

Men's Fashion Web Application

Student Name: Tarun Boricha

Enrollment ID: **201901080**

B. Tech. Project (BTP) Report

BTP Mode: *Off Campus*

Dhirubhai Ambani Institute of ICT (DA-IICT)

Gandhinagar, India

201901080 [at] daiict.ac.in

Mentor's Name: Jigar Bagdai

Company's Name: *Clomotech*

Company's Address: HD-128, WeWork Krishe Emerald
Kondapur Main Road, Laxmi Cyber City, Whitefields, Kondapur

Hyderabad, Telangana 500081, India

jigarbagdai [at] clomotech.com

On-Campus Mentor: BTP Coordinator

Abstract—This document is a model and instructions for Single Page Web Application. In this report, we discussed all the necessary steps that are required to build this Web Application. For development, we used Angular (TypeScript framework). We also discussed all the business logic that is used in this Web Application. Angular is used to develop more modular and more complex Web Applications. This application is for store seller that runs offline fashion business and wants to run online business also. This application shall enable all the CRUD operations for the seller to perform on the business product for makes the business task easier.

I. INTRODUCTION

In this project, I developed Fashion Web-APP using Angular. Angular is a framework based on TypeScript. Here I used Angular because Angular is a framework of TypeScript and TypeScript is a free open-source high-level programming language. TypeScript is OOPs based high-level programming language which is why we can use interesting OOPs features in our project and build an amazing Single Page Application also called Web-APP. Angular latest version is 15 and we used the latest version of Angular for our Web-APP. Angular is very useful for making Single Page Application's front-end modules. Single Page Application means JavaScript changes DOM (Document Object Model) means it changes HTML code during runtime. And that is why we cannot see the refresh page every time we interact with our Web-APP because Angular is not reached out to the server for every page load instead it loaded data in the background and JavaScript will change every DOM as the user want.

The application is a Men's fashion Application. Prepose of building this application is to provide ease to the Business owner that is running a small fashion business. Using this application business owners can put his/her business product on the Internet so that anyone can interact with the application using any device that has an Internet connection because it is Web-APP. This application will provide Business owners with functionality like create-product, read-product, update-product, and delete-product. The business owner can do all these operations very efficiently so which will provide easiness to the business owner to run the business. Users can sign-up for this application by providing its details and able to login into this application after that users can finally able to shop in this application and place the order also.

There are some prerequisites to understanding this report with better understanding.

- HTML (Hyper Text Markup Language) For App Structure
- CSS (Cascading Style Sheet) For Styling
- TypeScript (Primary Programming Language for Angular Web-APP Development) For App Logic

II. LEARNING OUTCOMES

Here is the list of Technical Skills that I acquired during my internship:

- Web-Development
- HTML Programming Language
- CSS
- Bootstrap (CSS framework)
- JavaScript
- Angular (A JavaScript Framework for Front-end Development)
- TypeScript (Primary Programming Language for Angular)

III. CONTRIBUTIONS

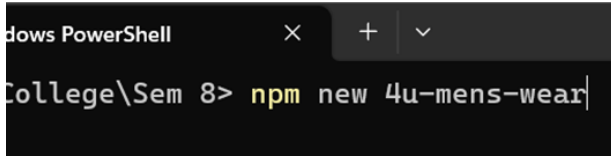
- Build Single-Page-Application
- Build Fashion Web Application on Angular (TypeScript framework) that is build and managed by Google.
- Used JSON server for back-end part of the application.

IV. PROJECT SETUP

First, we need Angular CLI that will help to automate many development tasks that are required to build our Web-APP. CLI also helps to heavily optimize our code when we finally deploy our application to the browser. And it also converts TypeScript code to JavaScript code because browsers only understand JavaScript language. We can install Angular CLI globally (so that we cannot install it every time we create a new angular Web-APP) by following the command:

```
Sem 8> npm install -g @angular/cli
```

After that, we can create our Angular-APP using the following command:



```
Windows PowerShell
College\Sem 8> npm new 4u-mens-wear
```

Here we are using routing in our Application, Router will provide to move from one view of the application to another view so to navigate between the different views in our Web-APP we need Router. Every view has its component or vice versa every component has its view. For styling our Web-APP pages we are using CSS. Here we have to modify our app component HTML template so that we can use Router in our application and we are ready with our application setup. We also installed Bootstrap in our application to easily style our web pages.

V. DOCUMENT THE REQUIREMENTS:

- The seller can log in to Web-APP
- Creation and management of product by the seller (All CRUD operations on a product can be done by the Seller)
- The seller filters the product to make CRUD operation easier
- Users can Sign-up into Web-APP
- Users can log in to Web-APP
- Single Page Application (Primary Goal)
- User can shop
- User Places the order
- The user has a cart and can add and remove the product from the cart

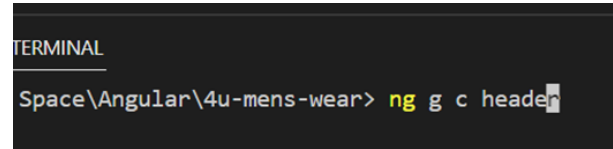
VI. PROJECT WORK-FLOW

Let's start with Components are the primary key feature of Angular and we also called them a basic building block of our Angular application. In this application, we are going to build many different components and then finally we can compose our application using these components. At first, when we are creating our Angular app in the project setup, we have one root component also called the app component which holds our entire application. So, this root component will be the component where we later nest or add the other components that we are going to build in this project (we can also call it the basic building block of our application).

Why components are needed and why it's very useful when we build complex web applications? It gives us the functionality to split up our complex web app into reusable parts and we can use these components more than once in different places on the web pages. The component has its logic so we also don't have to write business logic again when we are reusing that component. Components have their own styling and HTML template so that we also don't have to again write the code.

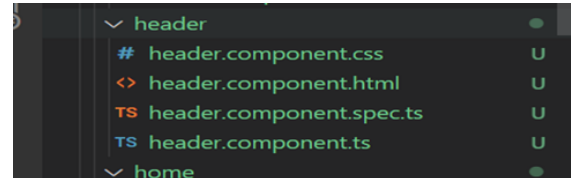
A. Header Component

Let's start with creating a Header component we can create the component in our application using the following command:



```
TERMINAL
Space\Angular\4u-mens-wear> ng g c header
```

After running this command, we get the following files in our project:



```
▼ header
  # header.component.css      U
  <> header.component.html     U
  TS header.component.spec.ts U
  TS header.component.ts      U
▼ home
```

Here header.component.css file will contain styling for that component, the header.component.html file will contain the HTML template, and all the business logic of the header component will hold in the header.component.ts file.

Our header component will contain a side navigation button, business branding, a search bar, and a menu bar. Here menu bar will change dynamically according to different scenarios if the user has a login, then the menu bar will be for the user, and the same will change for seller and no-user login.

Business Logic for the header: so, to dynamically change the menu bar in the header we have to use the Router module. For understanding more about modules, you can check this reference: Modules. Using the router module we can get the current URL of our Web-Application and according to Web-URL, we change our menu bar. For that, we have to take one variable called menuBarSwitch which is a string type variable and this variable will have different values according to different scenarios if the user login into the application, then we will assign the menuBarSwitch variable to 'user'. To find whether the user login into the application or not we have to check local storage and find if there is any data of the user. To find whether we are in the seller section of our web application we can check our current Web-APP URL and if the URL has a value of seller, then we assign menuBarSwitch to 'seller'. And if none of these conditions is true then we assign the variable to 'default'. After successfully assigning a variable value, we can use the ngSwitch directive to hide or show html template code depending on a variable value.

B. User Component

Now let's move to the User component. The user component will contain a user login and sign-up page. And template of the component will contain a login and sign-up form and the user can log in or sign-up by filling this form. Here we are going to use Angular FormsModule to retrieve the data from the form and after that, we can store this data in the back end for User authentication. After the user successfully login

into the application we can store our local Cart data in the User Cart using the following logic: so first we can get our local Cart data from local storage and then we are going to store this data in the user Cart. We will also discuss how we can store product data in Cart for both scenarios whether User login or without User login. But first, let's discuss how User authentication can be done using the service. Use this reference to understand what is service in Angular:Service. We can generate service in our Angular application using the following command:

```
mens-wear> ng g service services/user
```

After generating a user service, we can create a User Sign-up service in our user service. This function has data as input parameter (data of user details like name, email, and password) and we can now store this data using HttpClientModule in the back-end using the post method.

```
constructor(private http: HttpClient, private router: Router) { }
userSignupservice(data: signup) {
  if (data.name != '' && data.email != '' && data.password != '') {
    this.http.post('http://localhost:3000/user',
      data, { observe: 'response' }).subscribe((result) => {
        this.router.navigate(['']);
      });
  }
}
```

This is a snippet for the User sign-up function and after successfully storing the data we can navigate to the home page using a router.

User Login Service: This function has data as input parameter and using this data we can request to get this data using HttpClientModule Get method. And if data is present in back end, then we get some result from the Get method. And if result has value of true and has some length then user login authentication is success and now, we store this result in local storage for the reference of whether user is login or not.

```
UserLoginService(data: Login) {
  this.http.get('http://localhost:3000/user?email=${data.email}&password=${data.password}',
    { observe: 'response' }).subscribe((result: any) => {
      if (result && result.body && result.body.length) {
        localStorage.setItem('user', JSON.stringify(result.body));
        this.router.navigate(['']);
      } else {
        this.UserLoginFailed.emit(true);
      }
    });
}
```

This is a snippet for the User login service function and after successfully storing the data in the local storage we can navigate to the home page using a router. The same authentication for seller login is used in this web application.

C. Add-to-Cart Component

The add-to-Cart component is a complex and very important component we can say because we used so much business logic for this component and it's also a very interesting feature of our web application. A user can add the product to the cart from the product details page. First, we have made some

conditions for adding the product to the cart. So, a user can not add the product before choosing the size of the product. After selecting the size, the Add-to-Cart button is now enabled and we used event binding in this button to add the product to the cart.

So, to add the product to the cart we have two scenarios: first is with user login and second is without user login. Let's first discuss without a user scenario. So, if the user is not login into the application, we store the product that we want to add to the cart in the local storage. Following is the snippet for storing the product in the local storage: The function has one input parameter that we passed in the event binding of the add-to-cart button.

```
localAddToCartService(data: product) {
  let cartData = [];
  let localData = localStorage.getItem('LocaladdToCart');
  if (!localData) {
    localStorage.setItem('LocaladdToCart', JSON.stringify([data]));
    this.cartData.emit([data]);
  } else {
    cartData = JSON.parse(localData);
    cartData.push(data);
    localStorage.setItem('LocaladdToCart', JSON.stringify(cartData));
    this.cartData.emit(cartData);
  }
}
```

Second scenario: with user log in. So, if the user login into the application, then we can store the add-to-cart product in the user cart in the back end so that the user can access its cart from any device by login into the application. For storing the product in the user cart, we used product service. Following is the snippet for the UseraddToCart service:

```
UseraddToCart(data: addToCart) {
  return this.http.post('http://localhost:3000/Cart', data);
}
```

In the product service, we have one event emitter to store the current user's cart data so that if we want details of the current user's cart we don't have to reach every time to the server and get the Cart-list of the current user. This event emitter will assign the data whenever the user login into the application. So, when the user login into the application we request the cart data of that user using the following snippet, and then we assign this data to an event emitter that has the name CartData. Following is the snippet for getting the cart data of the current user:

```
getCartlist(data: number) {
  return this.http.get<product[]>('http://localhost:3000/Cart?userID=${data}',
    { observe: 'response' }).subscribe((result) => {
      if (result && result.body) {
        this.cartData.emit(result.body);
      }
    });
}
```

Here input parameter data is the user id of the current user and we can find this id from local storage where we stored the user details after successfully user login in the application.

D. Seller Guard

Guard in the Angular will control the accessibility of a route. We can control the accessibility of routes based on different conditions using guard. In our application, we

implemented a seller guard to control the seller route. If the seller has not login into the application, then we can access the seller login route but after the login, it's not ideal to have access to the seller login route. For this problem we used Guard and after the seller login into the application, we denied access to the seller login route using Guard.

E. Seller CRUD Operations

The seller can do all the CRUD operations in the application. To perform these operations in the application we used the following product service:

```
AddProductsService(data: product) {
  return this.http.post('http://localhost:3000/products', data, { observe: 'response' });
}

productListService() {
  return this.http.get<product[]>('http://localhost:3000/products');
}

deleteProductsService(data: number) {
  return this.http.delete('http://localhost:3000/products/${data}');
}

updateProductsService(data: product) {
  return this.http.put<product>('http://localhost:3000/products/${data.id}', data);
}
```

Create Product operation: to create or add a new product in the application we have to create the add-product component. In the template of this component, we create one form that will have fields of all the details of the product, and using NgFormModule we can get the data related to the product that the seller filled in this form. After that, we will use AddProductsService in ngSubmit to add the product data in the back-end. The seller can perform other CRUD operations using the different product services mentioned above.

F. Product Details Component

In the back end of our application, every product has its product id to identify the product. So, to open the product detail page we used the router link. We assign the routing path for the product detail component as path: 'product/:Productid'. Here Productid is unique for every product so we can use Productid to get the complete details of that product. We used getProductService of product service to retrieve the data of the product. Following is the snippet for getProductService:

```
getProductService(data: string) {
  return this.http.get<product>('http://localhost:3000/products/${data}');
}
```

Here input parameter data is the product id. And After getting all the details of the product using this service we display all the data of the product in the product details page.

G. Back-end

For the back-end part of the application, we used a JSON server. It is not ideal to use a JSON server for the back-end part of the web development. But in this project, our primary focus is on the front end, not the back end so we don't spend too much time in the back end part. JSON server is the fake API that we can use to test the web application but it's not good practice to use a JSON server. When we finally deploy

the application to the Internet JSON server can not handle big data and it is very slow to handle big tasks.

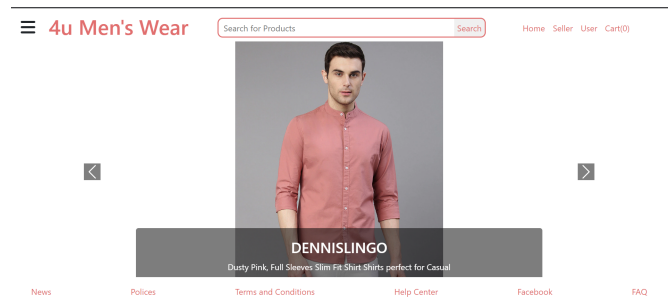
H. Bugs fixing

In the development of this application, we faced some of the bugs. But there is one major bug. Let's discuss how we fixed this bug. So, the bug is that when a user login into the application we checked whether there is any data in the local Cart, and if found we transfer this data from the local Cart to the user Cart. But if any product is already in the user Cart, then we get the duplicate item found after the user login in. So to solve this bug we request current user Cart data before transferring the local Cart data to User Cart and then we filter the data that is already in the User cart from the local Cart, and after applying the filter we transferred the local Cart data to User Cart. Following is the snippet for fixing this bug:

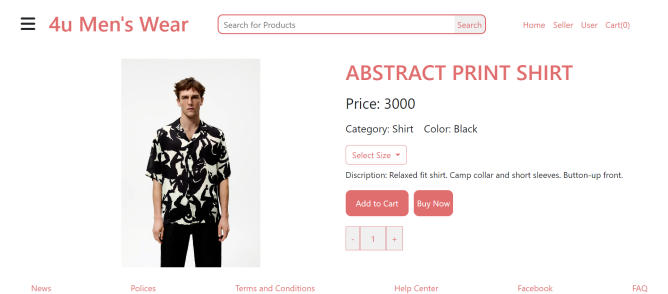
```
localCarttoUserCart() {
  let Data = localStorage.getItem('LocaladdToCart');
  localStorage.removeItem('LocaladdToCart');
  let CartData: product[] = Data && JSON.parse(Data);
  let userData = localStorage.getItem('user');
  let userID = userData && JSON.parse(userData)[0].id;
  if (CartData.length) {
    this.product.getCartlist(userID);
    this.product.cartData.subscribe((result) => {
      const alreadyAdded = new Set(result.map(e => e.productID));
      CartData = CartData.filter((item: product) => !alreadyAdded.has(item.id));
      CartData.forEach((product: product, index) => {
        let Cart: addToCart = {
          ...product,
          productID: product.id,
          userID,
          productSize: product.productSize
        };
        delete Cart.id;
        this.product.UseraddToCart(Cart).subscribe((result) => { });
      });
    });
  }
}
```

VII. SCREENSHOT

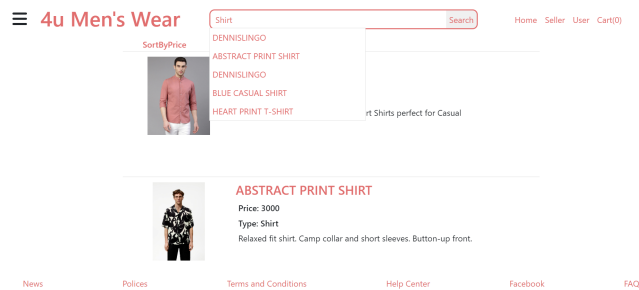
A. Home Page



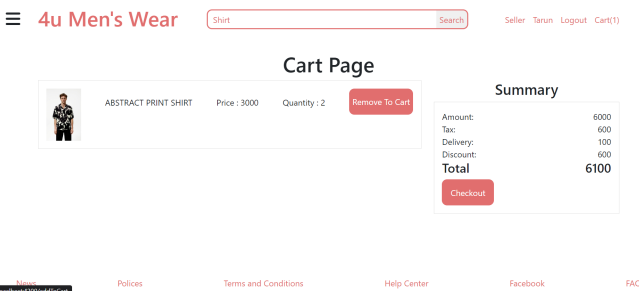
B. Product Details Page



C. Search Page



D. Cart Page



E. Seller Page



ACKNOWLEDGMENT

I would like to acknowledge that the entire work that is required to complete this whole project was done by me and no one is participating in the work to build this project. I would thank my mentor Mr. Jigar Bagdai for his time and efforts provided throughout this Internship. He also gives me some advice and suggestions that were helpful to me during the project's completion.

CONCLUSION

According to Statista Angular is the top-rated web application framework to develop the frontend. Angular is one of the industry's leaders in web construction platforms. It is a framework significantly designed for companies and developers to develop single-page very big complex applications. Angular CLI makes your life easier. We can create more modular applications with Angular easily. Our primary goal for this project is to build a single-page application and Angular has this ability. These are the reason why we used Angular to develop this application. Of course, a JSON server is not a good idea to use but we can push further and make our API faster and more reliable using Angular's HTTP Client and In-Memory-Web-API service.

REFERENCES

- [1] Angular E-Commerce Project: Build a Web App Tutorial. (2022, March 21). Snipcart.
- [2] S. Sachar, Kamini and L. Suneja, Review Paper on Mean Stack for Web Development. International Journal for Scientific Research and Development, vol. 5, no.1, pp.497-498, April 2017.
- [3] A. Lerner, "The complete book on angularjs. Fullstack IO", 2013.
- [4] Chansuwath, W. and Senivongse, T. (2016, June). A model-driven development of web applications using AngularJS framework. 2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS).
- [5] M. Mikowski and J. Powell, Single Page Web Applications: JavaScript End-to-End, Manning Publications Co., 2013.
- [6] [online] Available: <https://angularjs.org/>.
- [7] N. Jain, P. Mangal and D. Mehta, "AngularJS: A modern MVC framework in JavaScript", J. Global Research in Computer Science, vol. 5, no. 12, 2015, pp. 17-23.
- [8] S. Kaewkao and T. Senivongse, "A model-driven development of web-based applications on Google App Engine platform", Proc. 10th National Conf. Computing and Information Technology (NCCIT 2014), pp. 140-145, 8-9 May 2014.
- [9] S. A. Mubin and A. H. Jantan, "A UML 2.0 profile web design framework for modeling complex web application", Proc. 2014 Int. Conf. Information Technology and Multimedia (ICIMU), pp. 324-329, 2014.