing the data to each component of the application
    - So the logic of what data will go on which component will go here
    - Create a file product.repository.ts inside Model folder
        - Again this repository file will also be a service file -> that's why it uses the @Injectable() decorator
        - First it will create a **products** field which is an array of type **Product**
        - Second field is **categories** which is array of type String
        - Inside its constructor we'll use the static datasource service {since we need the data from this service itself}
        - We'll use the ***getProducts()*** and use the subscribe function (of the observable class from rxjs) to get the data from the Observable class
            - **Map** operator -> It will segregate the array into categories
            - **Filter** Operator -> It will filter out the categories uniquely
        - Then it'll define the ***getProducts() and getProduct()*** function to get the products ID wise and Category wise
        - Lastly, it'll define a function ***getCategories()*** *which will return the list of categories*
- Creating the **Feature Module**
    - Create a module file [**model.module.ts]**(inside Model folder)
    - Under Providers field write the names of both the services -> ProductService and StaticDataSource service
    - The @NgModule decorator tells angular which classes will be used as Services
    - We're separating out this Module from our App Module as we don't want to mix the Data Fetching code and Component Code together
    - Also, if we scale our application, others can also use this file
- Starting the Store
    - The basic structure of our store will be a two column with categories on the left and products list on the right side
    - The starting point of our application will be **Home.component.ts** which will contain the data as well as logic
        - It's constructor will contain the service call to ProductRepository

            - It will contain functions to get Products() and Categories() from the ProductRepository service class
        - Next we will define the presentation logic in home.component.html
    - Creating the Home Feature Module
        - Create a file home.module.ts
        - Import different module classes into this
    - Next Step is to import the Home.module.ts inside app.module.ts
        - Inside the imports field make the necessary changes.
    - Finally Run the product and see that home.component.html will fetch the numbe of categories as well as products from ProductRepository

- **Adding Store Features and Product Details**
    - **Displaying the Product Details**
    - Inside Home.component.HTML
        - Go under the div of products
        - Make sure to change the text color from the out div class so that products could be visible
        - Using the ngFor directive -> access products array (from repository service) and store each product in a local variable
        - print name and description of every product using
            - product.name (this name field is accessed from product.model.ts folder (where product constructor is defined)
        - And we use pipes in the currency field
        - Finally run the project
            - It will succesfully fetch the data and print all the 15 products along with their description
    - **Adding Category Selection**
        - Here we'll filter the list of products by Category
        - Open Home.component.ts
            - Define a public field **selectedCategory**
            - Inside ***products()***
                - return *getProducts(this.selectedCategory)*
            - Along with that, make another function -> changeCategory() which will take parameter newCategory of type string (using ? means this argument is optional)
                - Inside that, we're assigning value received by newCategory to selectedCategory
        - Go back to home.component.html
            - Inside categories div
                - add a click event which will call the **changeCategory()** function
                - Then make buttons and use ngFor to go through all the categories in the categories[] and place a click event on each button
                    - Such that when user click on each category -> changeCategory will call

        - Here we've use the property binding [class.active] = selectedCategory -> That button will activate
    - **Adding Pagination**
        - Now to inside home.component.ts
            - Add two more fields
                - productsPerPage = 5
                - selectedPage = 1
        - Filtering the products by category has helped make the product list more manageable, but a more typical approach is to break the list into smaller sections and present each of them as a page, along with navigation buttons that move between the pages.
        - Add the following functions inside **"home.component.ts"**
            - ***ChangePage()***
                - Sets the page number for the *selectedPage* variable
            - ***changePageSize()***
                - *Changes page size (3,4,6,8)*
            - ***getPageNumbers()***
                - *Contain logic for navigating through different pages*

- **Creating a Shopping Cart**
    - Inside our model folder we'll add one more file with the name - "cart.model.ts"
    - The next step is to register the Cart class in -> model.module.ts
        - So that any other part of the application can use this cart service in its component
- **Creating Cart Summary Component**
    - We'll create the cart summary component inside our navbar component
        - To do that, create a component file cartSummary.component.ts inside Navbar folder
            - Inside its constructor, we'll use the Cart service
        - Create a html template for this component "cartSummary.component.html"
            - This template uses the Cart object provided by its component to display the number of items in the cart and the total cost
        - After that, register this component inside App.module
    - Integrating Cart into Navbar
        - Go to navbar.component.html
            - Below Product Description add <cart-summary> tag
        - Go to to home.component.ts
            - add Cart service into its constructor
            - add function addtocart() which will display number of items added