Sports Store V2

Authentication

- **Implementing Authentication**
  - In this section, we'll add support for authenticating the user by sending an http request to our json server
  - The Authentication Credentials Supported by the RESTful Web Service
  - Admin Credentials are:
    - Username: admin | Password: secret
  - First we'll send the authentication request to /login URL
    - If the correct credentials are sent to the /login URL, then the response from the RESTful web service (JSON Server) will contain a JSON object like this:
    - {

      "success": true,
      "token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJkYXRhIjoiYWRtaW4iLCJleHBpcmVz
      SW4iOiIxaCIsImlhdCI6MTQ3ODk1NjI1Mn0.lJaDDrSu-bHBtdWrz0312p_DG5tKypGv6cA
      NgOyzlg8"
      }

    - The success property describes the outcome of the authentication operation, and the token property contains the JWT, which should be included in subsequent requests using the Authorization HTTP header in this format:
  - Add authentication in **rest.datasource.ts**
    - It will return whether the authentication is successful or not
    - Add the ***authenticate()*** function
      - It will check the authenticate process and returns success (by checking the content inside *auth_token*)
  - Then make a service file **auth.service.ts**
    - It'll determine whether the application has been authenticated
    - The *authenticate* method receives the user's credentials and passes them on to the data source *authenticate* method, returning an Observable that will yield true if the authentication process has succeeded and false otherwise.
  - Next register this service file inside **mode.module.ts**
  - Then inside auth.component.ts
    - Import the authenticate service and write the authentication logic inside NgForm
    - We'll pass the credentials to ***subscribe(),*** which will take the response and pass it through **/admin/main**
  - For proper authentication

        - we'll add a file **auth.guard.ts** inside admin folder
        - It'll make sure that only the specified credentials are verified for logging into the admin component
        - Register this service inside **admin.module.ts**

- **Extending the Data Source & Repositories**
    - Our next step is to send the requests to the JSON server for making changes in the order and product repository classes
    - We're going to add some new operations in **rest.datasource.ts** file [under model folder]
        - Here, we'll add functions for
            - saveProduct()
            - deleteProduct()
            - updateProduct()
            - getOrder()
            - deleteOrder()
            - updateOrder()
    - Next inside **product.repository.ts** we'll add the logic for saving and deleting products
        - along with that we've to make changes inside the constructor by replacing ***StaticDatasource*** service with ***RestDatasource*** service
        - Also remember to import the **RestDatasource** service in each of those files
    - Next, make the changes inside ***order repository*** by adding methods that allows orders to be modified and deleted
        - open **order.repository.ts**
            - write code for load, save, update and delete orders
            - Also remember to change the **Staticdatasource** to **Restdatasource**
        - The order repository defines a **loadOrders** method that gets the orders from the repository and is used to ensure that the request isn't sent to the RESTful web service until authentication has been performed
- **Creating Administration Feature Structure**
    - Inside the admin folder add a file ***productTable.component.ts***
        - It will show the list of products and gives the option to edit/delete product
    - Then add the file ***productEditor.component.ts*** inside admin
        - allow the user to enter the details required to create or edit a component
    - To create the component that will be responsible for managing customer orders, I added a file called orderTable.component.ts to the src/app/admin folder
- **Preparing the common content and feature module**
    - The components created in the previous section will be responsible for specific features. To bring those features together and allow the user to navigate between them, we need to modify the template of the placeholder component that I have been using to demonstrate the result of a successful authentication attempt
    - Replacing the contents of ***admin.component.html***

        - This template contains a router-outlet element that will be used to display the components from the previous section. There are also buttons that will navigate the application to the ***/admin/main/products*** and ***/admin/main/orders*** URLs, which will select the products or orders features. These buttons use the **routerLinkActive** attribute, which is used to add the element to a CSS class when the route specified by the **routerLink** attribute is active.
    - We'll implement the logout method inside ***admin.component.ts*** file
    - Next we will implement some paths inside ***admin.module.ts***
    - Replacing Content in the ***productTable.component.ts*** File in the ***src/app/admin*** Folder
        - we'll add the logic for showing the products list to the admin
- **Implementing the Product Feature**
    - The initial administration feature presented to the user will be a list of products, with the ability to create a new product and delete or edit an existing one
    - We'll remove the existing placeholder template from the ***productTable.component.ts*** and write the functionality for *getProducts() & deleteProducts()*
    - The other operations will be handled by the editor component, which will be activated using routing URLs in the component's template. To provide the template, we've added a file called ***productTable.component.html*** in the **src/app/admin** folder.
    - The template contains a table that uses the **ngFor** directive to generate a row for each product returned by the component's **getProducts()** method. Each row contains a Delete button that invokes the component's delete method and an Edit button that navigates to a URL that targets the editor component. The editor component is also the target of the Create New Product button, although a different URL is used
- **Implementing the Product Editor**
    - Components can receive information about the current routing URL and adapt their behavior accordingly. The editor component needs to use this feature to differentiate between requests to create a new component and edit an existing one
    - Angular will provide an **ActivatedRoute** object as a constructor argument when it creates a new instance of the component class and that can be used to inspect the activated route. In this case, the component works out whether it should be editing or creating a product and, if editing, retrieves the current details from the repository.
    - There is also a save method, which uses the repository to save changes that the user has made. To provide the component with a template, I added a file called ***productEditor.component.html*** in the **src/app/admin** folder
    - In the same way, we can implement the Orders feature
        - See the template code inside ***orderTable.component.html***
- Finally, run your project using command: **ng serve**