Project Title

"YourShop: A Basic E-Commerce Demo Site with Angular Routing and CRUD Operations"

Introduction

"YourShop" is a basic e-commerce demo site built with Angular. It showcases key features of Angular such as routing, child routes, and services. The application includes user authentication with login and registration functionality. It also demonstrates CRUD operations, which are fundamental for any real-world application. Data persistence is achieved using Local Storage. This project serves as a practical guide for understanding and implementing these features in Angular.

Description

"YourShop" is an e-commerce demo site that serves as a practical example of an Angular application. It incorporates essential features such as routing and child routes for navigation, services for managing data flow, and Local Storage for data persistence. The application also includes user authentication mechanisms with login and registration features. Furthermore, it demonstrates CRUD (Create, Read, Update, Delete) operations, which are integral to any data-driven application. This project is an excellent resource for developers seeking to understand and implement these Angular features in a real-world context.

Installation

Follow these steps to set up and run the "YourShop" project:

1. **Install Node.js and npm**: Angular requires Node.js version 10.9.0 or later. You can download Node.js from here. This will also install npm (Node Package Manager) which is used to manage Node.js packages.

2. **Install Angular CLI**: The Angular CLI is a command-line interface tool that you use to initialize, develop, scaffold, and maintain Angular applications. Install it globally using npm:

   ```
   npm install -g @angular/cli
   ```

3. **Clone the repository**: Clone the "YourShop" project repository from GitHub to your local machine.

4. **Install project dependencies**: Navigate to the project directory and run the following command to install the necessary dependencies:

   ```
   npm install
   ```

5. **Run the application**: Start the development server with the following command:

   ```
   ng serve
   ```

The application will be available at `http://localhost:4200/`.

6. **Install Bootstrap**: This project uses Bootstrap for styling. Install it using npm:

```
npm install bootstrap
```

Then, add the Bootstrap CSS file in the "styles" array inside the `angular.json` file:

```
"styles": [
  "node_modules/bootstrap/dist/css/bootstrap.min.css",
  "src/styles.css"
],
```

Remember to restart the server after installing Bootstrap. You can stop the server by pressing `Ctrl+C` in the terminal, and start it again with `ng serve`.

Use Cases and Functionality

"YourShop" is designed to demonstrate a variety of use cases common in e-commerce applications. Here are the main functionalities provided by the application:

User Registration and Login

User Registration

The following code snippet from `login.component.ts` handles user registration in the application. It creates a `signUp` object to hold the user's information and an array `signUpArray` to store these objects. The `onsignUp()` method is called when the user submits the registration form. This method adds the `signUp` object to the `signUpArray` and then stores the array in Local Storage. An alert is displayed to the user to confirm successful registration.

Copilot Prompt

```
//create a SignUp Object
signUp: any = {
  userId: '',
  email: '',
  password: '',
};

//create an array to store the signUp objects
signUpArray: any = [];

onsignUp(): void {
  // Store data to local storage
```

```
    this.signUpArray.push(this.signUp);
    localStorage.setItem('signUpArray', JSON.stringify(this.signUpArray));
    alert('Registration successful');
  }
```

User Login

The following code snippet from `login.component.ts` handles user login in the application. It retrieves the `signUpArray` from Local Storage and checks if the entered user ID and password match any of the stored user data. If a match is found, the user is navigated to the layout component. If not, appropriate error messages are displayed.

Copilot Prompt

```
onSignIn(): void {
  // Retrieve data from local storage
  let signUpArray = JSON.parse(localStorage.getItem('signUpArray'));

  if (signUpArray) {
    let user = signUpArray.find(user => user.userId === this.signIn.userId);

    if (user) {
      if (user.password === this.signIn.password) {
        this.router.navigate(['/layout']); // Navigate to the layout component
      } else {
        alert('Incorrect password');
      }
    } else {
      alert('User not found');
    }
  } else {
    alert('No user registered');
  }
}
```

ngOnInit Lifecycle Hook

The following code snippet from `login.component.ts` shows the `ngOnInit` lifecycle hook. This method is called when Angular finishes initializing the component. In this case, it retrieves the `signUpArray` from Local Storage when the component is initialized.

Copilot Prompt

```
ngOnInit(): void {
  // Fetch data from local storage
  const storedData = localStorage.getItem('signUpArray');
  if (storedData) {
```

```
      this.signUpArray = JSON.parse(storedData);
    }
  }
```

CRUD Operations

The application allows for the creation, reading, updating, and deletion of data. This is demonstrated in the management of user information and product listings.

Class Products

The `Products` class in `newproduct.component.ts` defines the structure of a product object in the application.

This class has four properties: id, name, price, and description. The constructor method initializes these properties when a new instance of the Products class is created. ```

Copilot Prompt

```
//create a class Products and define the properties of the product

export class Products {
  id: number;
  name: string;
  price: string;
  description: string;
  constructor() {
    this.id = 0;
    this.name = '';
    this.price = '';
    this.description = '';
  }
}
```

ngOnInit

Copilot Prompt

```
// Fetch the list of products from Local Storage when the component is initialized

ngOnInit(): void {
  this.products = JSON.parse(localStorage.getItem('products') || '[]');
}
```

OpenModal and CloseModal

The openModal and closeModal methods handle the opening and closing of a modal, respectively.

Copilot Prompt

```
//create a function to open and close the modal

// Open the modal
openModal() {
  let modal = document.getElementById('myModal');
  if(modal!=null)
  modal.style.display = "block";
}

// Close the modal
closeModal() {
  let modal = document.getElementById('myModal');
  if(modal!=null)
  modal.style.display = "none";
}
```

Save Product Data

The saveProduct method saves a new product to Local Storage and updates the list of products.

Copilot Prompt

```
//create a function name saveProduct to save the product to the local storage and
push the product to the products array and close the modal also clear the input
fields and check for duplicate product also alert user when product is added
  saveProduct() {
    let products = JSON.parse(localStorage.getItem('products') || '[]');
    if (products.length > 0) {
      // Find the maximum product ID
      let maxId = Math.max(...products.map((product: Products) => product.id));
      this.productObj.id = maxId + 1;
    } else {
      this.productObj.id = 1;
    }

    if (products.some((product: Products) => product.id === this.productObj.id)) {
      alert('Product already exists');
      return;
    }

    products.push(this.productObj);
    localStorage.setItem('products', JSON.stringify(products));
    this.products = JSON.parse(localStorage.getItem('products') || '[]');
    this.productObj = new Products();
    this.closeModal();
  }
```

Delete Product

The deleteProduct method deletes a product from Local Storage and updates the list of products.

Copilot Prompt

```
//create a function to delete the product from the local storage and also from the
products array
  deleteProduct(product: any) {
    const confirmation = confirm('Are you sure you want to delete this product?');
    if (confirmation) {
      let products = JSON.parse(localStorage.getItem('products') || '[]');
      for (let i = 0; i < products.length; i++) {
        if (products[i].id == product.id) {
          products.splice(i, 1);
          break;
        }
      }
      localStorage.setItem('products', JSON.stringify(products));
      this.products = JSON.parse(localStorage.getItem('products') || '[]');
    }
  }
```

Edit Product

The editProduct method opens the modal for editing a product.

Copilot Prompt

```
//create a function to edit the product
  editProduct(product: any) {
    this.productObj = product;
    this.openModal();
  }
```

Update Product

The updateProduct method updates a product in Local Storage and updates the list of products.

Copilot Prompt

```
//create a function to update the product in the local storage and also in the
products array
  updateProduct() {
    let products = JSON.parse(localStorage.getItem('products') || '[]');
    for (let i = 0; i < products.length; i++) {
      if (products[i].id == this.productObj.id) {
```

```
          products[i].name = this.productObj.name;
          products[i].price = this.productObj.price;
          products[i].description = this.productObj.description;
          break;
        }
      }
      localStorage.setItem('products', JSON.stringify(products));
      this.products = JSON.parse(localStorage.getItem('products') || '[]');
      this.productObj = new Products();
      this.closeModal();
    }
```

LocalstorageService

The `localstorage.service.ts` file defines a service that handles operations related to Local Storage in the application.

The LocalstorageService class is a service that is provided in the root of the application. This means that it's a singleton and the same instance is used throughout the application.

The getProducts method retrieves the list of products from Local Storage. If the 'products' key does not exist in Local Storage, it returns an empty array. This method is useful for fetching the current list of products stored in the user's Local Storage.

Copilot Prompt

```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class LocalstorageService {

  constructor() { }

  // Copilot Prompt
  // Create a function to get the data from local storage key products
  getProducts(): any {
    return JSON.parse(localStorage.getItem('products') || '[]');
  }

}
```

CartserviceService

The `cartservice.service.ts` file defines a service that handles operations related to the shopping cart in the application.

Copilot Prompts

```
export class CartserviceService {

  constructor() { }

  //create an array to store the cart objects
  cartArray: any = [];

  //create a function to add to cart
  onAddToCart(product: any): void {
    this.cartArray.push(product);
    alert('Your product has been added to the cart!');
  }

  //create a function to get the total price of the products in the cart
  getTotalPrice(): any {
    let totalPrice = 0;
    for (let i = 0; i < this.cartArray.length; i++) {
      totalPrice += this.cartArray[i].quantity * this.cartArray[i].price;
    }
    return totalPrice;
  }

  //create a function to remove the product from the cart
  onRemoveFromCart(product: any): void {
    for (let i = 0; i < this.cartArray.length; i++) {
      if (this.cartArray[i].productId === product.productId) {
        if (this.cartArray[i].quantity > 1) {
          this.cartArray[i].quantity--;
        } else {
          this.cartArray.splice(i, 1);
        }
      }
    }
  }

  //create a function to clear the cart and remove all the products from the cart
  onClearCart(): void {
    this.cartArray = [];
    this.getTotalPrice();
    return this.cartArray;

  }


  //create a function to get the cart array
  getCartArray(): any {
    return this.cartArray;


  }
}
```

Productlist Component

The `productlist.component.ts` file defines a component that handles operations related to the product list in the application.

The ProductlistComponent class is a component that manages the product list operations in the application. It has a products property that stores the product objects.

The constructor of the ProductlistComponent class injects the LocalstorageService and CartserviceService services. It also initializes the products property with the products from Local Storage.

The ngOnInit method is a lifecycle hook that is called after Angular has initialized all data-bound properties of a directive. In this case, it retrieves the list of products from the LocalstorageService when the component is initialized.

The addToCart method adds a product to the cart. It checks if the product is already in the cart. If it is, it displays an alert message indicating that the product is already in the cart and returns. If the product is not in the cart, it adds the product to the cart by calling the onAddToCart method of the CartserviceService.

Copilot Prompt

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-productlist',
  templateUrl: './productlist.component.html',
  styleUrls: ['./productlist.component.css']
})
export class ProductlistComponent implements OnInit {

  //create an array to store the products
  products: any = [];

  constructor( private LocalstorageService: LocalstorageService, private
CartserviceService: CartserviceService) {

  }

  ngOnInit(): void {

    //get the data from the LocalstorageService
    this.products = this.LocalstorageService.getProducts();
  }

  //create a function to add the product to the cart and call the function
onAddToCart from the cart service and pass the product check for duplicate product
and alart user
  addToCart(product: any) {
    if(this.CartserviceService.cartArray.length > 0) {
      for (let i = 0; i < this.CartserviceService.cartArray.length; i++) {
        if (this.CartserviceService.cartArray[i].id == product.id) {
          alert('Product already in the cart');
```

```
            return;
          }
        }
      }
      this.CartserviceService.onAddToCart(product);
    }
  }
```

CartComponent

The `cart.component.ts` file defines a component that handles operations related to the cart in the application.

The CartComponent class is a component that manages the cart operations in the application.

The removeFromCart method removes a product from the cart by calling the onRemoveFromCart method of the CartserviceService.

The clearCart method clears the cart by calling the onClearCart method of the CartserviceService and then reloads the window.

The getTotalPrice method gets the total price of the products in the cart by calling the getTotalPrice method of the CartserviceService.

The OrderNow method places an order, displays an alert message indicating that the order has been placed successfully, clears the cart by calling the onClearCart method of the CartserviceService, and then reloads the window.

Copilot Prompt

```
export class CartComponent implements OnInit {

  constructor(private CartserviceService: CartserviceService) { }

  //create a function to remove the product from the cart and call the function onRemoveFromCart from the cart service and pass the product
  removeFromCart(product: any) {
    this.CartserviceService.onRemoveFromCart(product);
  }

  //create a function to clear the cart and call the function onClearCart from the cart service
  clearCart() {
    this.CartserviceService.onClearCart();
    window.location.reload();
  }

  //create a function to get the total price of the products in the cart and call the function getTotalPrice from the cart service
  getTotalPrice() {
```

```
    return this.CartserviceService.getTotalPrice();
  }

  //create a function name OrderNow to alert the user as "Order placed
successfully" and clear the cart
  OrderNow() {
    alert('Order placed successfully');
    this.CartserviceService.onClearCart();
    window.location.reload();
  }
}
```

License

```
put-your-code-here
```

Information about the license.