International Burch University

CEN 308 SOFTWARE ENGINEERING

PROJECT DOCUMENTATION

BH Fan Shop

Prepared by:

Ajdin Pašić

Nedim Bandžović

Proposed to:

Nermina Durmić, Assist. Prof. Dr.

Aldin Kovačević, Teaching Assistant

Sarajevo, 22.06.2022.

Table of Contents

1. Introduction

BH Fan Shop is a web application that offers online purchasing of different merchandise which is related to the football national team of Bosnia and Herzegovina such as jerseys, t-shirts, hoodies, balls and so on. Our application publishes any new merchandise that appears on the market both for kids, men and women in different sizes and colors. The official language of BH Fan Shop is English Language.

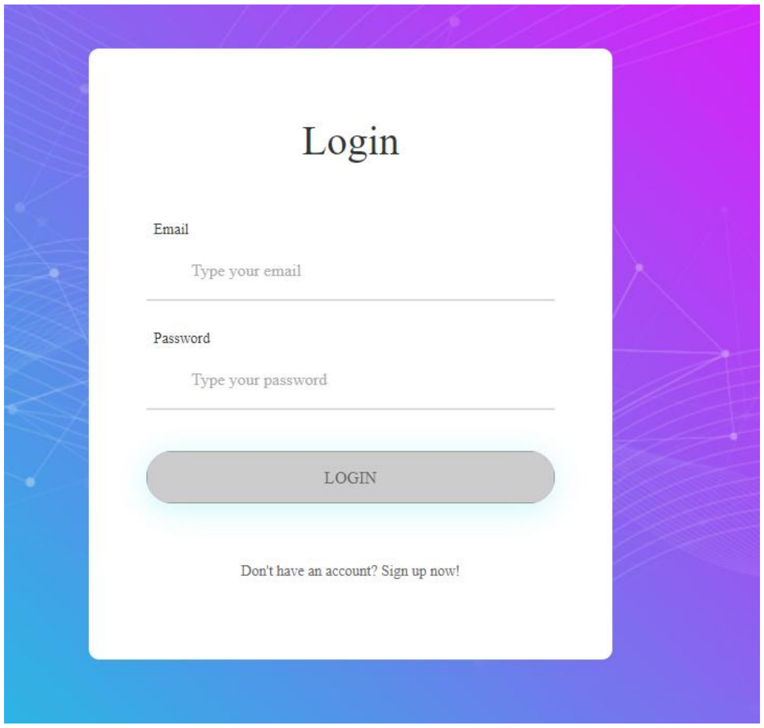
The application offers features like user registration and login, viewing the list of available products as well as displaying specific products. The user can also add products to user carts, edit information about the added product like quantity or even remove it from the cart. BH Fan Shop also has an easy filter and search option and besides these options, user has a detailed view about his previous purchases where he can easily check all the necessary information about all of his activities on the platform.

The application's code can be found on: https://github.com/ajdinpasic/SEProject

Selenium tests which are made for basic features of BH Fan Shop can be found here: https://github.com/nedimbandzovic/SETests

BH Fan Shop is deployed on Vercel and can be accessed here: https://fan-shop.vercel.app/products

BH Fan Shop's API is hosted on the Heroku platform and its link is: https://fan-shop-api.herokuapp.com/api-docs/



Login

Email

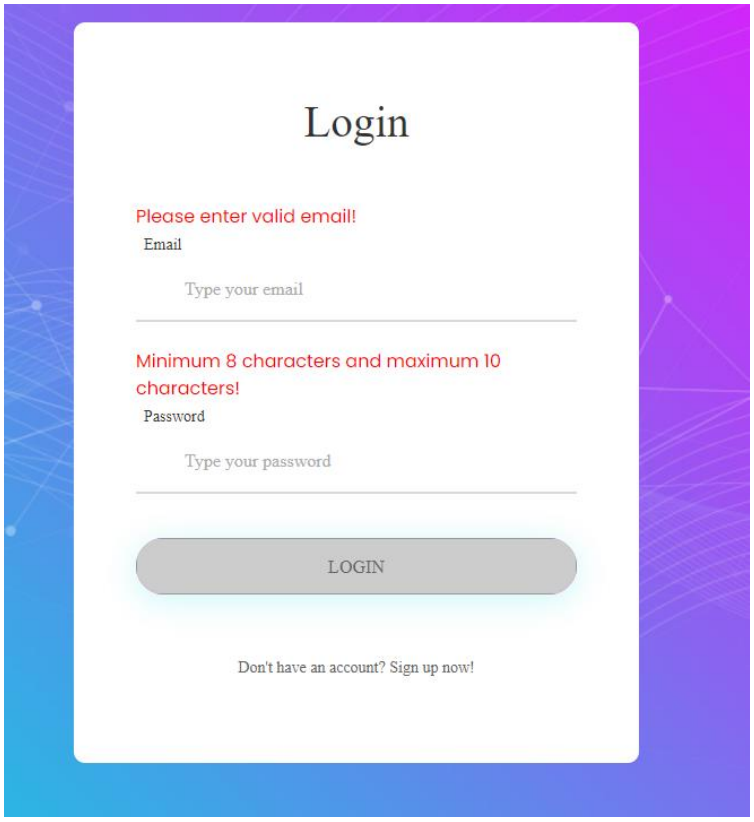
Type your email

Password

Type your password

LOGIN

Don't have an account? Sign up now!



Login

Please enter valid email!

Email

Type your email

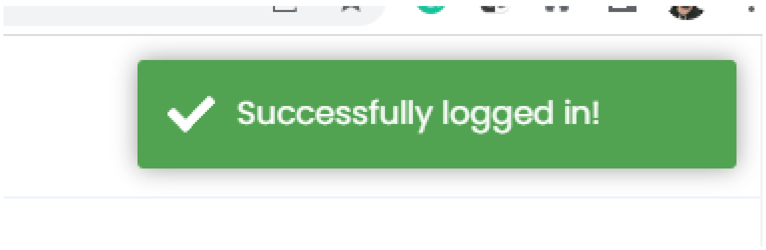
Minimum 8 characters and maximum 10 characters!

Password

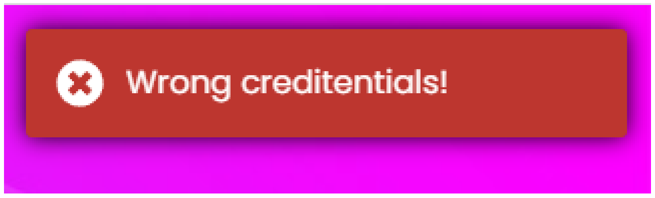
Type your password

LOGIN

Don't have an account? Sign up now!

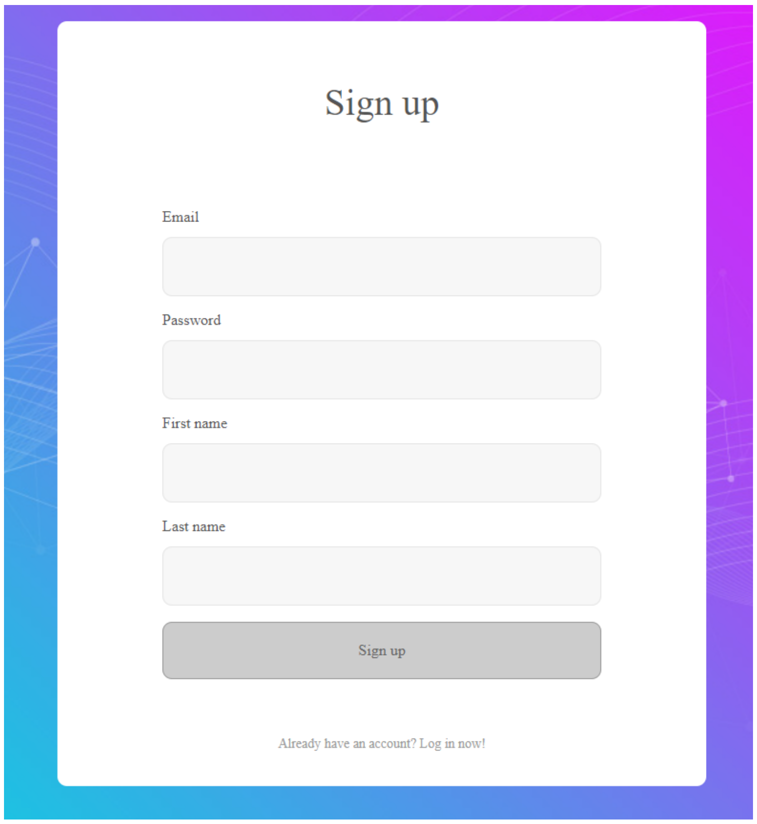


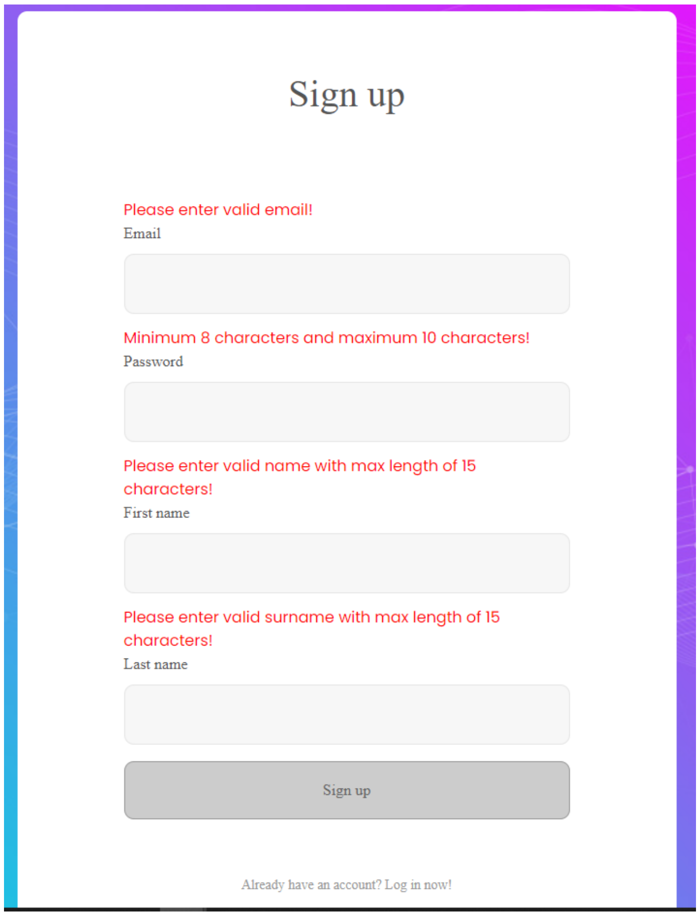
Successfully logged in!



Wrong creditentials!

Login feature with the respective Toastr messages which indicate if the process was successful or not





Sign up

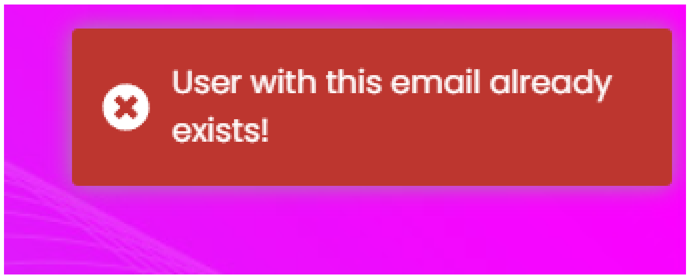
Please enter valid email!
Email

Minimum 8 characters and maximum 10 characters!
Password

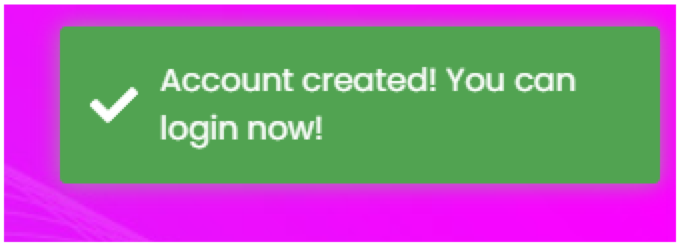
Please enter valid name with max length of 15 characters!
First name

Please enter valid surname with max length of 15 characters!
Last name

Already have an account? Log in now!

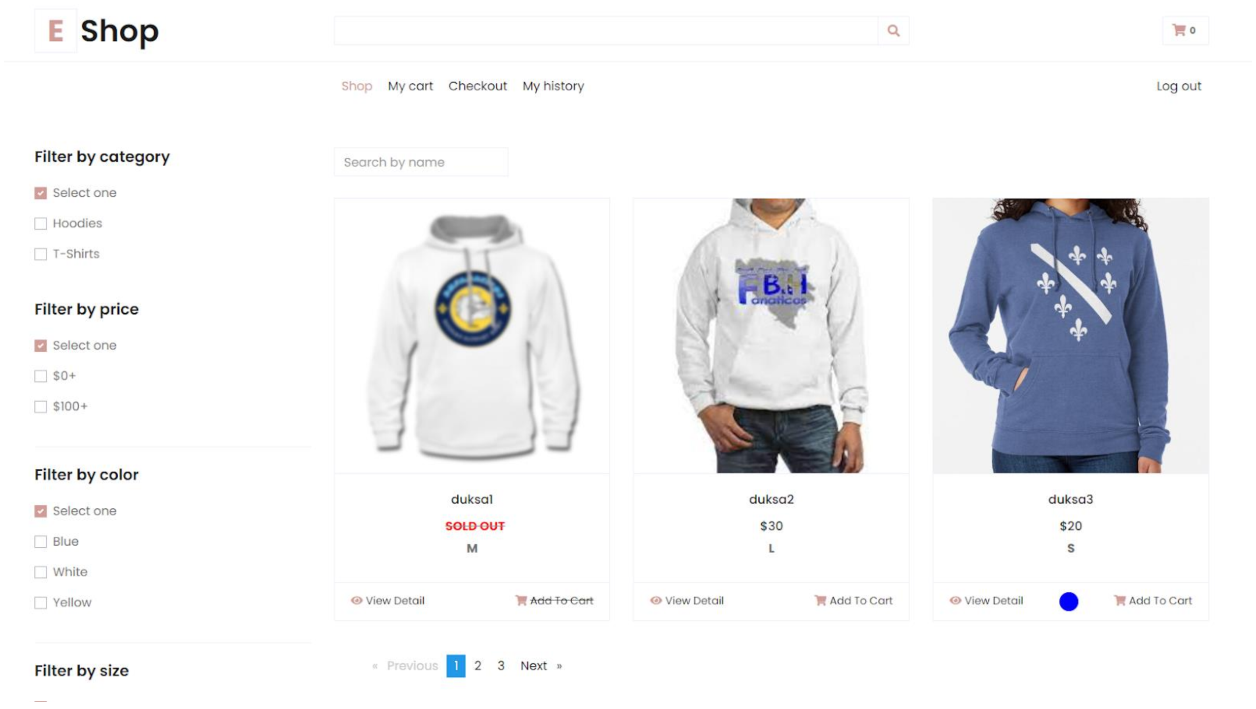


User with this email already exists!

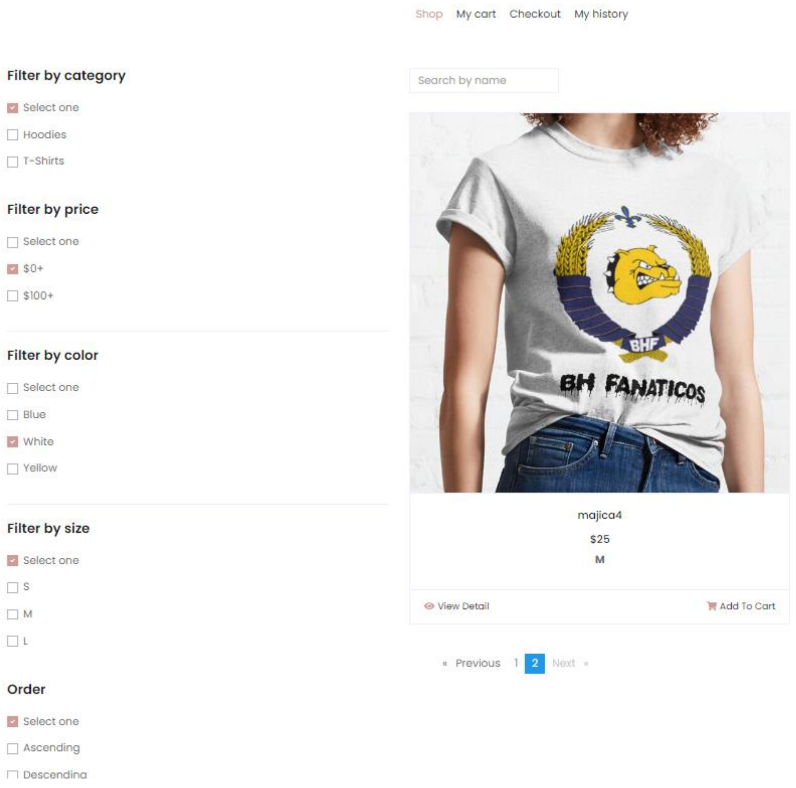


Account created! You can login now!

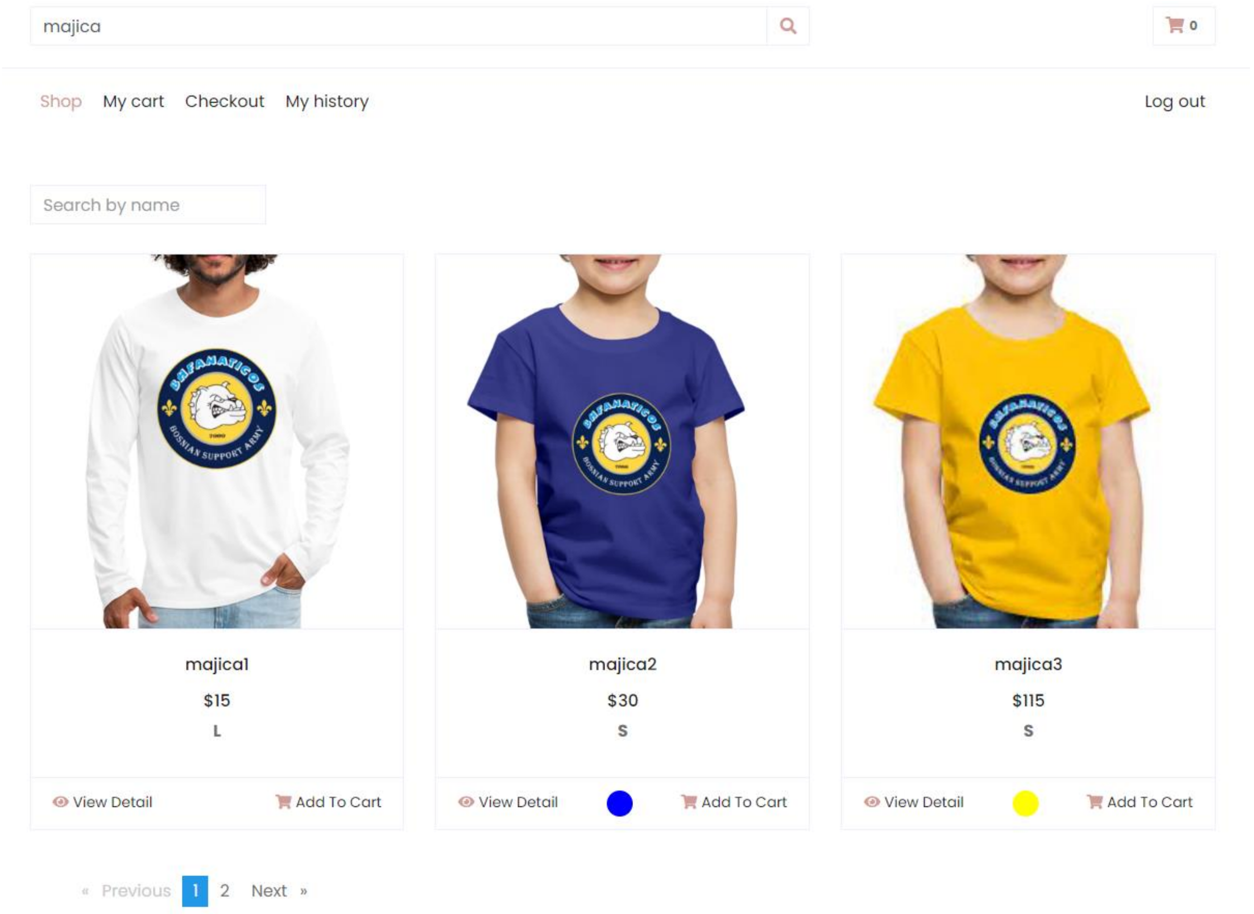
Register feature with the respective Toastr message which indicate if the process was successful or not.



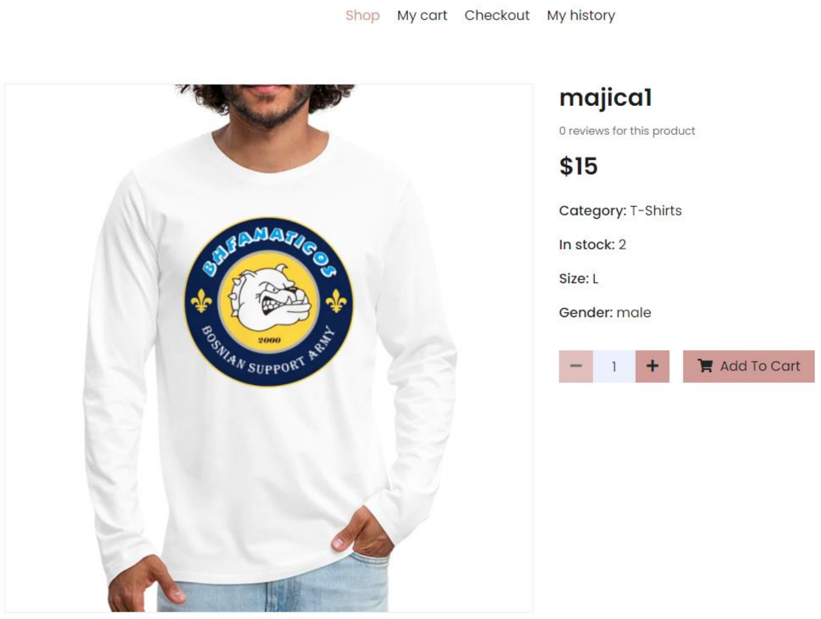
Display products with internal search and pagination

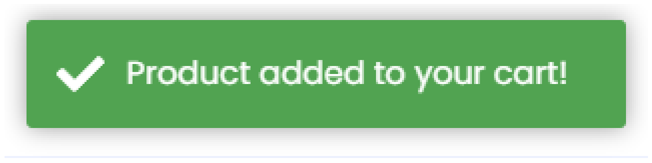


Filtering products by their size, price and etc.

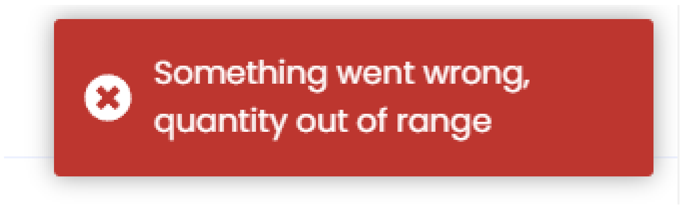


Search feature for products



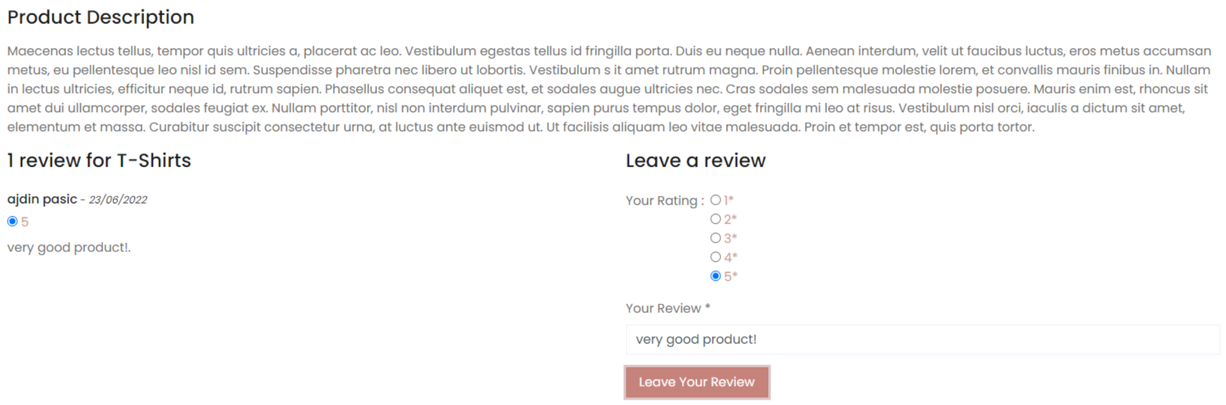


Product added to your cart!



Something went wrong, quantity out of range

Add to cart feature with its respective output messages



Product Description

Maecenas lectus tellus, tempor quis ultricies a, placerat ac leo. Vestibulum egestas tellus id fringilla porta. Duis eu neque nulla. Aenean interdum, velit ut faucibus luctus, eros metus accumsan metus, eu pellentesque leo nisl id sem. Suspendisse pharetra nec libero ut lobortis. Vestibulum s it amet rutrum magna. Proin pellentesque molestie lorem, et convallis mauris finibus in. Nullam in lectus ultricies, efficitur neque id, rutrum sapien. Phasellus consequat aliquet est, et sodales augue ultricies nec. Cras sodales sem malesuada molestie posuere. Mauris enim est, rhoncus sit amet dui ullamcorper, sodales feugiat ex. Nullam porttitor, nisl non interdum pulvinar, sapien purus tempus dolor, eget fringilla mi leo at risus. Vestibulum nisl orci, iaculis a dictum sit amet, elementum et massa. Curabitur suscipit consectetur urna, at luctus ante euismod ut. Ut facilisis aliquam leo vitae malesuada. Proin et tempor est, quis porta tortor.

1 review for T-Shirts

ajdin pasic - *23/06/2022*
● 5
very good product!.

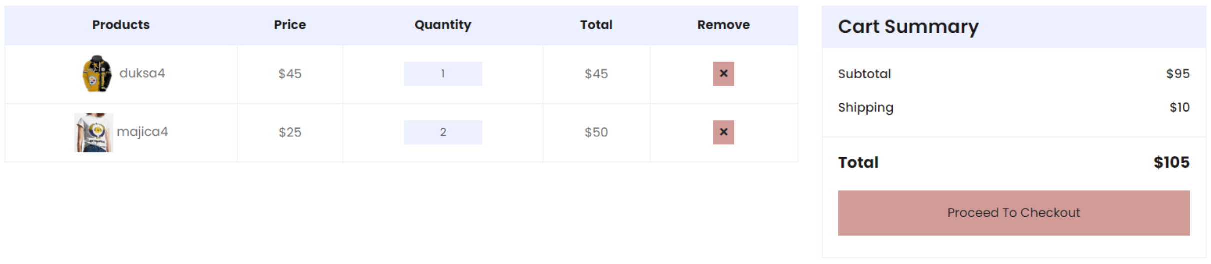
Leave a review

Your Rating : ○ 1* ○ 2* ○ 3* ○ 4* ● 5*

Your Review *

very good product!

Leave user feedback

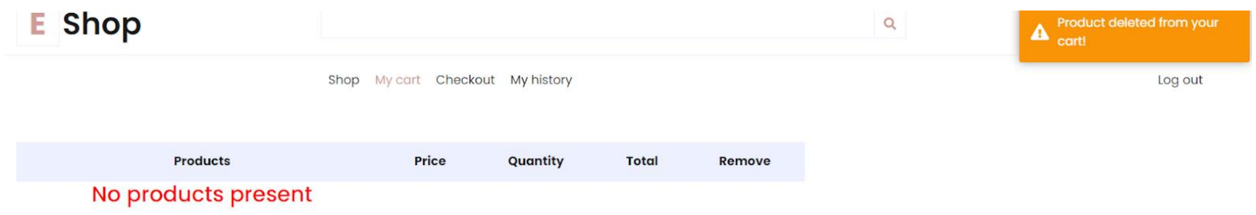


Products	Price	Quantity	Total	Remove
duksa4	$45	1	$45	×
majica4	$25	2	$50	×

Cart Summary

Subtotal	$95
Shipping	$10
Total	**$105**

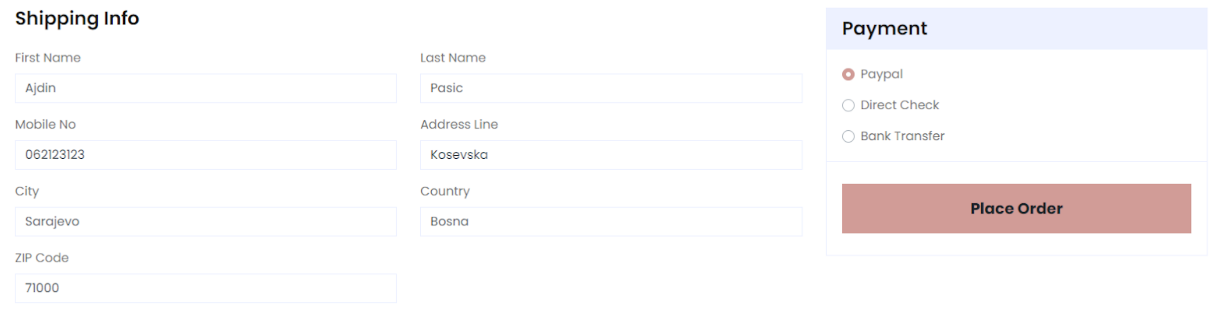
Displaying all ordered products with their total price



Product deleted from your cart!

Products	Price	Quantity	Total	Remove
No products present				

Feature which allows the user to remove products from cart



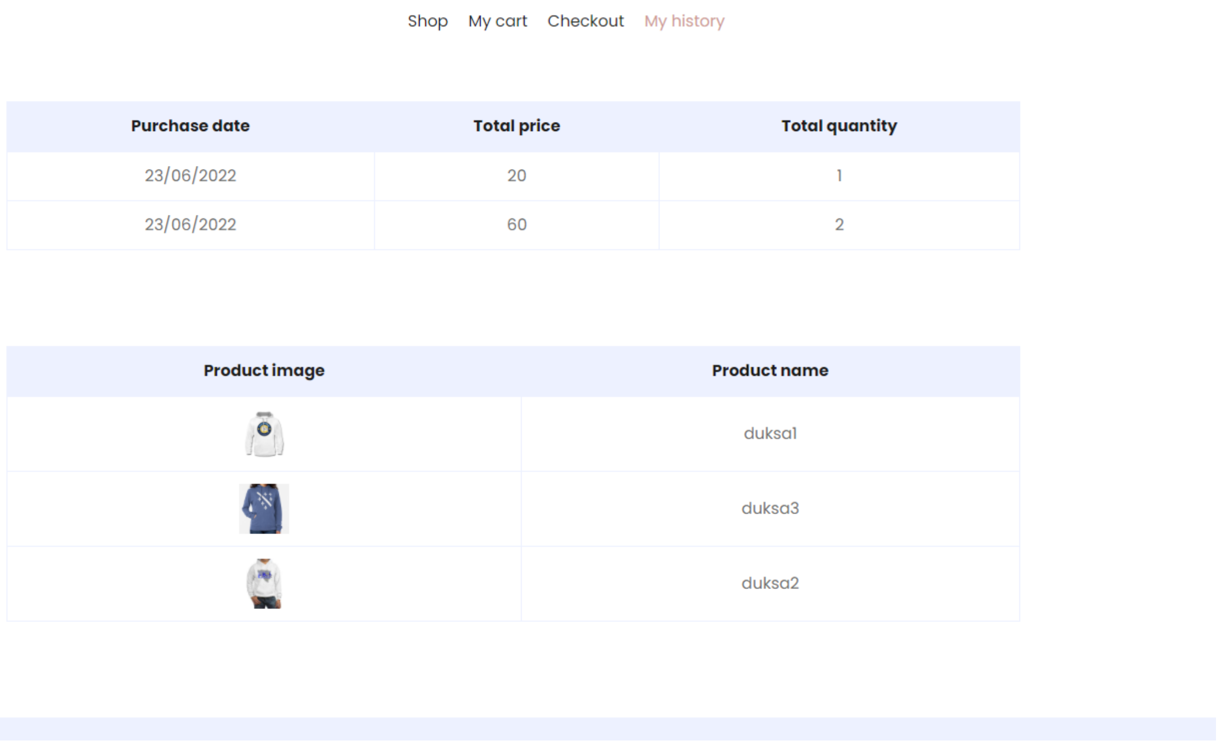
Shipping Info

First Name: Ajdin	Last Name: Pasic
Mobile No: 062123123	Address Line: Kosevska
City: Sarajevo	Country: Bosna
ZIP Code: 71000	

Payment

[✓ Paypal] Direct Check Bank Transfer

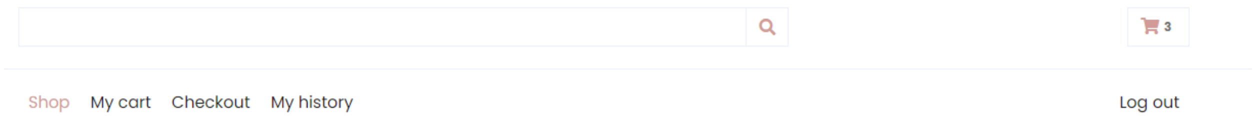
Final checkout



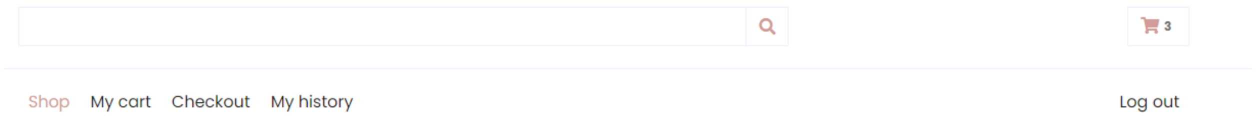
Purchase date	Total price	Total quantity
23/06/2022	20	1
23/06/2022	60	2

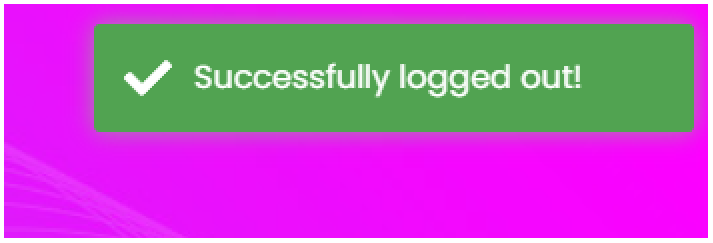
Product image	Product name
	duksa1
	duksa3
	duksa2

Display order and purchase history



Interactive UI which follows user's activities





Successfully logged out!

Logout feature with its respective message

2. Project Structure

2.1 Technologies

For the purpose of development of BH Fan Shop, we were prepared to use many technologies and frameworks which were needed for successful completion of the project.

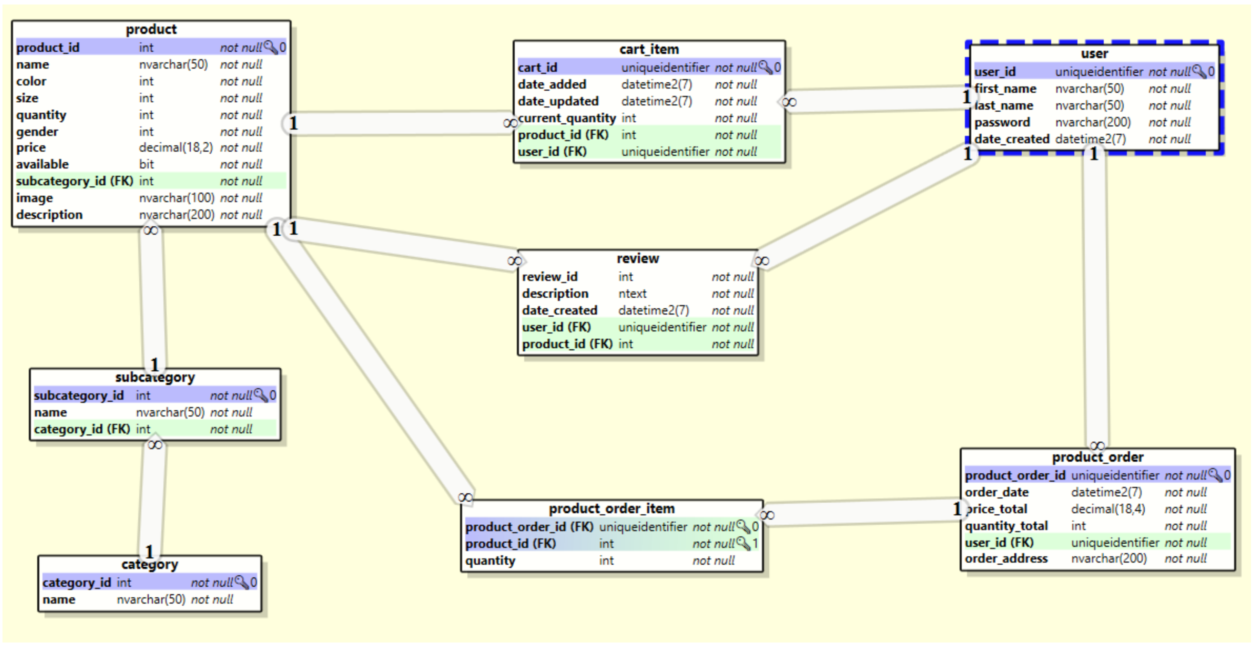
The frontend part was developed with the use of AngularJS and Bootstrap while the backend part of the application was fully developed with the use of Node.js. In the backend part, we used Swagger for routes that are connecting the backend with the frontend as well as for documentation and Composer for dependencies. We also used additional libraries like UUIDv4 for generation of random IDs for our users in the database and so on.

The database we used is a MySQL database which is hosted on the db4free.net platform.

When it comes to the coding standards, two standards were used (one for the backend and other one for the frontend parts of the BH Fan Shop). The standard to which the backend part fully complies is the Google JavaScript Style Guide (https://google.github.io/styleguide/jsguide.html) while the standard for the frontend is also Google's Style Guide, but dedicated for TypeScript (https://google.github.io/styleguide/tsguide.html).

When it comes to the implementation of these standards, we tended to respect these standards in the whole project, so the usage of the Google JavaScript Style Guide can be found in the files under the /api folder or in the "app.js" file while the usage of the Google TypeScript Style Guide can be found all across the /angular-spa folder.

2.2 Database Entities


product
product_id int not null 0
name nvarchar(50) not null
color int not null
size int not null
quantity int not null
gender int not null
price decimal(18,2) not null
available bit not null
subcategory_id (FK) int not null
image nvarchar(100) not null
description nvarchar(200) not null
cart_item
cart_id uniqueidentifier not null 0
date_added datetime2(7) not null
date_updated datetime2(7) not null
current_quantity int not null
product_id (FK) int not null
user_id (FK) uniqueidentifier not null
user
user_id uniqueidentifier not null 0
first_name nvarchar(50) not null
last_name nvarchar(50) not null
password nvarchar(200) not null
date_created datetime2(7) not null
review
review_id int not null
description ntext not null
date_created datetime2(7) not null
user_id (FK) uniqueidentifier not null
product_id (FK) int not null
subcategory
subcategory_id int not null 0
name nvarchar(50) not null
category_id (FK) int not null
category
category_id int not null 0
name nvarchar(50) not null
product_order_item
product_order_id (FK) uniqueidentifier not null 0
product_id (FK) int not null 1
quantity int not null
product_order
product_order_id uniqueidentifier not null 0
order_date datetime2(7) not null
price_total decimal(18,4) not null
quantity_total int not null
user_id (FK) uniqueidentifier not null
order_address nvarchar(200) not null
1
∞

ERD of the BH Fan Shop's MySQL database

The MySQL database of BH Fan Shop has eight tables: **product**, **subcategory**, **category**, **cart_item**, **review**, **product_order_item**, **user**, **product_order**. As it is visible in the database's ERD, there are many relations between the respective tables.

Entities for the '**product**' table are:

- **product_id;**

- **name**;

- **color**;

- **size**;

- **quantity**;

- **gender**;

- **available**;

- **image** –> link to the image of the product;

- **description**;

- **subcategory_id** -> ID of the subcategory where the product belongs to;

- **price**.

Entities for the '**subcategory'** table are:

- **subcategory_id**;

- **name**;

- **category_id** -> ID of the category to which the subcategory belongs to.

Entities for the '**category'** table are:

- **category_id**;

- **name**.

The next table is the '**cart_item'** table and this table is representing the items inside the cart. Entities for this table are:

- **cart_id**;

- **date_added** -> date when the item was added to the cart;

- **date_updated** -> initially null, but in case the user changes something in his cart, this column will contain the date and time when that item was modified.

- **current_quantity ->** quantity of all products inside the cart;

- **product_id** -> ID of the product which is in the user's cart;

- **user_id** -> ID of the user who is working with the current cart.

Entities for the '**review'** table are:

- **review_id**;

- **description**;

- **date_created**;

- **user_id ->** ID of the user who is leaving the review for the product;

- **product_id ->** ID of the product for which the review is being left.

Entities for the '**product_order_item**' table are:

- **product_order_item** -> ID of the order;

- **product_id** -> ID of the individual product being ordered;

- **quantity** -> quantity of the product being ordered.

Entities for the '**user'** table are:

- **user_id**;

- **first_name**;

- **last_name**;

- **password** -> hashed version of the password (hashed by an classic Java hash function)

- **date_created** -> date when the account was created;

- **email**;

- **token** – random token used to validate the user during login and logout. Becomes null after logout.

Entities for the '**product_order'** table:

- **product_order_id** -> ID of the finalized order;

- **order_date** -> date when the order was finalized;

- **price_total**;

- **quantity_total** -> quantity of all items in the cart;

- **user_id** -> ID of the user who is doing the order;

- **order_address** -> address of the user who is doing the order.

2.3 Architectural Pattern

When it comes to the architectural pattern of BH Fan Shop, we are sure that it's pattern is very rare among other projects. The architectural pattern that was used is event-driven architecture. The reason why we used this architecture is because it has been proven that TypeScript / JavaScript applications get the best performance with this approach. Also, we would like to mention that in the first edition, the application was tended to have n-tier architecture structure, but then we realized that the performance dropped and did not produce the needed results (most probably due to the asynchronous nature of Node.js in which the backend was written in).

The event-driven architecture has three participants: event producer, event router and event consumer.

The event producer role is related to all activities which the user does on the UI. This means that everything that the user does, starting from login to adding products to cart is actually a role of an event producer. All methods which take and track user input participate in this role.

The event router role is related to routing functions that serve as a bridge between the frontend and the backend. The functions which were written in the classes that can be found in /angular-spa/src/app/services/ serve as routers which call the routes that invoke the queries that need to be performed in the API.

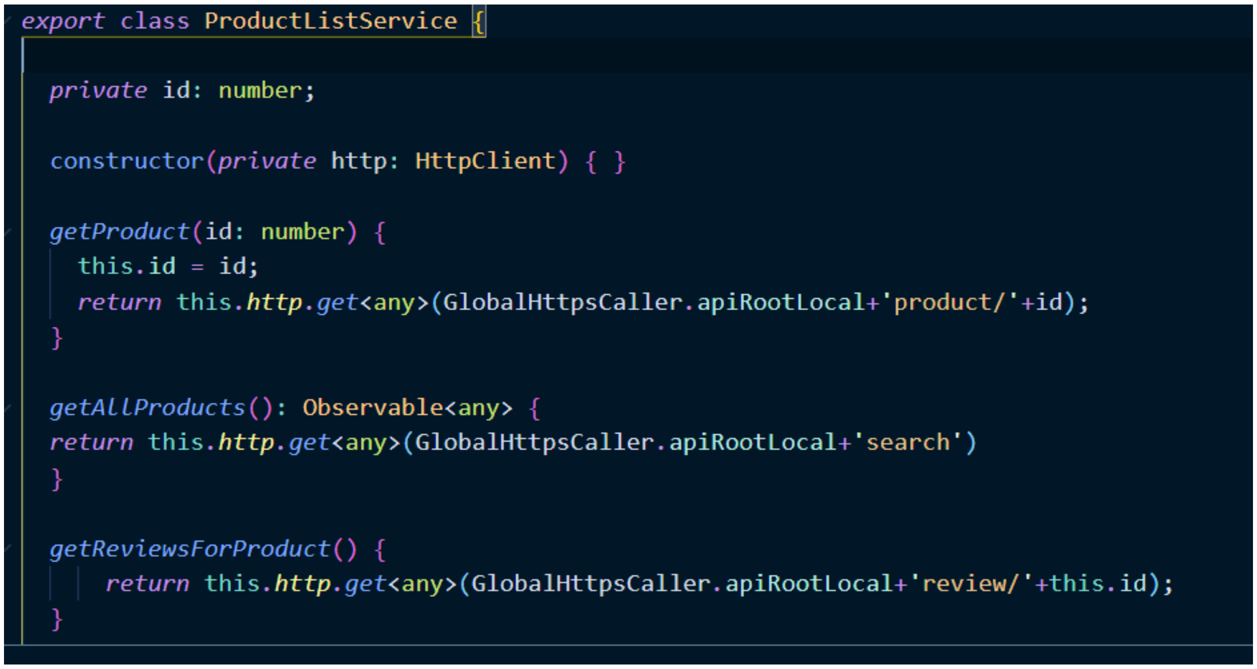
The event consumer role is related to the backend part which can be found in app.js, which contains all necessary routes and queries for the application. After the user performs some activity on the UI, the results of that activitiy are sent through the routers to the API, which invokes activity on the database (editing quantity, adding new items, generation of a token during login etc).

2.4 Design Patterns

Due to the specific nature of the technologies which were used for development of BH Fan Shop, many patterns were used. Some of the patterns which were used are:

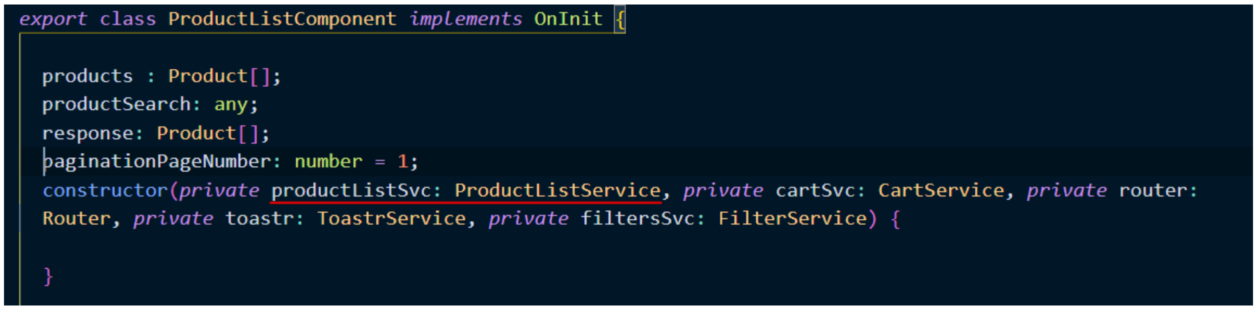
- Dependency Injection

Can be found in (angular-spa/src/app/services/product-list.services.ts), for example. This is a pattern in which one object of a class receives object of another class that depends on leading to separation of concerns of constructing objects and loosely coupling.

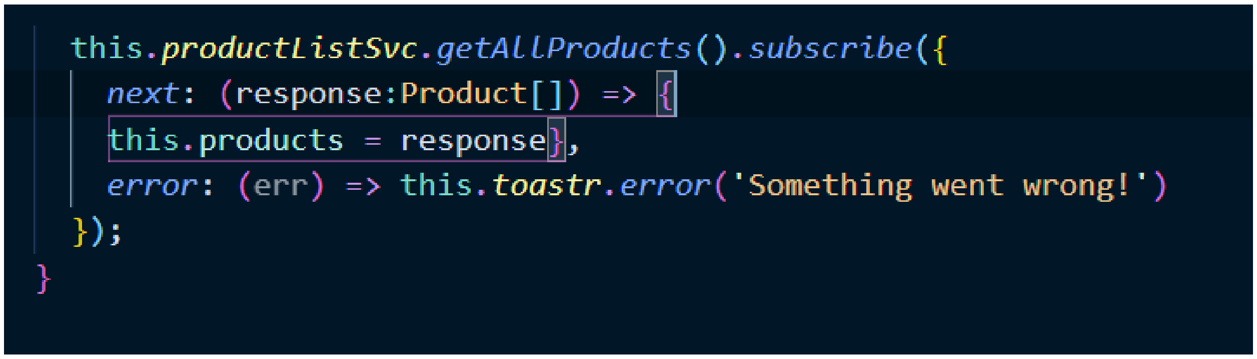


```
export class ProductListService {

  private id: number;

  constructor(private http: HttpClient) { }

  getProduct(id: number) {
    this.id = id;
    return this.http.get<any>(GlobalHttpsCaller.apiRootLocal+'product/'+id);
  }

  getAllProducts(): Observable<any> {
  return this.http.get<any>(GlobalHttpsCaller.apiRootLocal+'search')
  }

  getReviewsForProduct() {
      return this.http.get<any>(GlobalHttpsCaller.apiRootLocal+'review/'+this.id);
  }
```

On the picture, we can see that he have a ProductListService class, so in order to use its methods, we inject an object of this class into another TypeScript component during its initialization.



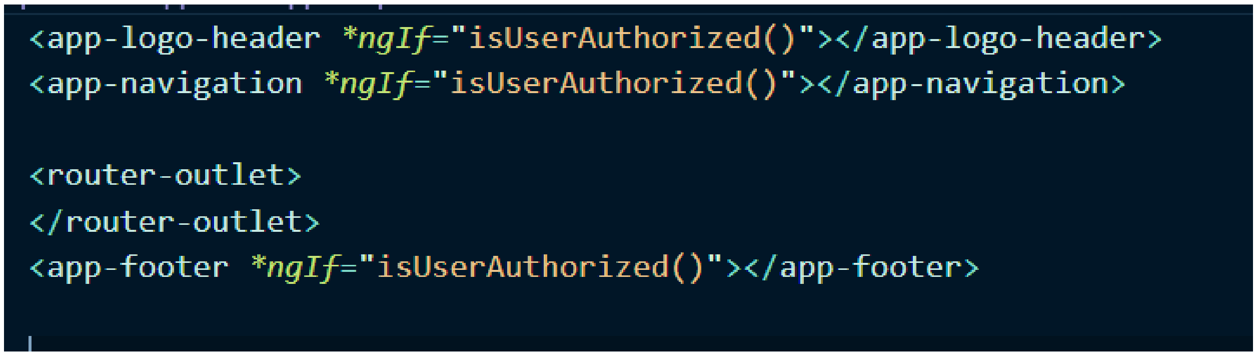
```
export class ProductListComponent implements OnInit {

  products : Product[];
  productSearch: any;
  response: Product[];
  paginationPageNumber: number = 1;
  constructor(private productListSvc: ProductListService, private cartSvc: CartService, private router:
  Router, private toastr: ToastrService, private filtersSvc: FilterService) {

  }
```



```
  this.productListSvc.getAllProducts().subscribe({
    next: (response:Product[]) => {
    this.products = response},
    error: (err) => this.toastr.error('Something went wrong!')
  });
}
```

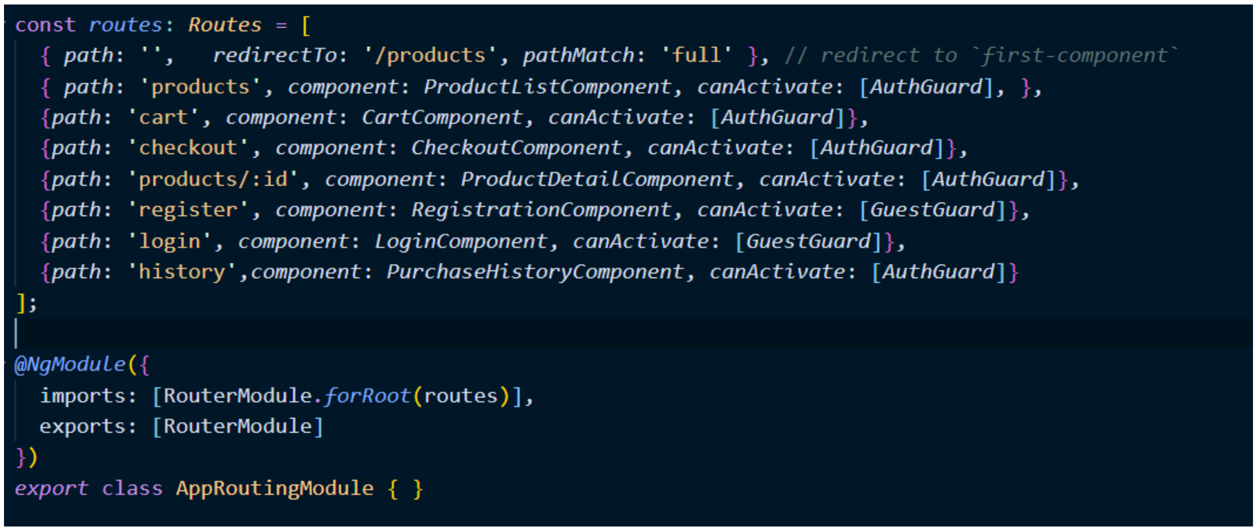
- Lazy Pattern

Lazy pattern examples can be found in many places in our project, for example (angular-spa/src/app/app.component.html, line 1 and line 2). It is used to load components on the application's need. Depending on the current router link that has been hit, browser loads only requested template and injects necessary content. It results in better performance and usability.



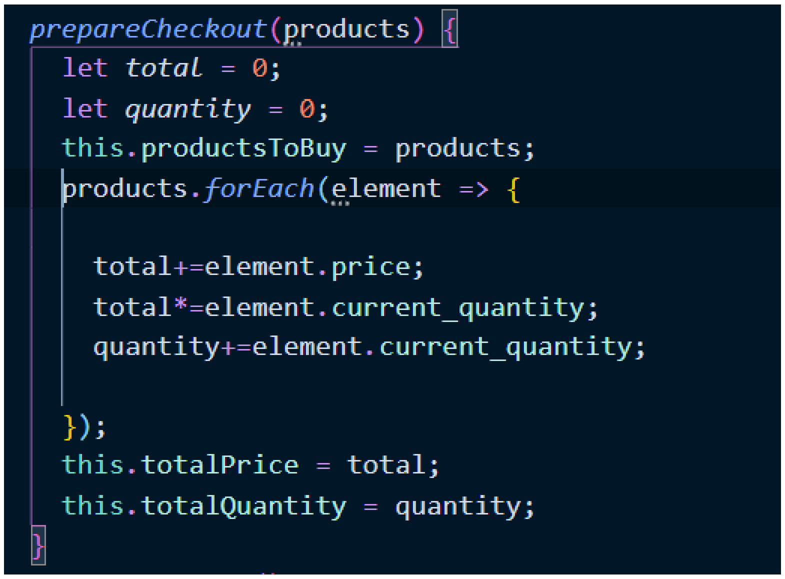
```
<app-logo-header *ngIf="isUserAuthorized()"></app-logo-header>
<app-navigation *ngIf="isUserAuthorized()"></app-navigation>

<router-outlet>
</router-outlet>
<app-footer *ngIf="isUserAuthorized()"></app-footer>
```

In this example, app.component.html holds tags of other components. Header, navigation and footer will always be loaded but content inside of them will be determined by the route which has hit in router-outlet.

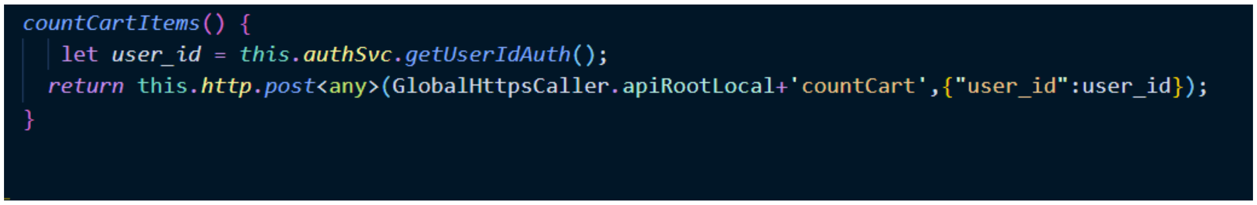

```
const routes: Routes = [
  { path: '',   redirectTo: '/products', pathMatch: 'full' }, // redirect to `first-component`
  { path: 'products', component: ProductListComponent, canActivate: [AuthGuard], },
  {path: 'cart', component: CartComponent, canActivate: [AuthGuard]},
  {path: 'checkout', component: CheckoutComponent, canActivate: [AuthGuard]},
  {path: 'products/:id', component: ProductDetailComponent, canActivate: [AuthGuard]},
  {path: 'register', component: RegistrationComponent, canActivate: [GuestGuard]},
  {path: 'login', component: LoginComponent, canActivate: [GuestGuard]},
  {path: 'history',component: PurchaseHistoryComponent, canActivate: [AuthGuard]}
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

- Solid Principles

SOLID is an acronym that stands for five key design principles: **single responsibility principle, open-closed principle, Liskov substitution principle, interface segregation principle, and dependency inversion principle**. All five are commonly used by software engineers and provide some important benefits for developers.

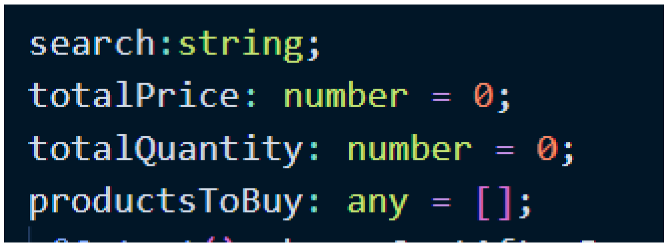

```
prepareCheckout(products) {
  let total = 0;
  let quantity = 0;
  this.productsToBuy = products;
  products.forEach(element => {

    total+=element.price;
    total*=element.current_quantity;
    quantity+=element.current_quantity;

  });
  this.totalPrice = total;
  this.totalQuantity = quantity;
}
```

In the example above, we have a function that has single-responsibility to calculate total price and quantity for checkout (angular-spa/src/app/services/logoservices.ts, line 91)



```
countCartItems() {
    let user_id = this.authSvc.getUserIdAuth();
  return this.http.post<any>(GlobalHttpsCaller.apiRootLocal+'countCart',{"user_id":user_id});
}
```

So, by using the solid principles, instead of duplicating the URL for the domain, we use static constant variables from a single file. (angular-spa/src/app/services/logoservices.ts, line 105)



```
search:string;
totalPrice: number = 0;
totalQuantity: number = 0;
productsToBuy: any = [];
```

Variables that are strictly defined (angular-spa/src/app/services/logoservices.ts, line 13).

- Singleton

This pattern was used in api/models/CoreModel.js during establishment of a connection with the database. Due to the fact that the database was hosted on a free platform and we could not count on good performance of the database, we wanted to have only one connection per user. With the structure of the class, the connection will be created only once and will work until the user leaves the platform. This was also proven when we tested the backend part of the application, where we executed multiple test queries on one connection.

3. Conclusion

BH Fan Shop turned out to be a good but also a difficult product for development, because there were many patterns that had to be followed and there were many components that had to be connected in order for the application to work. Also, we realized that after checking our code, the majority of our code respected the Google JavaScript and TypeScript guides so no major job in refactoring had to be done.

However, the application is open for improvements and we left space for future changes which will occur (additional features, new models etc). Also, JavaScript has no limitations when it comes to creativity so most probably there are possible improvements in order of using other functions or other methods which can improve the functioning of the application. The most interesting part is that for one feature (hashing the password) we used a method whose logic was defined more than thirty years ago, so that is a sign that behind BH Fan Shop stands a lot of work and exploration.

There were difficult parts and segments during the development of BH Fan Shop and the most major flaw is the database. The database is hosted on a free platform so no great performance can be expected and we experienced various bugs and database shutdowns during development. However, after the development was concluded, the database, surprisingly, works properly. Other segments like deployment and coding were also producing some issues, but for deployment, we solved the issues by hosting backend and frontend part on different platforms (Vercel for frontend and Heroku for backend).

BH Fan Shop produced many stress during development, but again we are happy with the final product.