

Ecommerce Full-stack Website Description

NEST Interior design

Main technologies:

- 1) Angular 8
- 2) RXJS
- 3) Angular Material
- 4) Bootstrap 4.4.1
- 5) .Net Core 2.2
- 6) .Net Entity Framework Core 2.2
- 7) SQL Server DB (Code First)

Instructions for Running the application

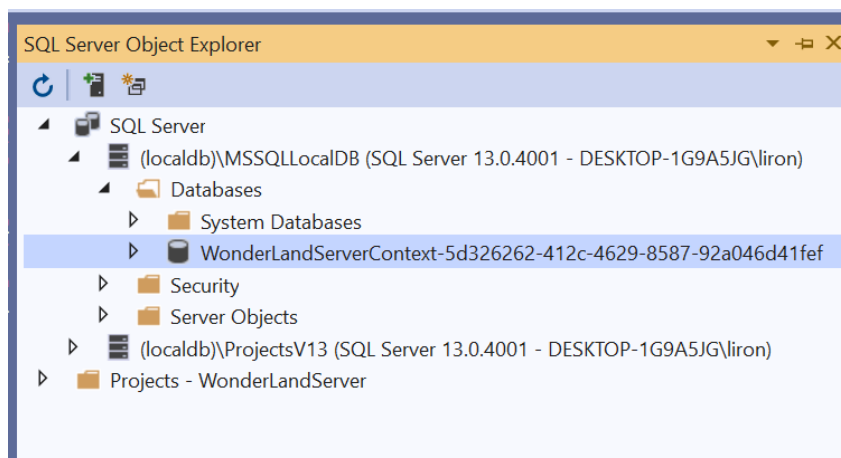
The .NET Project is in the folder called “Wonderland Server” which includes the backend and also the clients side angular app in it too, the Client side is in this path:

“WonderLandServer\WonderLandServer\Wonderland”

I used Visual Studio Code and Visual Studio 2019 in building the project.

- Before proceeding please install node modules in this folder in the path above: “npm i”
- Open the .NET solution in the main folder “WonderLandServer” then build the solution, if some errors are presented please ignore and build the solution again.
- Then open Package Manager Console and type “Update-Database” to build the database according to migrations.

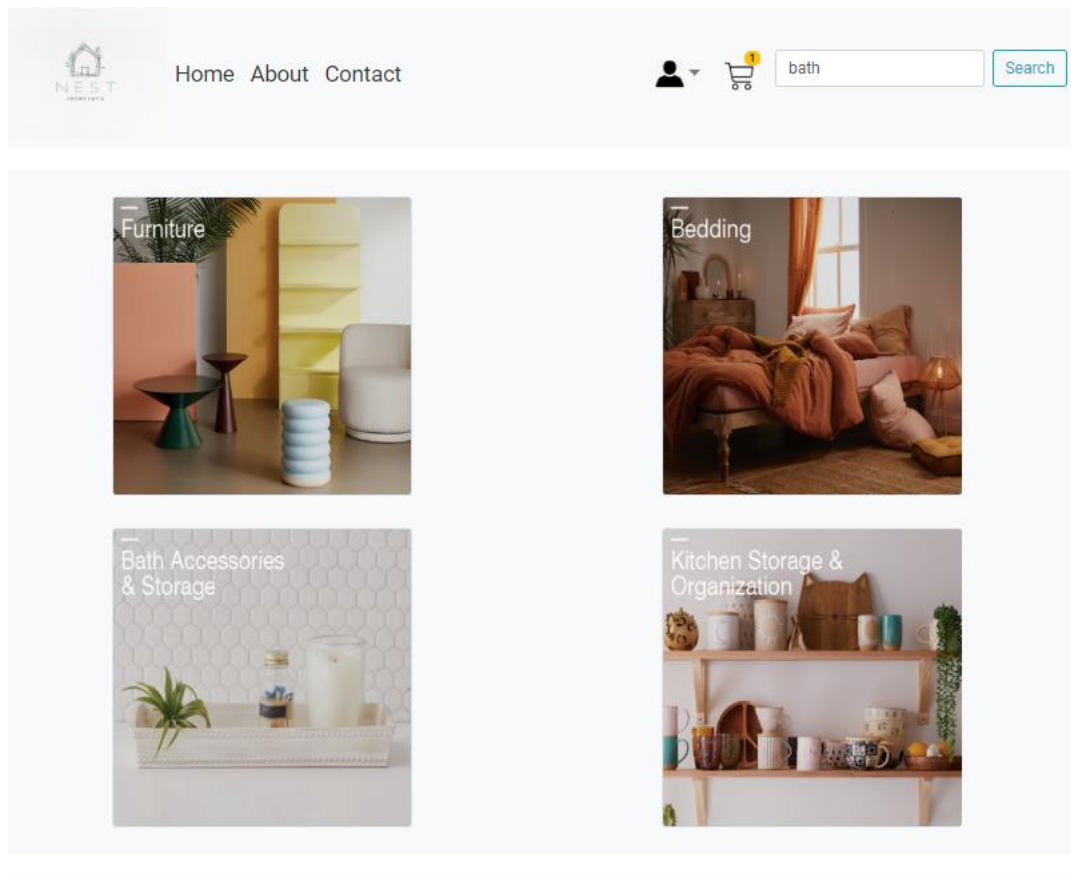
Then to see the Database please open “SQL Server Object Explorer” from Visual Studio 2019 and the database name “WonderLandServerContext...” will be there:



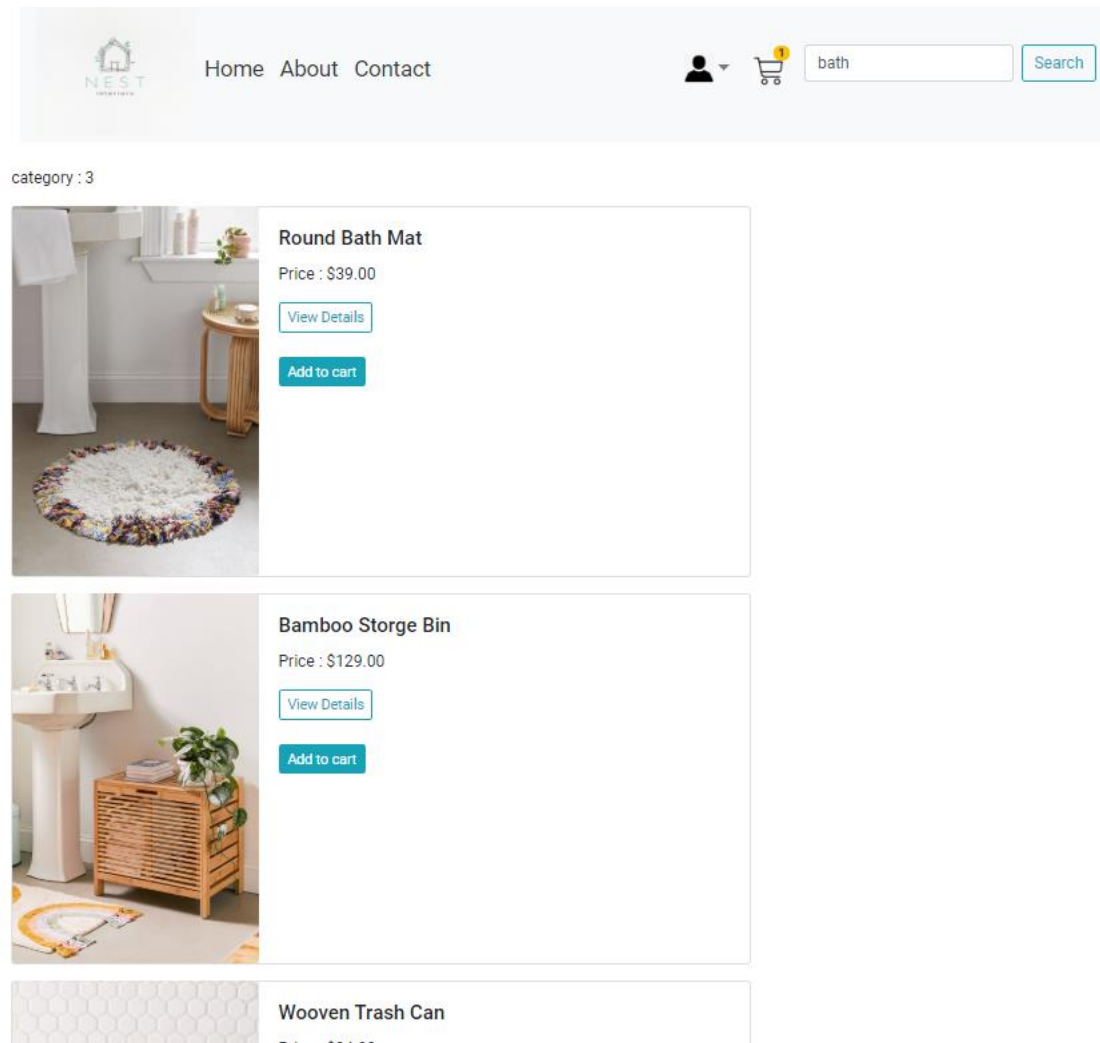
- Now, after building the database in order to run the application we can run it from visual studio 2019, “F5” and this will run the server and the Angular client application both on the same port. You can also still run the client side by angular CLI command line in this path
“WonderLandServer\WonderLandServer\Wonderland”: “ng serve” but the server in .NET have to be running first.

General Description of the website:

The first page that opens is app component which includes the navigation bar saved in header component and the home page saved in home component:



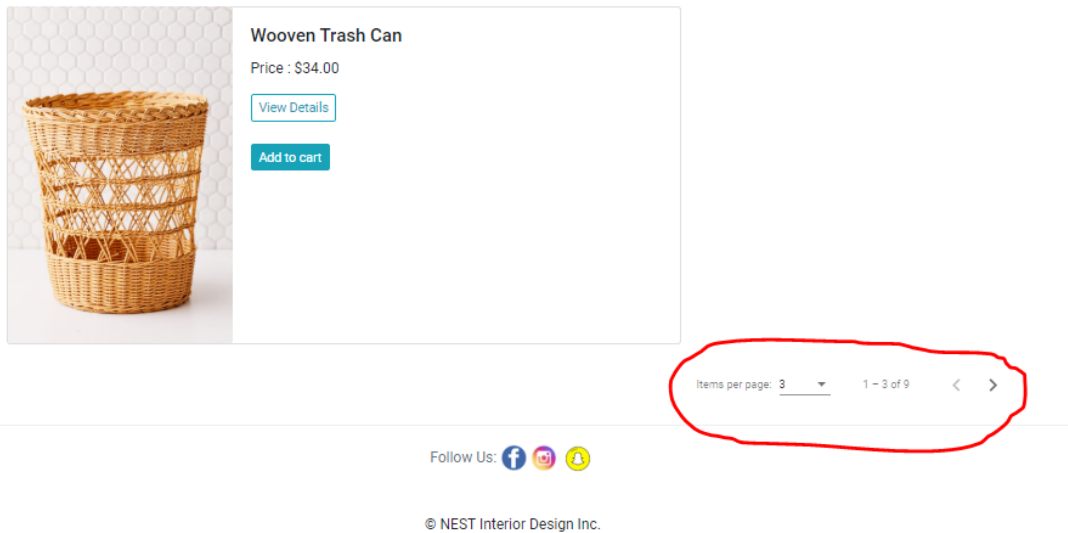
There are 4 categories of products in the home page. Choose any category and you will be routed to the products page to get all the products of the chosen category.



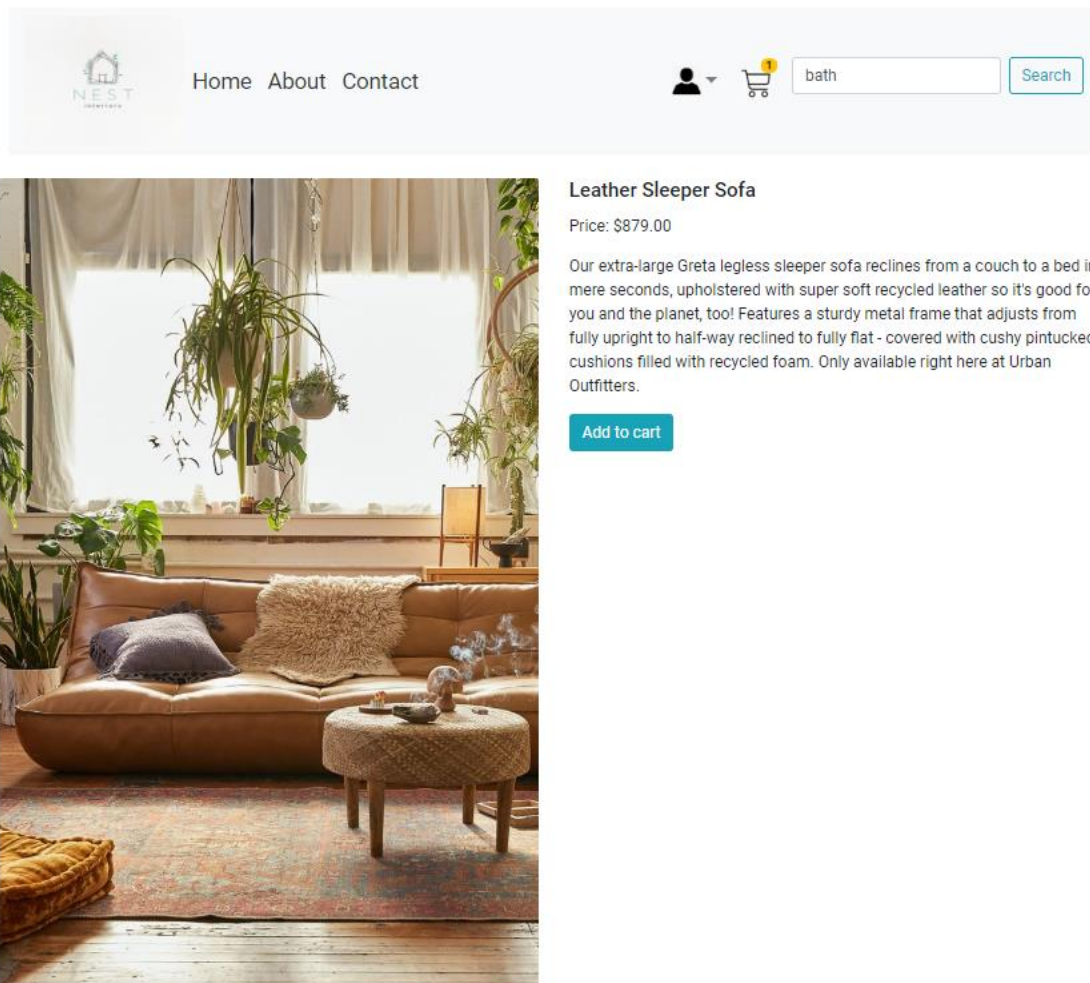
You can add to cart any product once from here and you will see the number in the cart icon in the navigation bar updating accordingly.

note: you can add the product only once, if you press "Add to cart" again, you will get an alert that the item is already in cart. In order to add the quantity of the product you need to do it later when you go to the cart.











Additionally, in the bottom of this page there is a paginator to scroll between pages Which I built using Angular Material. You can scroll between pages here and also change the number of products presented in any given page:



Also, there is an option to press on the photo of any product, or the “View Details” button and get to the product details page saved in product-details component. You can add to cart also from here:



Now! ... after adding some products, you press on the cart icon in the navigation bar above and get to the cart page component:

<div> Home About Contact</div> <div>  <input type="text" value="Search"/> <input type="button" value="Search"/></div>			
	Title	Price	
	Rubberwood Sofa	\$699.00	<div>+ 1 -</div>
	Throw Blanket	\$49.00	<div>+ 1 -</div>
	Dot Duvet Cover	\$258.00	<div>+ 2 -</div>
	Storage Cabinet	\$699.00	<div>+ 1 -</div>
		\$1,705.00	
<input type="button" value="Payment"/>			
Follow Us:   			
© NEST Interior Design Inc.			

Here you will see all the products you added, here you can add quantity as well. Then we can press the “Send Order” button to get to the check-out page component BUT if the user is still anonymous he will be redirected to the account page component in order to first finish Logging in:

localhost:44310/account?redirectUrl=%2Fcheck-out

Home About Contact

Sign In
Registered Customers
Email address
Password
Forgot Password?
Secured Sign In

Create an Account
Register to enjoy personalized services
Create an Account

Follow Us:

© NEST Interior Design Inc.

For now, you can Sign In using a default User which is already saved in my application:

Email: admin@admin.com

Password: admin

Once you sign in successfully you will be redirected to the page from where you came from which is saved in the URL in a query parameter called “redirectUrl” as presented in the image above with red underline, in this case it’s check-out page:

Home About Contact

Admin Search

Order Summary:	
Rubberwood Sofa :	\$699.00
Marta Ottoman :	\$299.00
Patchwork Quilt :	\$129.00
Throw Blanket :	\$49.00
Total :	\$1,176.00

Email Full Name

Address 1234 Main St

Address 2 Apartment, studio, or floor

City State Zip

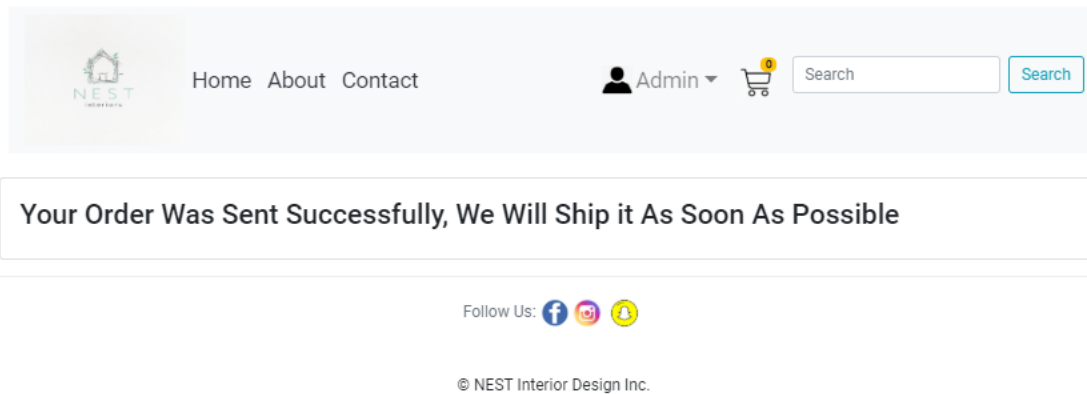
Choose...

☐ Agree to terms & Conditions

Payment

In the checkout-page you will see the form on the left where you need to fill in costumer details and address, it’s built as a reactive-form in Angular with validations, the button is disabled until the form is in valid state. On the right side you will find an order summary with all the products you added, their prices and the total price.

Once you fill the form and submit it successfully by pressing the “Send Order” Button you will be redirected to the Order-Success page to inform the user that the order was sent successfully (if there was any problem in contacting the serve/database we will see a popup alert registering an error):



The cart will get emptied upon finishing the order successfully.

Now if you check the database in the “SQL Server Object Explorer” in Visual Studio, and then check the Orders table you will find that the order was registered:

dbo.Orders [Data]									
Max Rows: 1000									
	OrderId	email	name	address	address2	city	state	zip	gridCheck
1	1	liron@liron.com	liron	1234 telaviv, 12	12d	Tel Aviv	Israel	521	True
2	2	liron@liron.com	liron	1234 telaviv, 12...	12d	Tel Aviv	Israel	521	False
3	3	liron@liron.com	liron	1234 telaviv, 12...	12d	Tel Aviv	Israel	521	False
4	4	liron@liron.com	liron	1234 telaviv, 12...	12d	Tel Aviv	Israel	521	False
5	5	liron@liron.com	liron	1234 telaviv, 12...	12d	Tel Aviv	Israel	521	False
6	6	liron@liron.com	liron	1234 telaviv, 12...	12d	Tel Aviv	Israel	521	False
7	7	liron@liron.com	liron	1234 telaviv, 12...	12d	Tel Aviv	Israel	521	False
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

With a new automatically generated id number and the rest of the address details.

now in order to see the details of the products purchased in each order, check the Order-Items table in the database and look for the order-item that has the same Order Id number you are looking for:

dbo.OrderItems [Data]				
Max Rows: 1000				
	OrderItemId	Quantity	OrderId	ProductId
1	1	1	1	36
2	2	1	1	34
3	3	1	1	40
4	4	1	1	21
5	5	1	2	36
6	6	1	2	35
7	7	1	3	41
8	8	1	3	36
9	9	1	4	41
10	10	1	4	36

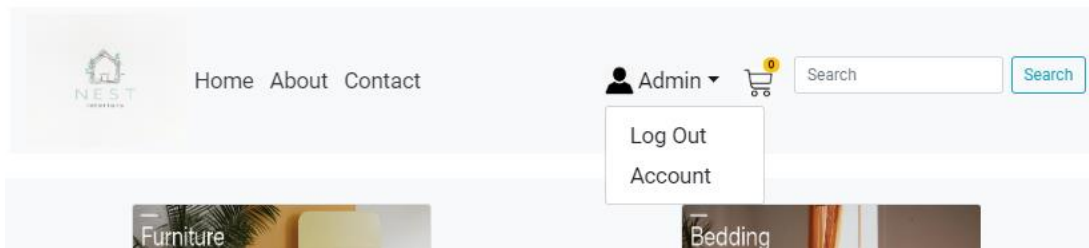
Each order-item in this table is connected to the same Order Id number, and also each one of them has a product Id to point to a specific product in the Products table in the database and also has another column for the Quantity of this specific product.

Note the products table in the database has been already seeded as we ran the application on the first time automatically with a list of products:

dbo.Products [Data] X						
dbo.OrderItems [Data]						
dbo.Orders [Data]						
Max Rows: 1000						
	id	category	img	title	description	price
▶	1	1	Images/furnitur...	Wood Coffee T...	Lacquered acaci...	159
	2	2	Images/beddin...	Floral Knot Set	With a vintage-i...	129
	3	3	Images/bath-to...	Toothbrush Hol...	Prop up your t...	9
	4	3	Images/bath-h...	Rolling Hamper	Store blankets ...	59
	5	3	Images/bath-st...	Bathroom Stora...	Organize towel...	219
	6	3	Images/bath-b...	Basket Set	Perfect for stori...	59
	7	3	Images/bath-or...	Make-Up Orga...	Showcase your ...	39
	8	3	Images/bath-d...	Over The Door ...	Over-the-door ...	69
	9	3	Images/bath-tr...	Wooven Trash ...	Basket trash ca...	34
	10	3	Images/bath-St...	Bamboo Storge...	Store spare line...	129
	11	3	Images/bath-m...	Round Bath Mat	Bring a whimsic...	39
	12	4	Images/kitchen...	Hanging Basket	Wood + cotton...	39
	13	4	Images/kitchen...	Cutting Board	These unique it...	34
	14	4	Images/kitchen...	Utensil Set	Coordinate you...	36
	15	4	Images/kitchen...	Salt & Pepper S...	Salt and peppe...	20
	16	4	Images/kitchen...	Tea Infuser & C...	Bring a boho fe...	16
	17	4	Images/kitchen...	Folding Dish Ra...	No need to be ...	34
	18	4	Images/kitchen...	Bamboo Jar Set	Set of four glas...	37
	19	2	Images/beddin...	Floral Cotton Set	Inspired by cou...	139

I used “DBProductsInitializer” which is saved in the .Net project in the Models folder, this call is instantiated upon running the application “Program”.

Now! ... back to the website, notice that after Signing in you see the name of the User in the navigation bar, if you press on it we can either go to the Login page or Log out:



Let's log out for now and go to login page again (which can be reached also by pressing the account icon), and create a new account, press the "Create an Account" button and get to the "create-account" page component:

A screenshot of the 'create-account' page. The header is identical to the previous image, but the user profile icon is a generic person icon. Below the header is a registration form with the following fields: 'Username', 'Email', 'Password', and 'Confirm Password'. Each field has a corresponding input box. At the bottom left of the form is a 'Submit' button. Below the form, there is a 'Follow Us:' section with icons for Facebook, Instagram, and YouTube. At the very bottom, there is a copyright notice: '© NEST Interior Design Inc.'

This is a reactive-form with validations and custom validations, after submitting the form successfully you get a popup/Modal built using Angular Material stating the registration was successful and presenting the option to go to the "Login" page:

Username

Liron

Email

Liron@gmail.com

Password

.....

Confirm Password

.....

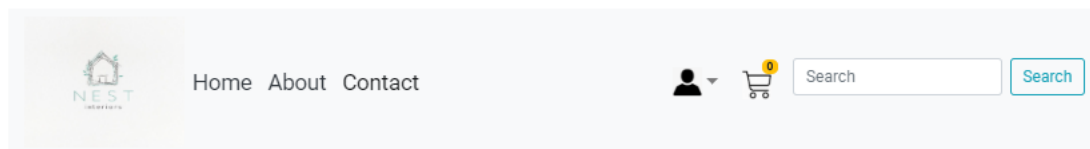
Submit

Hi Liron

You Registered Succesfully

Login Close

In the navigation bar you will find a link to the “Contact Us” page:



Contact Us

Email address

name@example.com

Your Name

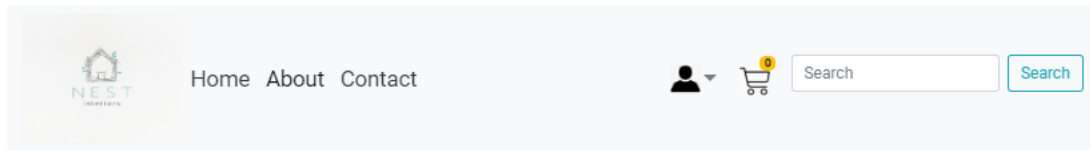
Your Message

Submit

This is a template driven form, when you submit it successfully the form will reset, and an alert will pop up, stating the newly generated id of this message as it is saved in the database in the “ContactMessages” Table:

dbo.ContactMessages [Data]				
	ContactMessa...	email	name	message
1		liron@liron.com	liron	This is my mess...
*	NULL	NULL	NULL	NULL

In the navigation bar we have a link for an “About” Page that gets us to this page:



Interior Your NEST...

NEST is a lifestyle retailer dedicated to inspiring customers through a unique combination of product, creativity and cultural understanding.

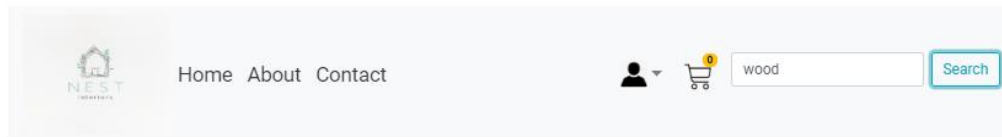
Your home should be your haven, the place where you can be you. That's why at NEST Interior we offer products, expertise, and inspiration to help you live your style.

"Our interiors are an insight into our brains.
It is a collaboration of design, art, humor, irony, functionality, and the street."
— Amanda Talbot, Rethink: The Way You Live

Follow Us:   

© NEST Interior Design Inc.

In addition, there is an option in the navigation bar, to search for products based on their title or their description, using the search box. This search will get the products component but with a filter based on the search words you typed:



Desk

Price : \$799.00

[View Details](#)

[Add to cart](#)



Rubberwood Sofa

Price : \$699.00

[View Details](#)

[Add to cart](#)



Floor Mirror

Price : \$199.00



Technical Description of Code:

Client Side:

Angular Services description:

1) Storage-Service:

```
5  @Injectable({
6    providedIn: 'root'
7  })
8  export class StorageService {
9
10     private userSubject = new BehaviorSubject(null) ;
11     userData = this.userSubject.asObservable() ;
12
13
14     private usersSubject = new BehaviorSubject([{
15       username: 'Admin',
16       email: 'admin@admin.com',
17       password: 'admin'
18     }]) ;
19     usersData = this.usersSubject.asObservable() ;
20
21     constructor() { }
22
23     updateUser(user: UserModel) {
24       this.userSubject.next(user) ;
25     }
26
27     updateUserList(usersList: UserModel[]) {
28       this.usersSubject.next(usersList) ;
29     }
30 }
```

This service manages the State of the users and the current logged-in User in my Angular Application, by using the “BehaviorSubject” in this service and then creating an observable out of it in the property “userData” and “usersData”, and then use it to access the list of users (like in account component) or access to the current logged in User like in the header-component we have to communicate with the Storage Service and subscribe to these observables. This Storage Service lives as long as the application is running, NOTICE! That if the Website was refreshed this Storage will be lost.

2) Cart Service:

```
8   export class CartService {
9
10      private cartData = new BehaviorSubject([]) ;
11      currentData = this.cartData.asObservable() ;
12
13      constructor() { }
14
15      update(newValue : Item[]) {
16          this.cartData.next(newValue) ;
17      }
18  }
```

This service manages the State of the cart and the products that are added to it, in a very similar way to the service above “Storage-Service”.

I used this service in cart component and in other components in the app to keep in sync with the cart items like the number of items in cart presented in the navigation bar on top near the cart icon in the header component, and also in these components: products, checkout, product-details.

3) Crud Service:

```
11  export class CrudService {
12
13      private usersList: UserModel[];
14
15      constructor(private storageService: StorageService) {
16          this.storageService.usersData.subscribe(users => this.usersList = users);
17      }
18
19      // Get the Current logged in user
20      getUser() : Observable<UserModel> {
21          return this.storageService.userData;
22      }
23
24      Login(user: LoggedUser) : boolean {
25          let userExist : boolean = false;
26          for (let u of this.usersList) {
27              if ((user.email === u.email)&&(user.password === u.password)) {
28                  userExist = true;
29                  var fullUser : UserModel = u ;
30              }
31          }
32
33          if (userExist) {
34              this.storageService.updateCurrentUser(fullUser) ;
35              return true ;
36          }
37          return false ;
38      }
39
40
41      Logout() {
42          this.storageService.updateCurrentUser(null) ;
43      }
```

The Crud is another service that manages the user interaction with the Storage Service, so as seen in the image above the function `getUser()` returns an Observable with the current logged in user, I subscribed to it in header component for example.

The function Login() gets a user object as a parameter and puts that user in the Storage Service as a current logged-in user.

The function Logout() removes the current logged-in user from the Storage Service and empties it.

The function addNewUser():

```
45   addNewUser(user: UserModel) {  
46     this.usersList.push(user);  
47     this.storageService.updateUsersList(this.usersList);  
48   }  
49 }
```

Takes a new User Object as an argument and saves it in the Users List function which includes all registered Users and it's in the Storage Service, please notice that users are not saved in the database yet so currently it's only saved as a state and this list will be removed upon exiting or refreshing the website.

4) Validation Service:

```
7   export class ValidationService {  
8     constructor() { }  
9   }  
10  
11  
12   export const passwordsMismatch: ValidatorFn = (control: FormGroup): ValidationErrors | null => {  
13     const password = control.get('password');  
14     const confirmPassword = control.get('confirmPassword');  
15  
16     return password && confirmPassword && password.value !== confirmPassword.value ? { 'passwordsMismatch': true } : null;  
17   };
```

This service is used to validate a password matching/mismatch field.

We used it in create-account component.

5) Products Service:

```
11 export class ProductsService {
12
13     constructor(private http : HttpClient) { }
14
15     getProducts() : Observable<Item[]> {
16         return this.http.get<Item[]>("https://localhost:44310/api/Products").pipe(
17             retry(1),
18             catchError(this.errorHandler)
19         )
20     }
21
22     getProductsById(id : number) {
23         return this.http.get<Item>("https://localhost:44310/api/Products/" + id).pipe(
24             retry(1),
25             catchError(this.errorHandler)
26         )
27     }
28
29     errorHandler(error) {
30         let errorMessage = '' ;
31         if (error.error instanceof ErrorEvent) {
32             // get client side error
33             errorMessage = error.error.message ;
34         } else {
35             // get server side error
36             errorMessage = `Error Code : ${error.status}\nMessage: ${error.message}` ;
37         }
38         return throwError(errorMessage) ;
39     }
40 }
```

This service is responsible for communicating with the server with REST Get Call using HTTP Client to pull from the database the list of products, and another function to get a specific product by its id, each one retries to reconnect again if it failed the first time that's why we have the "retry(1)". Both function make use of the function "errorHandler(error)" that logs information about the error and its code to the Console when a server error is detected. I used this service mainly in "products component"

6) Auth-Guard Service:

```
9 export class AuthGuardService implements CanActivate {
10
11   constructor(private auth: CrudService, private router: Router) { }
12
13   canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean | UrlTree | Observable<boolean | UrlTree> | Promise<boolean | UrlTree> {
14     var logged = false ;
15     this.auth.getUser().subscribe(user => { if (user) logged = true; }) ;
16     if (logged == false) this.router.navigate(['/account'], {queryParams: {redirectUrl: state.url} , queryParamsHandling: 'merge'})
17     return logged ;
18   }
19 }
```

This service implements the Angular interface “CanActivate”, it returns Boolean value, True if there is currently a logged-in user and false if not.

I used this service in app-routing.module in the routes settings of the paths that I wanted to protect from anonymous users, mainly in checkout component:

```
16 const routes: Routes = [
17   { path: '', component: HomeComponent },
18   { path: 'about', component: AboutComponent},
19   { path: 'contact', component: ContactComponent},
20   { path: 'account', component: AccountComponent},
21   { path: 'creat-account', component: CreateAccountComponent},
22   { path: 'products', component: ProductsComponent},
23   { path: 'products/:category', component: ProductsComponent},
24   { path: 'productsDetails/:id', component: ProductDetailsComponent},
25   { path: 'cart', component: CartComponent},
26   { path: 'check-out', component: CheckOutComponent, canActivate : [AuthGuardService]},
27   { path: 'order-success', component: OrderSuccessComponent, canActivate : [AuthGuardService]}
28 ];
```

If the “AuthGuardService” returns false, access to this route will be denied and if it returned true access to it will be permitted. Additionally, if access was denied the “AuthGuardService” service will redirect the client to the account/Login Page with a query parameter “redirectUrl” to save the place from which he was redirected from and that’s done in this line in the service:

```
if (logged == false) this.router.navigate(['/account'], {queryParams: {redirectUrl: state.url} , queryParamsHandling: 'merge'})
```


7) Checkout Service:

```
10 export class CheckoutService {
11
12   httpOptions = {
13     headers: new HttpHeaders({
14       'Content-Type': 'application/json; charset=utf-8'
15     })
16   };
17
18   constructor(private http : HttpClient) { }
19
20   sendOrderToDatabase(order) : Observable<any> {
21     return this.http.post<any>("https://localhost:44310/api/Orders", JSON.stringify(order), this.httpOptions ).pipe(
22       retry(1),
23       catchError(this.errorHandler)
24     );
25   }
26
27   errorHandler(error) {
28     let errorMessage = '' ;
29     if (error.error instanceof ErrorEvent) {
30       // get client side error
31       errorMessage = error.error.message ;
32     } else {
33       // get server side error
34       errorMessage = `Error Code : ${error.status}\nMessage: ${error.message}` ;
35     }
36     console.log(errorMessage);
37     return throwError(errorMessage) ;
38   }
39 }
40 }
```

This service uses REST call of POST to the server using HTTP Client, the function `sendOrderToDatabase(order)` takes an order object of this form:

```
var order = {
  OrderItems: OrderItem[ ],
  email: string
  name: string,
  address: string,
  address2: string,
  city: string,
  state: string,
  zip: number,
  gridCheck: Boolean
}
```

Every `OrderItem` object is of this form:

```
{
  ProductId: number,
  Quantity: number
}
```

Then it sends this Order object strigified as a JSON to the server in the body of the request to this header: "api/Orders" to be saved in the Orders Table and Order-Items table in the database. It also makes use of the `errorHandler()` function to log errors in the console.

I used this function in Checkout Component after the user adds products to the cart and submits his details.

8) Contact Service:

```
9  export class ContactService {
10
11    httpOptions = {
12      headers: new HttpHeaders({
13        'Content-Type': 'application/json; charset=utf-8'
14      })
15    };
16
17    constructor(private http : HttpClient) { }
18
19    contactMessageToDatabase(contactMessage) : Observable<any> {
20      return this.http.post<any>("https://localhost:44310/api/ContactMessages", JSON.stringify(contactMessage), this.httpOptions ).pipe(
21        retry(1),
22        catchError(this.errorHandler)
23      );
24    }
25
26    errorHandler(error) {
27      let errorMessage = '' ;
28      if (error.error instanceof ErrorEvent) {
29        // get client side error
30        errorMessage = error.error.message ;
31      } else {
32        // get server side error
33        errorMessage = `Error Code : ${error.status}\nMessage: ${error.message}` ;
34      }
35      console.log(errorMessage);
36      return throwError(errorMessage) ;
37    }
38  }
```

This service uses REST call of POST to the server using HTTP Client, the function `contactMessageToDatabase(contactMessage)` takes a `contactMessage` object which is exactly the same as in the Contact Us form that is in the ContactComponent Page, and sends it to the server to be saved in the Database in ContactMessages table.

General Notes:

* Throughout the Angular application I used RXJS observables to provide stream of data, and wherever I subscribed to it in any Component then I also unsubscribed in On Destroy of the component in `ngOnDestroy()` function:

```
ngOnDestroy(): void {
  if (this.subscription) this.subscription.unsubscribe() ;
}
```

This prevents Memory Leaks and accumulation of unused subscriptions instants in throughout the application which could make it very slow.

In some cases, I didn't unsubscribe because I wanted the subscription to remain active throughout the application like in my subscription in the "header component" to the cart state observable which is in the CartService, because I always want the number of products to be shown and active in the cart icon in the navigation bar:

```
ngOnInit() {
  this.cart.currentData.subscribe(d=> this.cartArray = d) ; |
  this.crud.getUser().subscribe(u => this.loggedUser = u) ;
}
```

Server Side:

In startup.cs file I added my .NET backend services to the dependency injection provider by registering them in the ConfigureServices() method :

```
public void ConfigureServices(IServiceCollection services)
```

One of the services I used is AddSpaStaticFiles() :

```
services.AddSpaStaticFiles(configuration =>
{
    configuration.RootPath = "Wonderland/dist";
});
```

This Service is responsible to allow using Single Page Application Files that are added to the .NET project, in this case it's the Angular Client-Side Application which is saved in the relative path "Wonderland". I made use of it also in Startup.cs file in the method Configure():

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
```

```
app.UseSpa(spa =>
{
    spa.Options.SourcePath = "Wonderland";
    if (env.IsDevelopment())
    {
        spa.UseAngularCliServer(npmScript: "start");
    }
});
```

As seen in the image above, I configured the path of this SPA to use the Angular CLI command line "npm start" to start the Angular App in the same time with the .NET project F5.