**MINI PROJECT**

***Submitted by***

**Shubhra Kumari [RA2011028010093]**
**Dushyant Rao [RA2011028010106]**
**Akil Ahamed [RA2011028010097]**
**Dhawal Gupta [RA2011028010081]**

***Under the Guidance of***

**Dr.N.Senthamarai**

**Associate Professor, Department of Networking and Communication**

***In partial satisfaction of the requirements for the degree of***

**BACHELOR OF TECHNOLOGY IN**
**COMPUTER SCIENCE ENGINEERING**

**with specialization in cloud computing**

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
*Deemed to be University u/s 3 of UGC Act, 1956*

**SCHOOL OF COMPUTING**

**COLLEGE OF ENGINEERING AND TECHNOLOGY**

**SRM INSTITUTE OF SCIENCE AND**

**TECHNOLOGY KATTANKULATHUR - 603203**

## BONAFIDE CERTIFICATE

Certified that this lab report titled **"Custom T-Shirt Design App"** is the bonafide work done by Dushyant Rao (RA2011028010106), Shubhra Kumari (RA2011028010093), Akil Ahamed (RA2011028010097), Dhawal Gupta(RA2011028010081) who carried out the project under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other work.

**SIGNATURE**

**Dr.N.Senthamarai**

**WAD – Course Faculty**

Assistant Professor

Department of Networking

and Communication

## CONTENTS

## OBJECTIVE

The objective of this document is the demonstration of using a SPA to achieve a very fast and responsive web application with diverse functionalities. It aims to solve problems associated with people buying the available against the preferable. The application avails users the opportunity to create their own design either with an existing template from the collections or a new one, it also allows users to store their designs and reedit their designs after it has been created.

In addition, users can use the saved data to make reorders with options such as size selection, print location, print type, color, font style and many more as described in the description box.

The current topic is a preferred study subject because it incorporates the full stack web development with modern technologies that allows users interact with the web page and get desired result which is an improvement to the traditional HTML pages. Before now, pages are often static with minimal or no user interaction involved. This study includes both the frontend (what the user sees and interacts with) and the backend (the enabler of the frontend experience).

## IMPLEMENTATION

A concise study was carried out after getting the requirement specification, the front end is developed via JSP, HTML, CSS, JAVA SCRIPT. A set of JS libraries have been used to format the web page.
To ease out the implementation plan a UML and flow diagram have been created to understand user needs.
After the user needs have been defined, frontend and backend integration has been done for the web page.

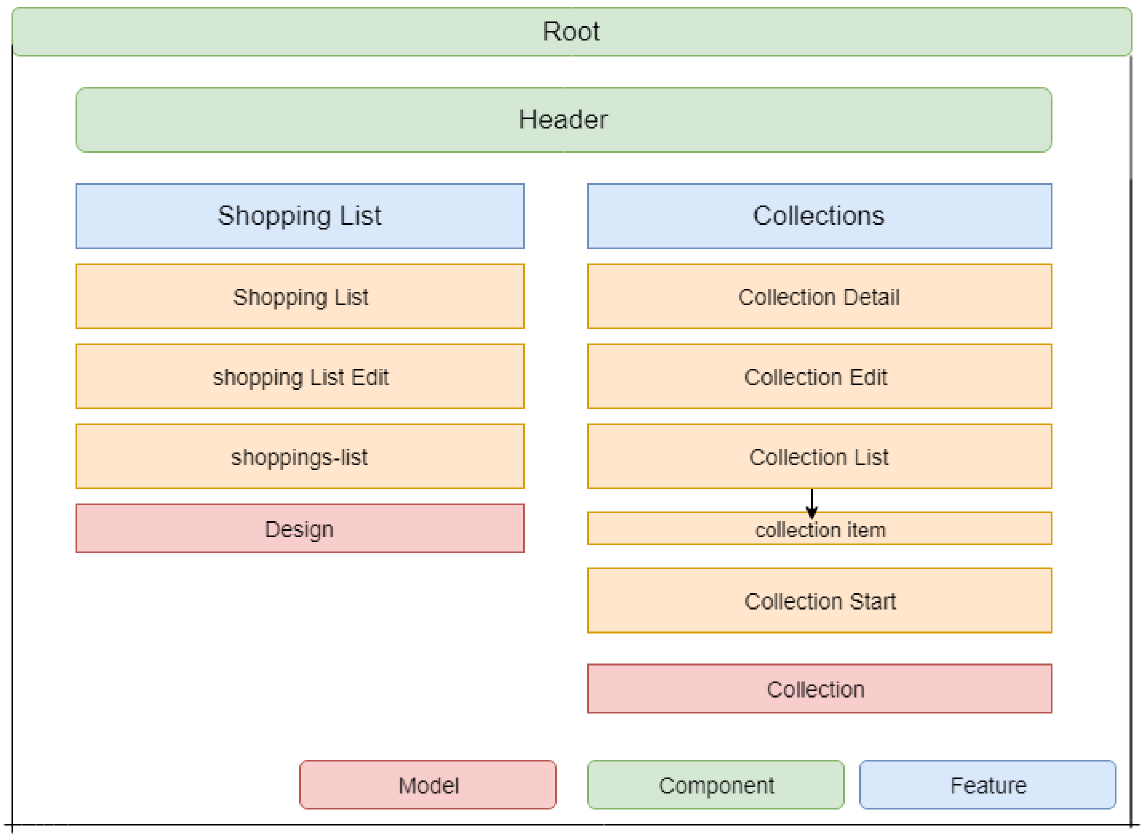
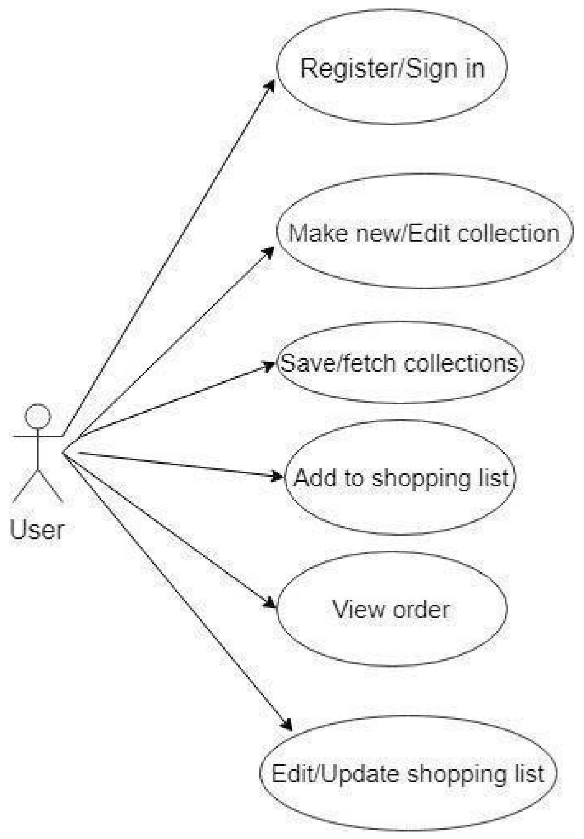


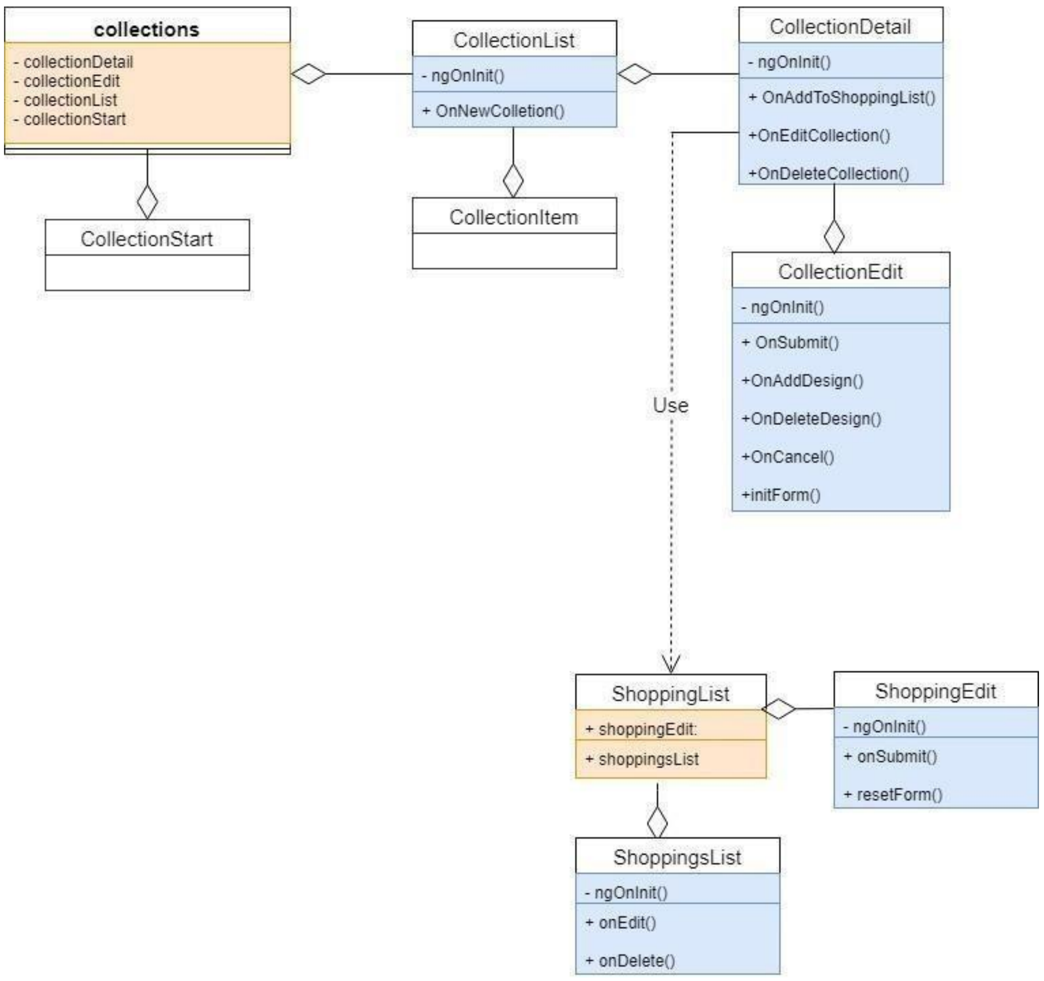
**Figure 1**: Implementation structure of the application

## Use case Diagram

A use case diagram consists of actors or users, uses cases and how they relate. This diagram helps model the system of the application and captures the functionality of the system. It gathers the requirement of the system, gets an exterior view of the system, identifies the external and internal factors that make up of the system and show interactions with the actors or users



## Class Diagram

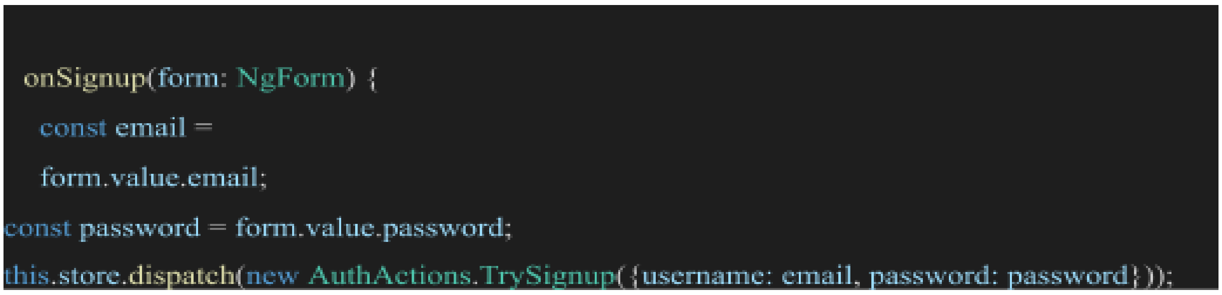


## CODES

This section describes how the code of the application is implemented, where they are used and how they are used to achieve the goal in this application. Angular JS, Angular 5 is used for the frontend of the application and Firebase is used for the database while NodeJS is used to run the backend.

### 1.1 User Sign Up Component

The user signUp component imports the AuthActions from the store where the rules for authenticating is set. The onSignUp method is implemented with the authentication rules which is the username: email and password: password. The rules are set according to the permission granted in Firebase from which each user registry details are stored as shown in Code Snippet 1

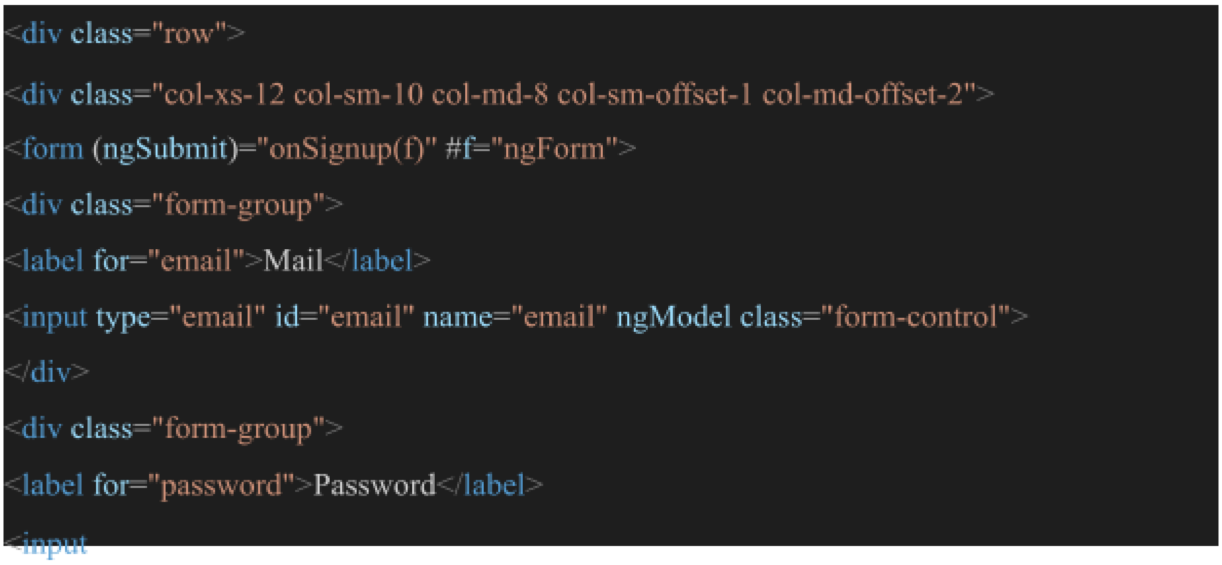


```
 onSignup(form: NgForm) {
   const email =
   form.value.email;
const password = form.value.password;
this.store.dispatch(new AuthActions.TrySignup({username: email, password: password}));
}
```

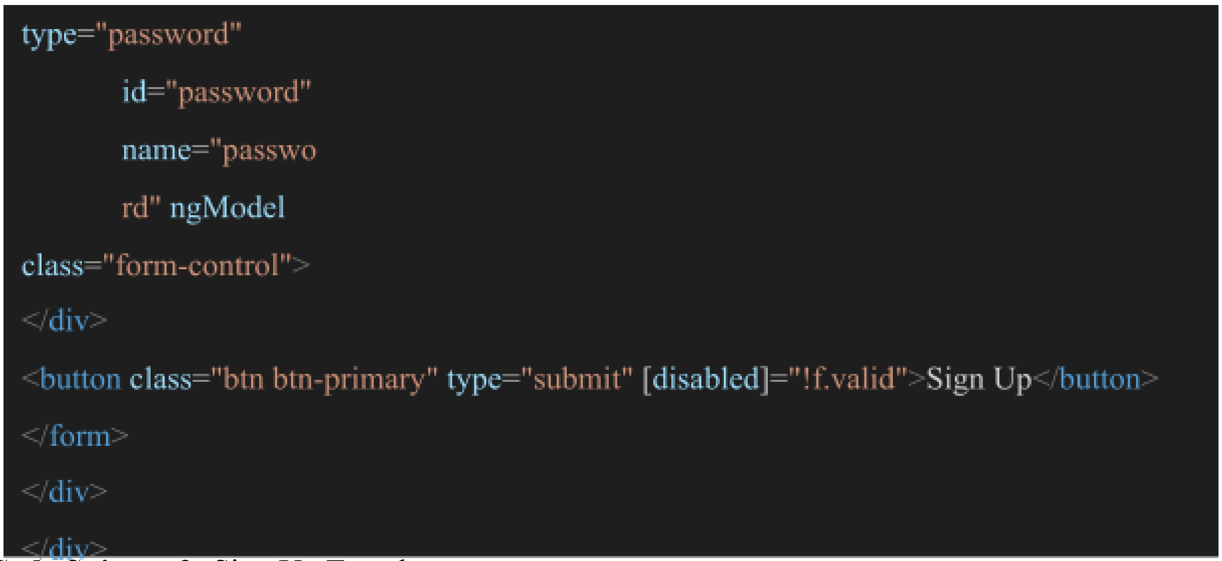
**Code Snippet 1:** Sign Up Function

### 1.2 User Sign Up Template

The template (HTML) represents the visual layout of the class on the web page. A form is created and a disabled function is added to render the submit button invalid until every sign-up detail is filled as shown in Code Snippet 2.



```
<div class="row">
<div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">
<form (ngSubmit)="onSignup(f)" #f="ngForm">
<div class="form-group">
<label for="email">Mail</label>
<input type="email" id="email" name="email" ngModel class="form-control">
</div>
<div class="form-group">
<label for="password">Password</label>
<input
```

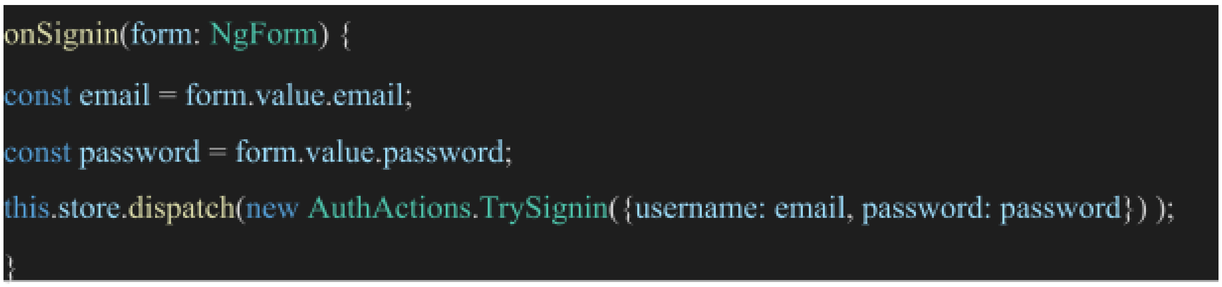


```
type="password"
        id="password"
        name="passwo
        rd" ngModel
class="form-control">
</div>
<button class="btn btn-primary" type="submit" [disabled]="!f.valid">Sign Up</button>
</form>
</div>
</div>
```

**Code Snippet 2:** Sign Up Template

### 1.3 User Sign in Component

The registered user can signin/login to access the functionalities (make new or edit collections and add or edit shopping list) of the application. The Signin class imports the AuthActions which sets the rules to authenticate the user. A token is generated every time the sign in function is called. This identifies the user on the page and gets destroyed when the user logs out. This is shown in Code Snippet 3.

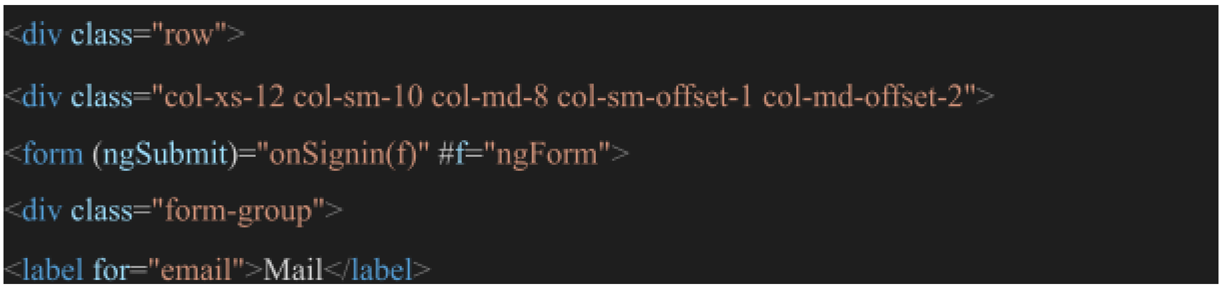


```
onSignin(form: NgForm) {
const email = form.value.email;
const password = form.value.password;
this.store.dispatch(new AuthActions.TrySignin({username: email, password: password}) );
}
```

**Code Snippet 3:** Sign Up Function

### 1.4 User Sign in Template

The sign in template shows the visual layout of the sign in page, only the user email and password is required to authenticate the user. Once the sign in details is invalid, the sign in button remain invalid and sends an error message to the user if details are incorrect. If sign details are correct, it redirects the user to the home page.



```
<div class="row">
<div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">
<form (ngSubmit)="onSignin(f)" #f="ngForm">
<div class="form-group">
<label for="email">Mail</label>
```

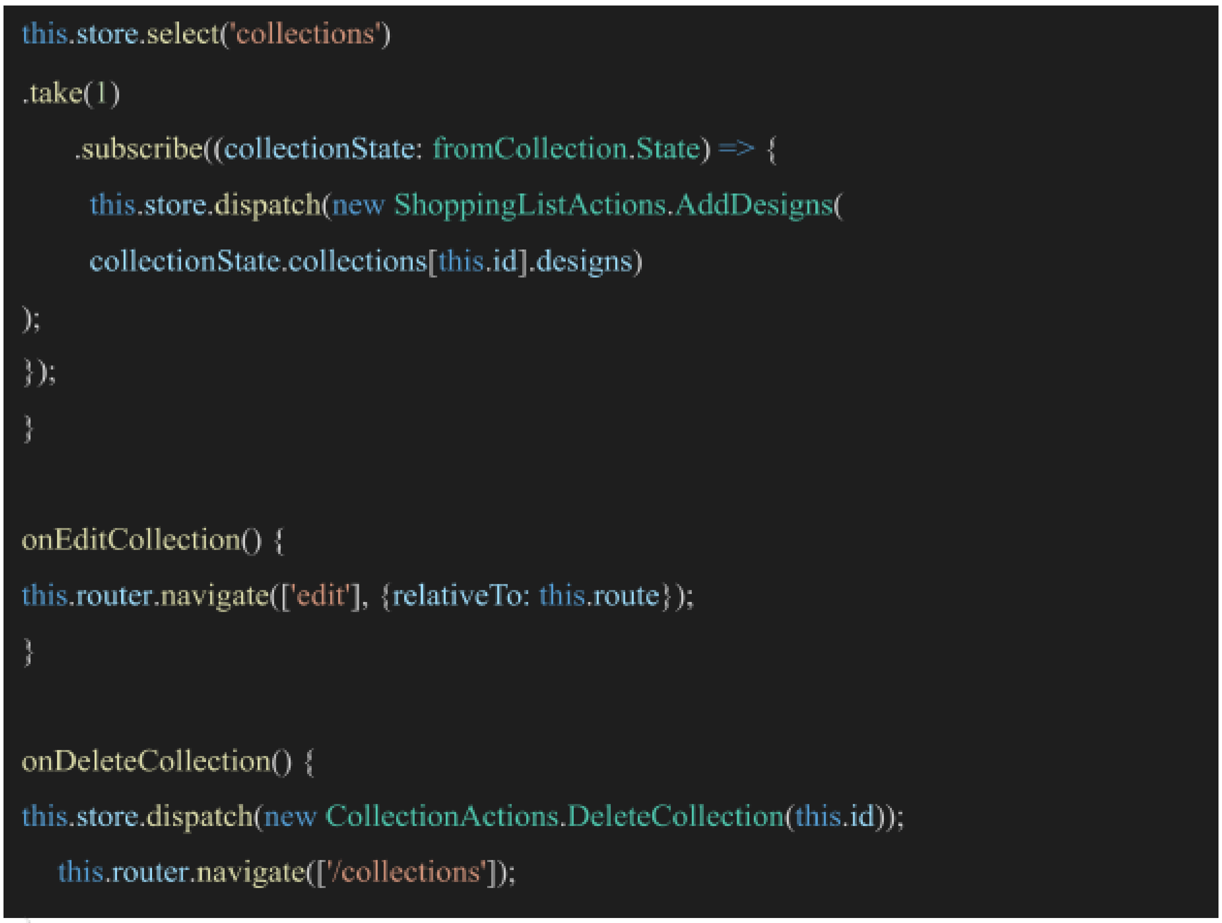
```
<input type="email" id="email" name="email" ngModel class="form-control">
</div>
<div class="form-group">
<label for="password">Password</label>
        <input
          type="passwor
          d"
          id="password"
          name="passwo
          rd" ngModel
class="form-control">
</div>
<button class="btn btn-primary" type="submit" [disabled]="!f.valid">Sign In</button>
</form>
</div>
</div>
```

**Code snippet 4:** Sign in Template

## 1.5 Collection Details Component

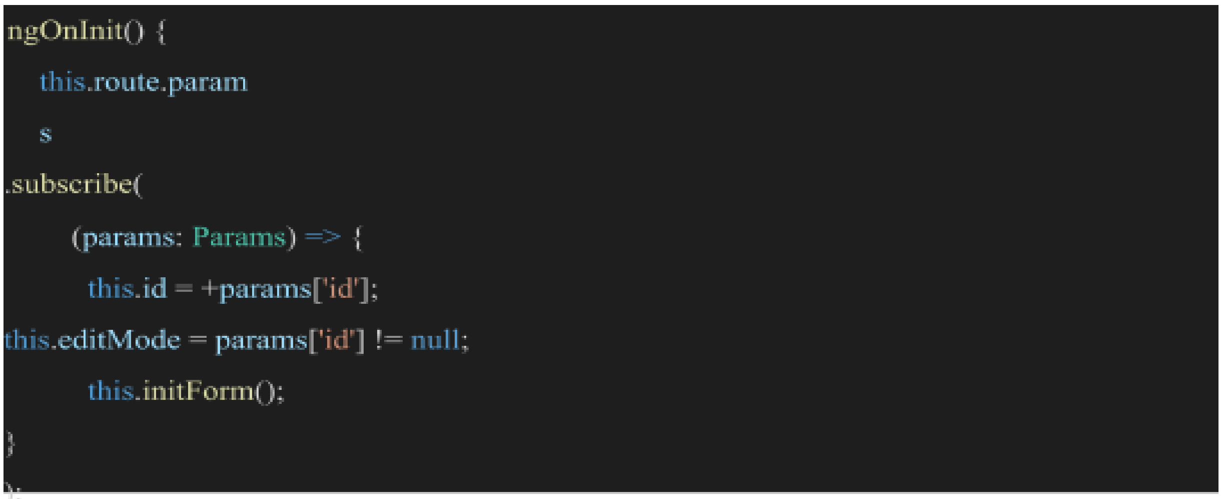
This component handles the function available to each collection in the list. It allows us to manage the collection selected which is only possible via authentication. The registered and logged in user can use the managed collection naviagtion. The functions here allow adding items to shopping list, edit collection and delete collection. The ngOnInit method is the life cycle hook which runs continuously the moment a collection is selected.

```
 ngOnInit() {
   this.route.param
   s
.subscribe(
      (params: Params) => {
        this.id = +params['id'];
this.collectionState = this.store.select('collections');
}
);
}

onAddToShoppingList() {
```


```
this.store.select('collections')
.take(1)
      .subscribe((collectionState: fromCollection.State) => {
        this.store.dispatch(new ShoppingListActions.AddDesigns(
        collectionState.collections[this.id].designs)
);
});
}

onEditCollection() {
this.router.navigate(['edit'], {relativeTo: this.route});
}

onDeleteCollection() {
this.store.dispatch(new CollectionActions.DeleteCollection(this.id));
    this.router.navigate(['/collections']);
}
```
**Code Snippet 5:** Collection Detail Component

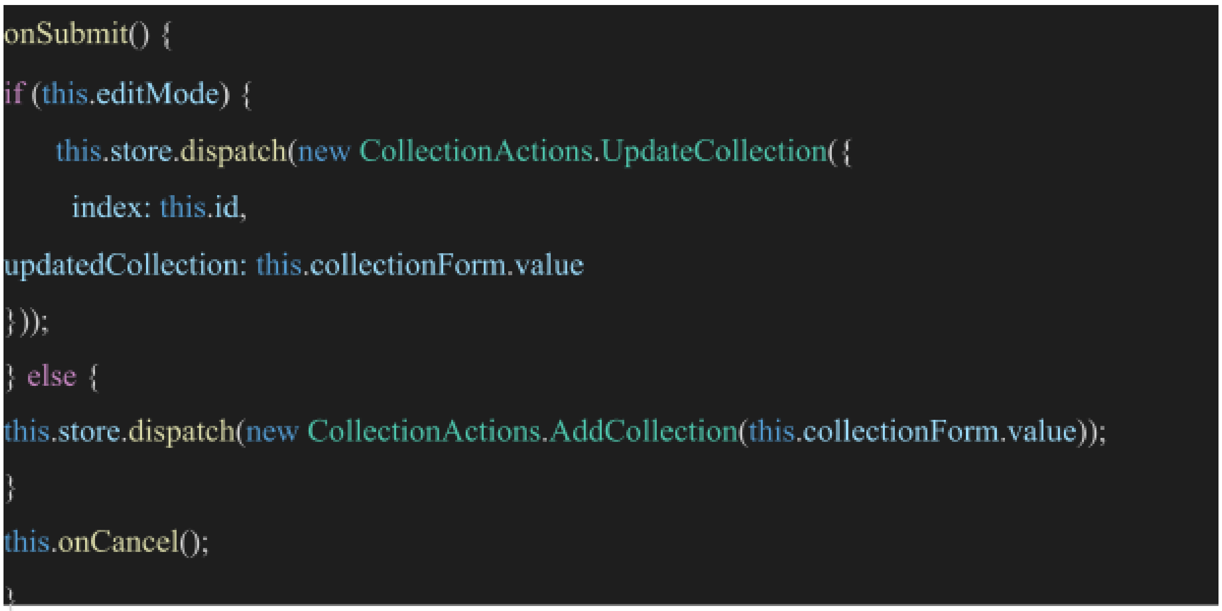
## 1.6 Collection Edit Component

In this component there are several methods that all work together to make up the edit collection. The first method implemented is the ngOnInit which is the life cycle hook of the component


```
ngOnInit() {
    this.route.param
    s
.subscribe(
        (params: Params) => {
          this.id = +params['id'];
this.editMode = params['id'] != null;
          this.initForm();
}
);
}
```
**Code Snippet 6:** ngOnInit Method

### 1.6.1 onSubmit Method

onSubmit method checks if in edit mode and dispatches to the UpdateCollection method call from the collections.actions in the store. If not in the edit mode, the AddCollection method is called and a new collection is created as shown in Code Snippet 7.

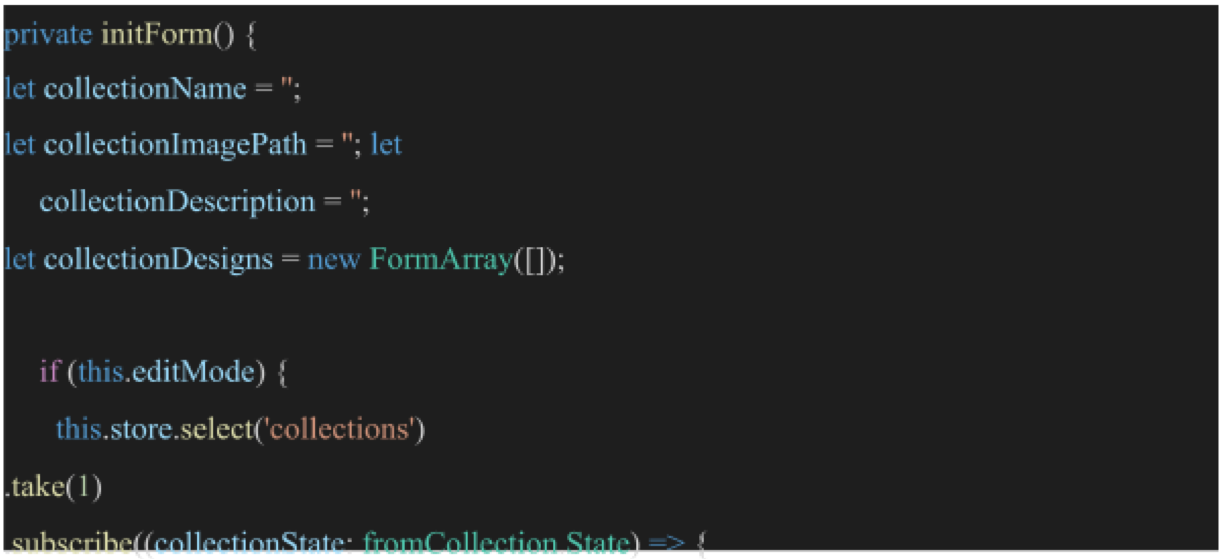


```
onSubmit() {
if (this.editMode) {
     this.store.dispatch(new CollectionActions.UpdateCollection({
       index: this.id,
updatedCollection: this.collectionForm.value
}));
} else {
this.store.dispatch(new CollectionActions.AddCollection(this.collectionForm.value));
}
this.onCancel();
}
```

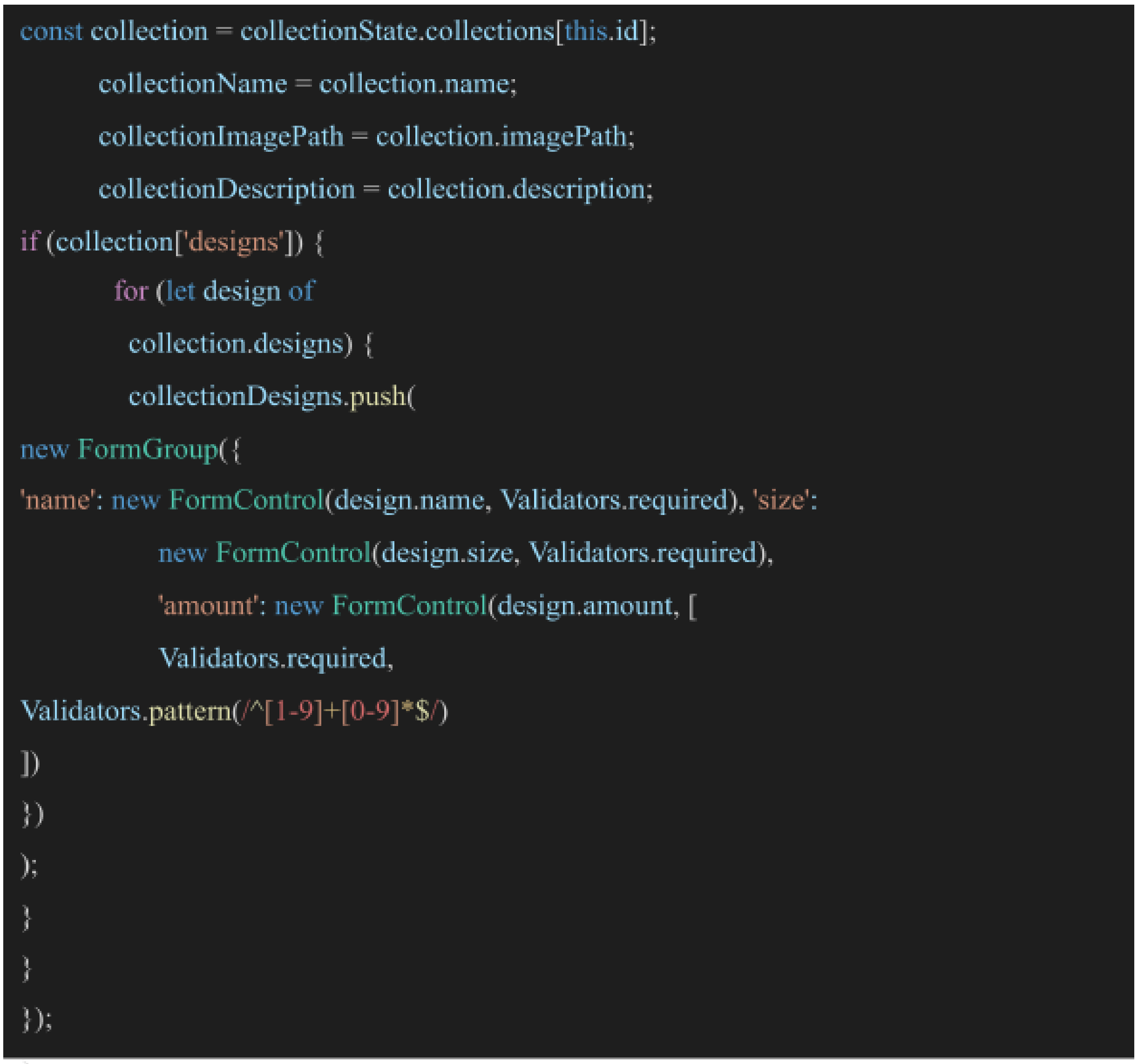
**Code Snippet 7:** onSubmit Method

### 1.6.2 InitForm Method

InitForm method creates the form on which each collection detail is filled. If in the edit mode, the details of the collection are displayed in the box and the detail can be edited. If not in the edit mode, an empty form is created with form validation in place to ensure no input is left empty by the user before submission. A pattern is included for the amount to ensure users cannot enter string or negative numbers. The method is made private to ensure it is only called within this component.



```
private initForm() {
let collectionName = '';
let collectionImagePath = ''; let
    collectionDescription = '';
let collectionDesigns = new FormArray([]);

    if (this.editMode) {
      this.store.select('collections')
.take(1)
.subscribe((collectionState: fromCollection.State) => {
```

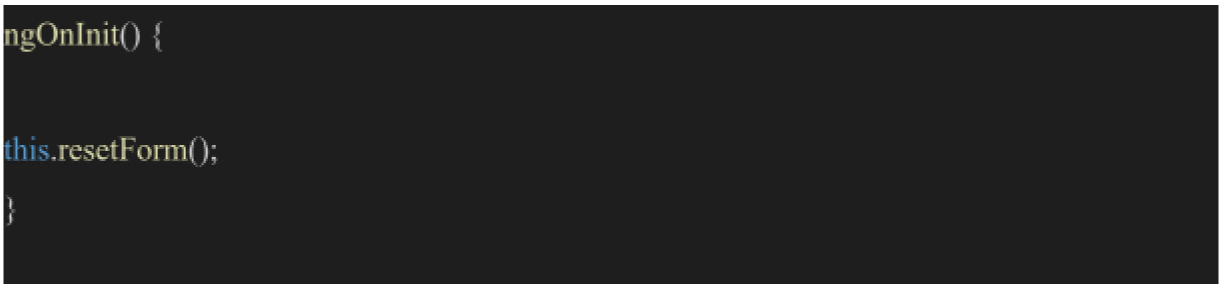


```
const collection = collectionState.collections[this.id];
        collectionName = collection.name;
        collectionImagePath = collection.imagePath;
        collectionDescription = collection.description;
if (collection['designs']) {
          for (let design of
            collection.designs) {
            collectionDesigns.push(
new FormGroup({
'name': new FormControl(design.name, Validators.required), 'size':
              new FormControl(design.size, Validators.required),
              'amount': new FormControl(design.amount, [
              Validators.required,
Validators.pattern(/^[1-9]+[0-9]*$/)
])
})
);
}
}
});
}
```

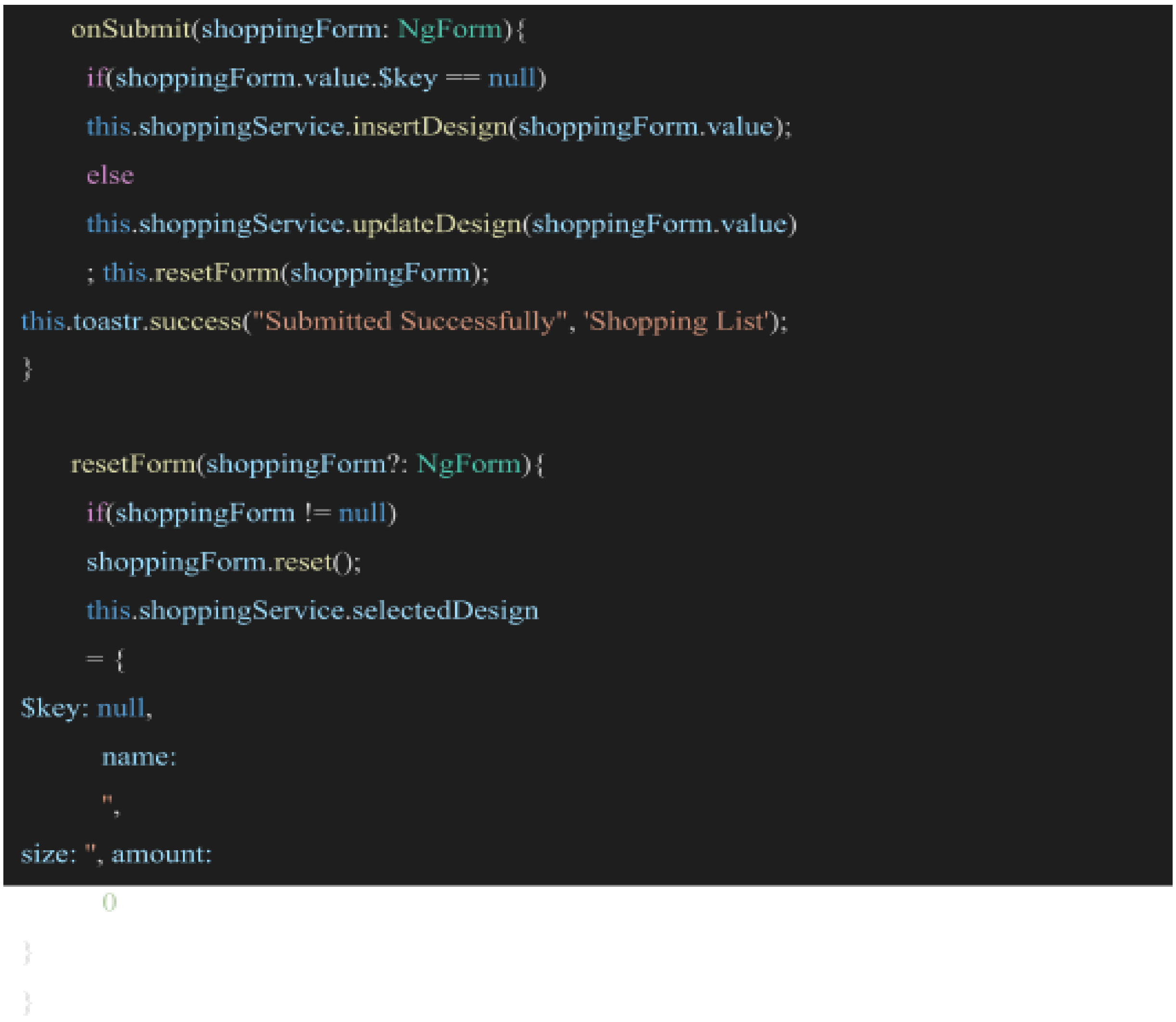
**Code Snippet 8:** initForm Method

### 1.7 ShoppingList Edit Component

This component is responsible for pushing the shopping list items into the database, the submit method and the reset form method are called in this component. The reset form method is called from the ngOnInit method which is the life cycle hook of the component. The insertDesign and updateDesign method will be called from the shoppingService and this service is imported into this component.



```
ngOnInit() {

this.resetForm();
}
```



```
    onSubmit(shoppingForm: NgForm){
      if(shoppingForm.value.$key == null)
      this.shoppingService.insertDesign(shoppingForm.value);
      else
      this.shoppingService.updateDesign(shoppingForm.value)
      ; this.resetForm(shoppingForm);
this.toastr.success("Submitted Successfully", 'Shopping List');
}

    resetForm(shoppingForm?: NgForm){
      if(shoppingForm != null)
      shoppingForm.reset();
      this.shoppingService.selectedDesign
      = {
$key: null,
        name:
        '',
size: '', amount:
        0
}
}
```

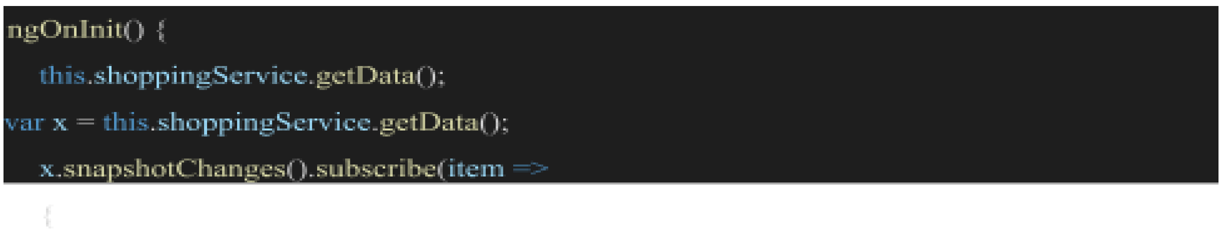
**Code Snippet 9:** shoppingEdit component

## 1.8 Shoppings List Component

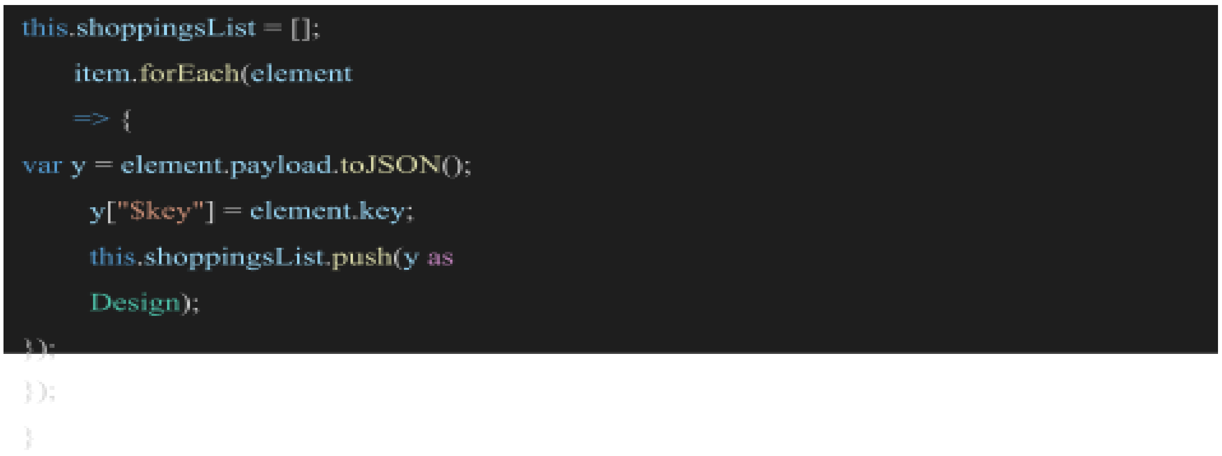
In this component, item values are edited and deleted in real time from the database. Users can edit (update and delete) their submitted and saved item. The shoppingService is imported to allow the use of some methods already implemented in it.

### 1.8.1 ngOnInit Method

This method is always called and it gets data from the database. It fetches each user's data from the database and displays on the shopping list page.



```
ngOnInit() {
   this.shoppingService.getData();
var x = this.shoppingService.getData();
   x.snapshotChanges().subscribe(item =>
     {
```

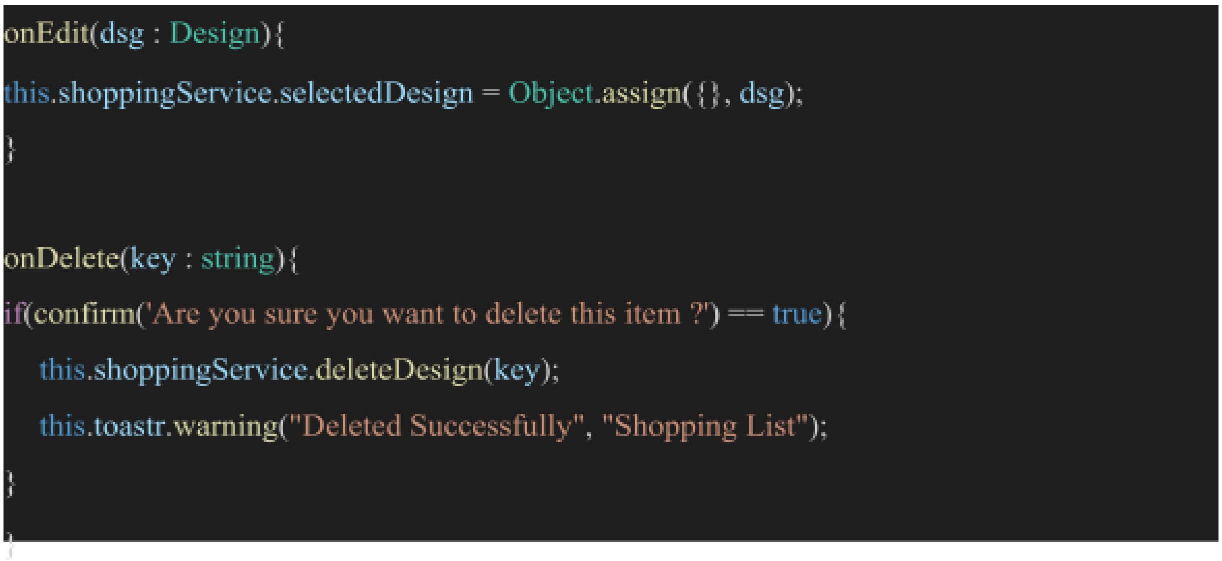


```
this.shoppingsList = [];
      item.forEach(element
      => {
var y = element.payload.toJSON();
        y["$key"] = element.key;
        this.shoppingsList.push(y as
        Design);
});
});
}
```

**Code Snippet 10:** ngOnInit Method

1.8.2 onEdit and onDelete Methods

The onEdit and onDelete methods are responsible for manipulating the data and respond when they are called

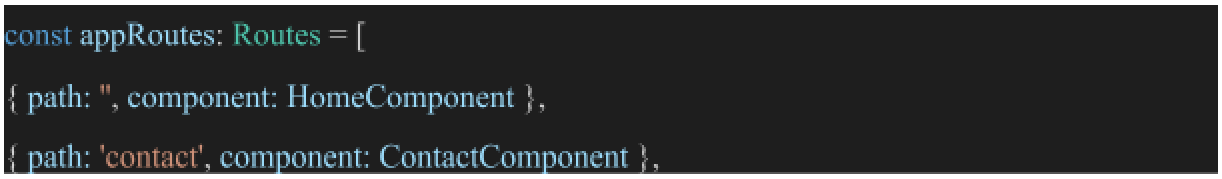


```
onEdit(dsg : Design){
this.shoppingService.selectedDesign = Object.assign({}, dsg);
}


onDelete(key : string){
if(confirm('Are you sure you want to delete this item ?') == true){
    this.shoppingService.deleteDesign(key);
    this.toastr.warning("Deleted Successfully", "Shopping List");
}
}
```

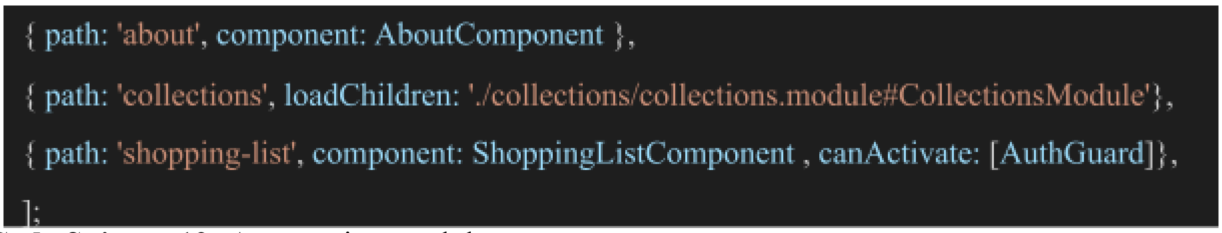
**Code Snippet 11:** onEdit and onDelete Methods

## 1.9 App Routing Module

The routing module shows how the routing of the application is carried out, it imports components to be available in the navigation of the application and their paths are set as seen in the Code Snippet 11. The AuthGuard is imported and used to access the page that needs authentication. The collections is loaded with children and this is called from the collections modules, which loads every child collection component.



```
const appRoutes: Routes = [
{ path: '', component: HomeComponent },
{ path: 'contact', component: ContactComponent },
```

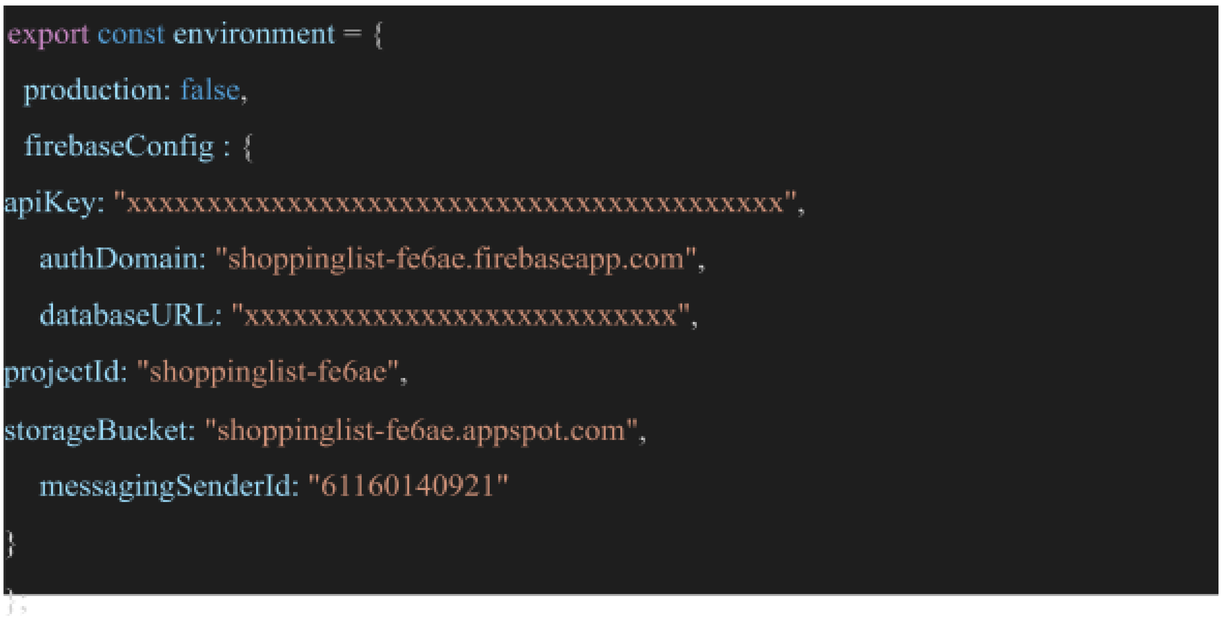


```
{ path: 'about', component: AboutComponent },
{ path: 'collections', loadChildren: './collections/collections.module#CollectionsModule'},
{ path: 'shopping-list', component: ShoppingListComponent , canActivate: [AuthGuard]},
];
```

**Code Snippet 12:** App routing modules

### 1.10 Firebase setup

To set up Firebase, create a new Firebase app on your Firebase console with your Google Account, add project and give a desired name. Afterwards, "Add Firebase to your web app" is clicked from the project overview. The config key is copied into *src/environments/environment.ts*



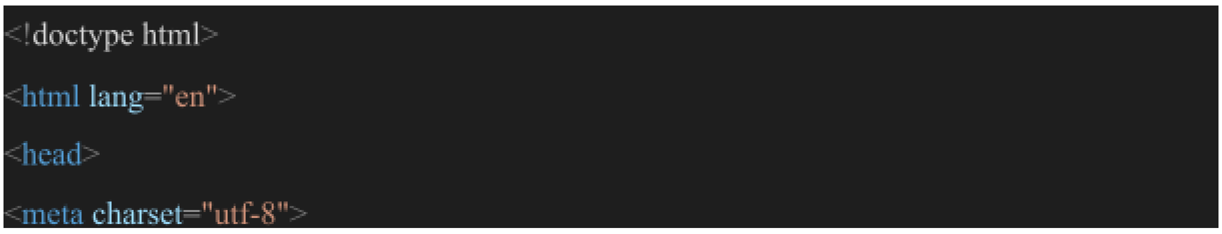
```
export const environment = {
 production: false,
 firebaseConfig : {
apiKey: "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
  authDomain: "shoppinglist-fe6ae.firebaseapp.com",
  databaseURL: "xxxxxxxxxxxxxxxxxxxxxxxxxxxx",
projectId: "shoppinglist-fe6ae",
storageBucket: "shoppinglist-fe6ae.appspot.com",
  messagingSenderId: "61160140921"
}
};
```

**Code Snippet 13:** Adding firebase to the web app

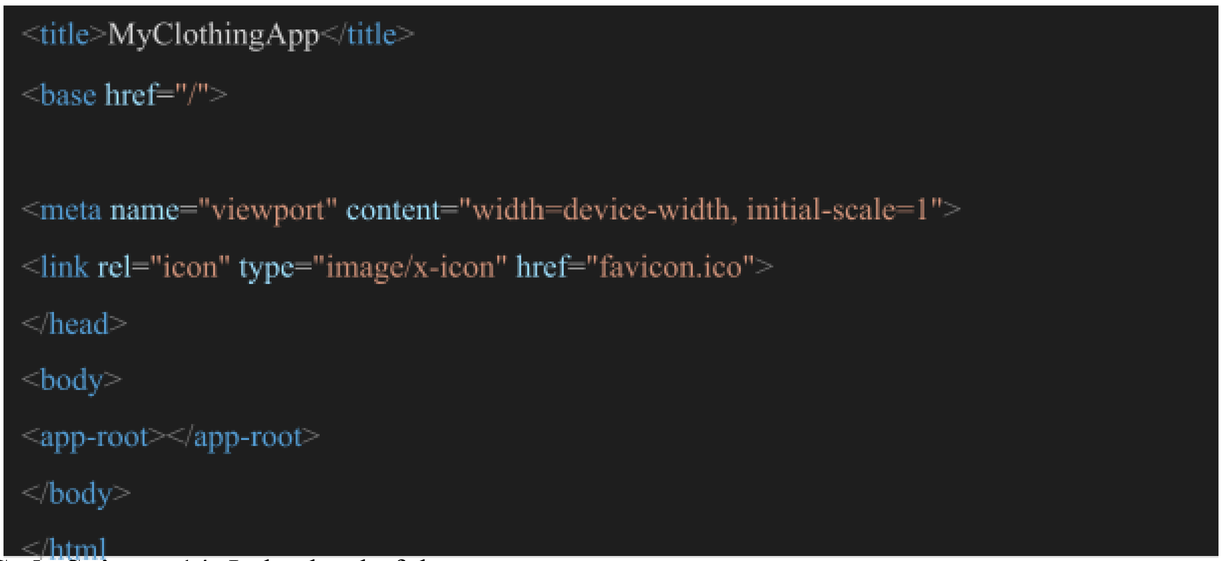
From the Database tab, the Rules are edited depending on how it should respond on the web page, and then published. /8/

### 1.11 Index.html

This is main view of the application from which other views are injected and displayed as a Single Page. Every other view in the app is initiated from the app-root. The app components are injected through the <app-root></app-root>.



```
<!doctype html>
<html lang="en">
<head>
<meta charset="utf-8">
```

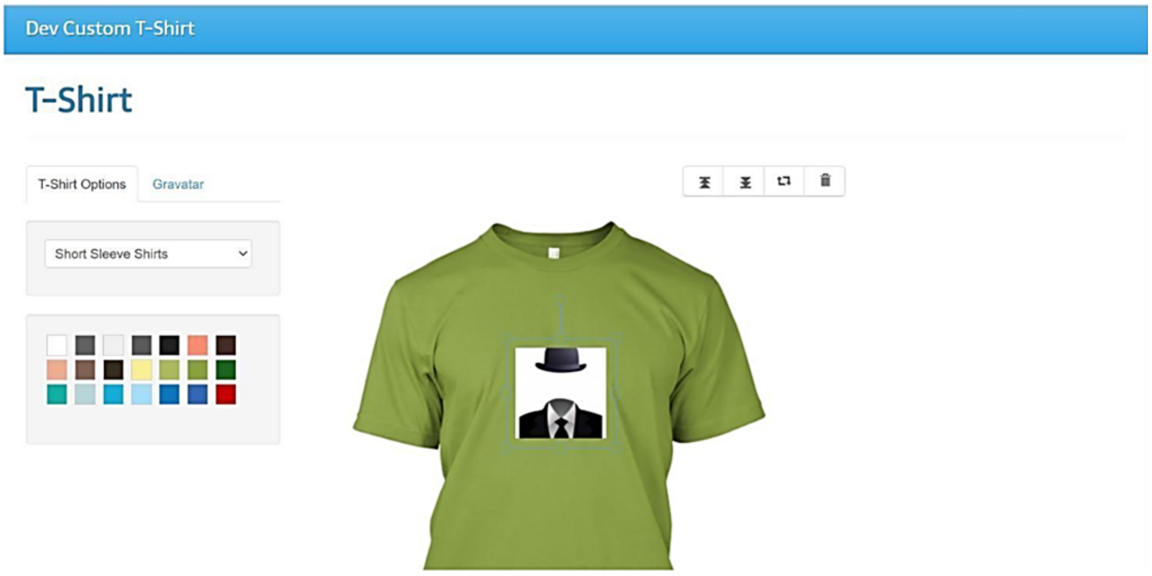


```
<title>MyClothingApp</title>

<base href="/">


<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="icon" type="image/x-icon" href="favicon.ico">

</head>

<body>

<app-root></app-root>

</body>

</html
```

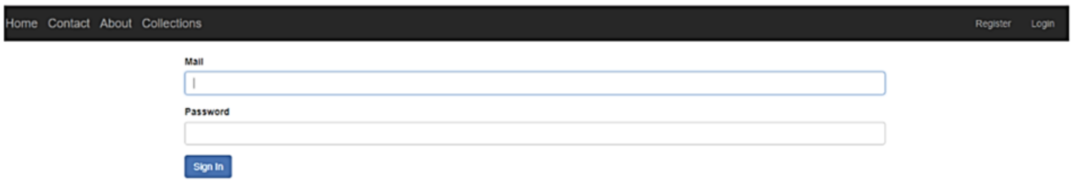
**Code Snippet 14:** Index.html of the app

## Graphical User Interface (GUI)

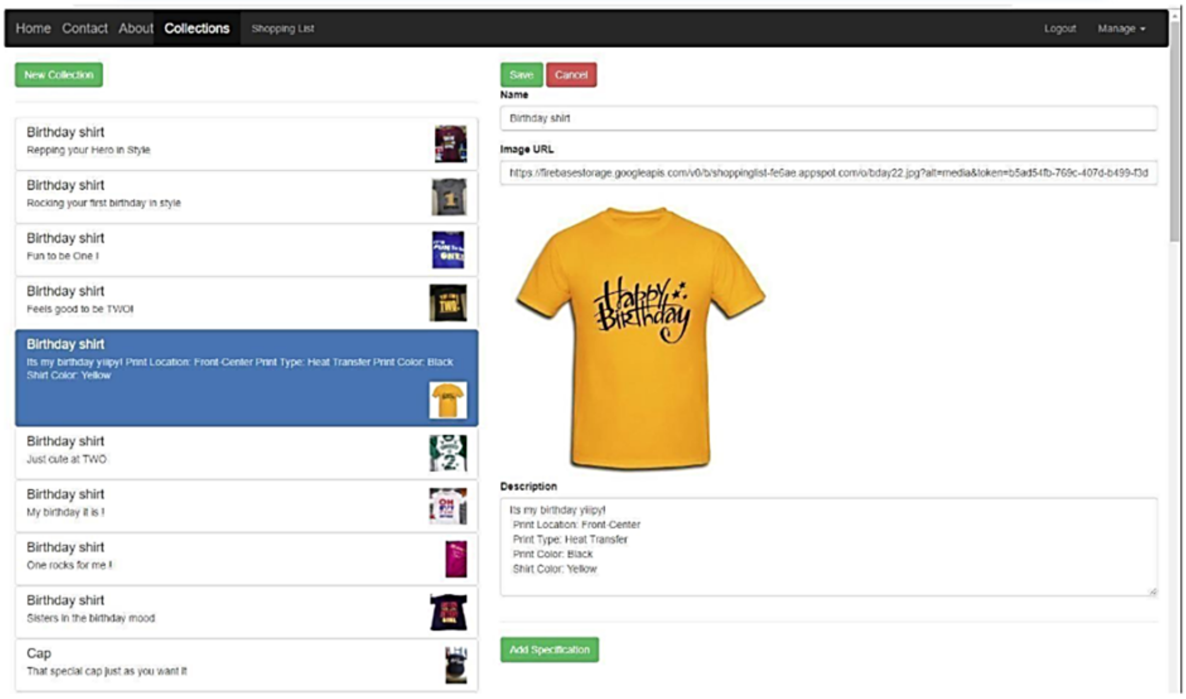
### 1) Home Page:



### 2) Login Page:



## 3) T-shirt Collection Page:



Birthday shirt
Repping your Hero in Style

Birthday shirt
Rocking your first birthday in style

Birthday shirt
Fun to be One !

Birthday shirt
Feels good to be TWO!

Birthday shirt
Its my birthday yiipy! Print Location: Front-Center Print Type: Heat Transfer Print Color: Black Shirt Color: Yellow

Birthday shirt
Just cute at TWO

Birthday shirt
My birthday it is !

Birthday shirt
One rocks for me !

Birthday shirt
Sisters in the birthday mood

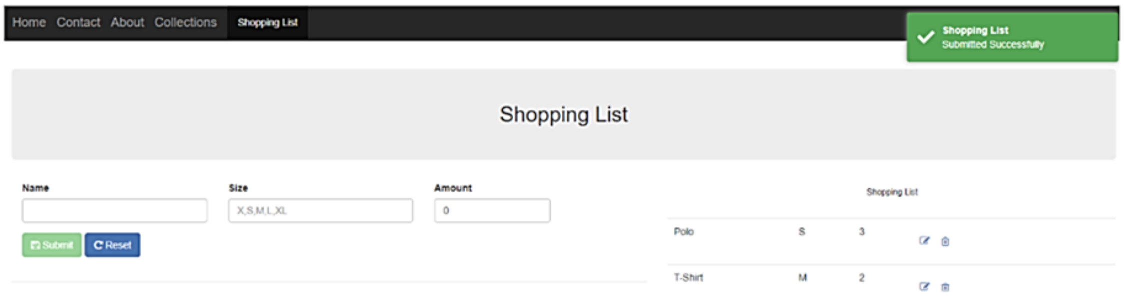
Cap
That special cap just as you want it

Name
Birthday shirt

Image URL
https://firebasestorage.googleapis.com/v0/b/shoppinglist-fe6ae.appspot.com/o/bday22.jpg?alt=media&token=b5ad54fb-769c-407d-b499-f3d

Description
Its my birthday yiipy!
Print Location: Front-Center
Print Type: Heat Transfer
Print Color: Black
Shirt Color: Yellow

## 4) Shopping List Page:



Shopping List
Submitted Successfully

Shopping List

Name

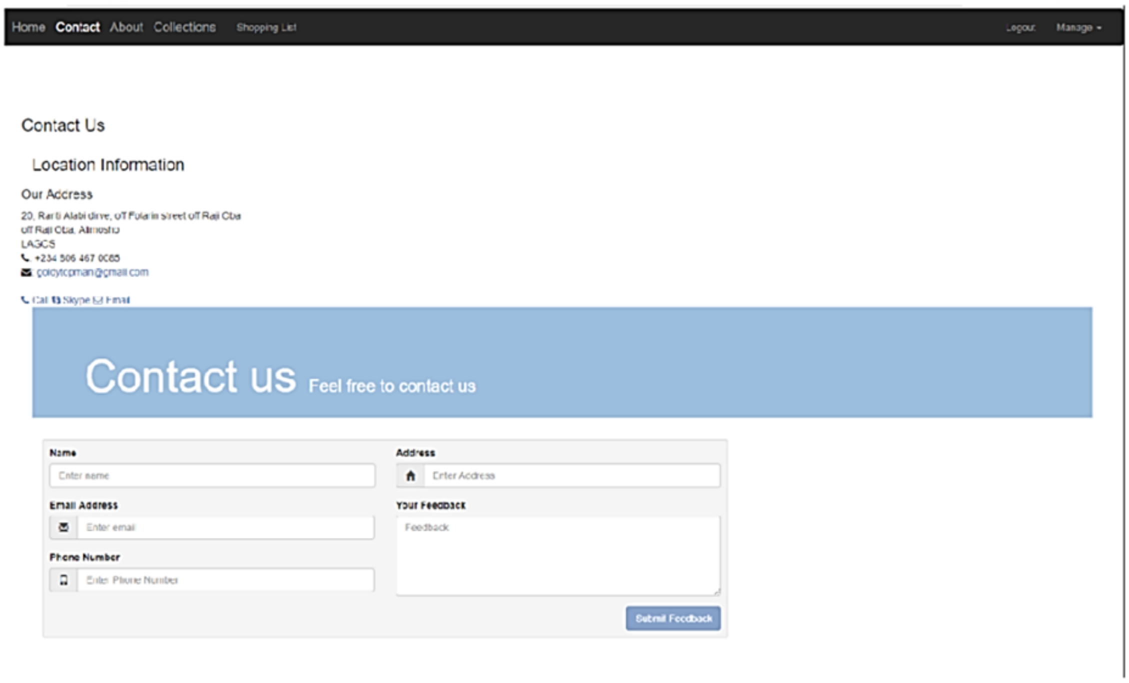
Size
X,S,M,L,XL

Amount
0

Shopping List

| | | |
|---|---|---|
| Polo | S | 3 |
| T-Shirt | M | 2 |

## 5) Contact Page:



Contact Us

Location Information

Our Address

20, Rafiu Alabi drive, off Folarin street off Raji Oba
off Raji Oba, Alimosho
LAGOS
+234 506 467 0085
coicytcpman@gmail.com

Call Skype Email

Contact us Feel free to contact us

Name
Enter name

Address
Enter Address

Email Address
Enter email

Your Feedback
Feedback

Phone Number
Enter Phone Number

## CONCLUSION AND FURTHER IMPROVEMENTS

This documentation includes the necessary information on the structure and building of the Custom Clothing Web Application. It has considered the use of different technologies and it involves communication between the frontend and backend of the application. It has considered feasibility studies of similar applications, concise analysis, design and implementation.

The use of Angular has proven to be one of the best technology for the design and implementation of this application with its ability to provide SPA which offers very good user experience and responsiveness, fast implementation and helpful tutorials to manage difficult tasks. The communication of Angular with Firebase on the backend has helped develop the application and incorporate functionalities with less difficulty.

Furthermore, the application has been tested against bugs and other factors that could have been unnoticed in period of development. The use of in-built testing environment provided by Angular CLI has made the test easy to write and monitor in real-time.

Creating a unique design from scratch was one of the challenging task faced in the development of this application which involves creativity and structuring.

For further improvements, Angular offers lots of functionalities that were not fully explored in the design and implementation of this application. These functionalities would help improve the interface by adding more components and animations, make the web page more dynamic for users and provide more services for users to make their custom cloths. The view of the application can be further improved, in which user input in the description field can be seen dynamically on every selected collection without just listing them in the description box of the application.

The database can be further designed with a more robust NoSQL database like MongoDB in the future when the data size increases.

Lastly, full test should be done on the application such as the integration and end-2-end(E2E) test which involves browser automation, useful for insurance policy and sanity-check. This test will alert when something goes wrong for deep investigation.