

WIPRO CAPSTONE PROJECT

SHOP FOR HOME

C3 - GROUP – G2

Submitted in the partial fulfillment of the
requirements for Wipro certification

JAVA FULL STACK DEVELOPMENT

Submitted by
Dhanuja A
Rongali Jaswanth Kumar
Karthik Shokeen
Suryasathwik P V
Venkata Sai Chekuri

under the guidance of
Mr. Javeed Mohammed Husnuddin



ABSTRACT

Shop for home ordering Platform is a form of household e-commerce that allows consumers to directly buy goods or services from a seller over the internet using a web browser. Consumers find a product of interest by visiting the retailer directly or by searching among alternative vendors using a shopping search engine, which displays the same product availability and pricing at different e-retailers. As of 2022, customers can shop online using a range of different Household appliances. Users can log in to the website to shop the products for home decoration. It has the features of adding to a cart, a Wishlist, and taking orders from home. A typical online store enables the customer to browse the firm's range of products and services.

TABLE OF CONTENTS

CHAPTER-1: INTRODUCTION

1.1 OBJECTIVE

1.2 USER REQUIREMENTS

1.3 ADMIN REQUIREMENTS

CHAPTER-2: TECHNOLOGIES REQUIRED

2.1 SUPPORTED OPERATING SYSTEM

2.2 FEATURES OF THE APPLICATION

CHAPTER-3: SOFTWARE USED

3.1 FRONTEND

3.2 BACKEND

3.3 DATABASE

CHAPTER-4: RESULTS

CHAPTER-5: CONCLUSION

END

CHAPTER-1

INTRODUCTION

This project Shop for Home has been developed in Angular, Java Spring Boot, and MySQL. The Shop for the Home project is an application based on managing the Home-decor and selling household items online. The main purpose of developing this project in Angular and Java Spring Boot is to manage all the details about home decor items, categories, companies, orders, sales, etc. There are two categories of users available in this project. The first one is the admin and the second one is the user. Admin can add home-décor item categories and can also manage the sales details. This Angular project is very helpful for maintaining sales activity in household items. In this project, customers can see the details of home-decor item categories, etc. Only Admin can edit or delete the details of the Home-decor item.

1.1 OBJECTIVE:

The main purpose of this module is to provide all the functionality related to customers. It tracks all the information and details of the customer. We have developed all Categories of CRUD (Create, Read, Update and Delete) operations for the customers. This is a role-based module where the admin can perform every operation on data but the customer will be able to view only his/her data, so access level restrictions have also been implemented on the project. We also provide customized Angular and Java Spring Boot" Projects for beginners.

1.2 User-Requirements:

1. As a user I should be able to log in, Logout, and Register into the application.
2. As a user I should be able to see the products in different categories.
3. As a user I should be able to sort the products.
4. As a user I should be able to add the products to the shopping cart.
5. As a user I should be able to increase or decrease the quantity added to the cart.
6. As a user I should be able to add “n” number of products in the cart.
7. As a user I should be able to get the Wishlist option.
8. As a user I should get different discount coupons.

1.3 Admin-Requirements:

1. As an Admin I should be able to log in, Logout, and Register into the application.
2. As an Admin I should be able to perform CRUD on Users.
3. As an Admin I should be able to Perform CRUD on the products.
4. As an Admin I should be able to get a bulk upload option to upload a CSV for products details.
5. As an Admin I should be able to get the stocks.
6. As an Admin I should be able to mail if any stock is less than 10.
7. As an Admin I should be able to get the sales report of a specific duration.
8. As an Admin I should be able to set users.

CHAPTER-2

TECHNOLOGIES REQUIRED

We have developed this project using the below technology

- **HTML:** The page layout has been designed in HTML.
- **CSS:** CSS has been used for all the designing part.
- **BOOTSTRAP:** It has been used for designing part.
- **JavaScript:** All the validation task and animations has been developed by JavaScript.
- **Java Spring Boot:** All the business and backend API logic has been implemented in Java Spring Boot.
- **.SQL:** SQL files have been used as a database for the project.
- **Angular:** All the frontend logic has been implemented over Angular and we used angular CLI for it.
- **Visual Studio Code-(VSS):** For Angular IDE, we have used Visual Studio Code.
- **STS:** We have used STS (Spring Tool Suite) for developing all spring boot APIs.
- **Tomcat:** Project will be run over the Tomcat server

2.1 Supported Operating System

We can configure this project on the following operating system.

- **Windows:** This project can easily be configured on the windows operating system.

2.2 Features of the application:

1. Registration
2. Login
3. Searching
4. Filtering
5. Sorting
6. Dynamic data
7. Responsive and compatible with different devices

CHAPTER-3

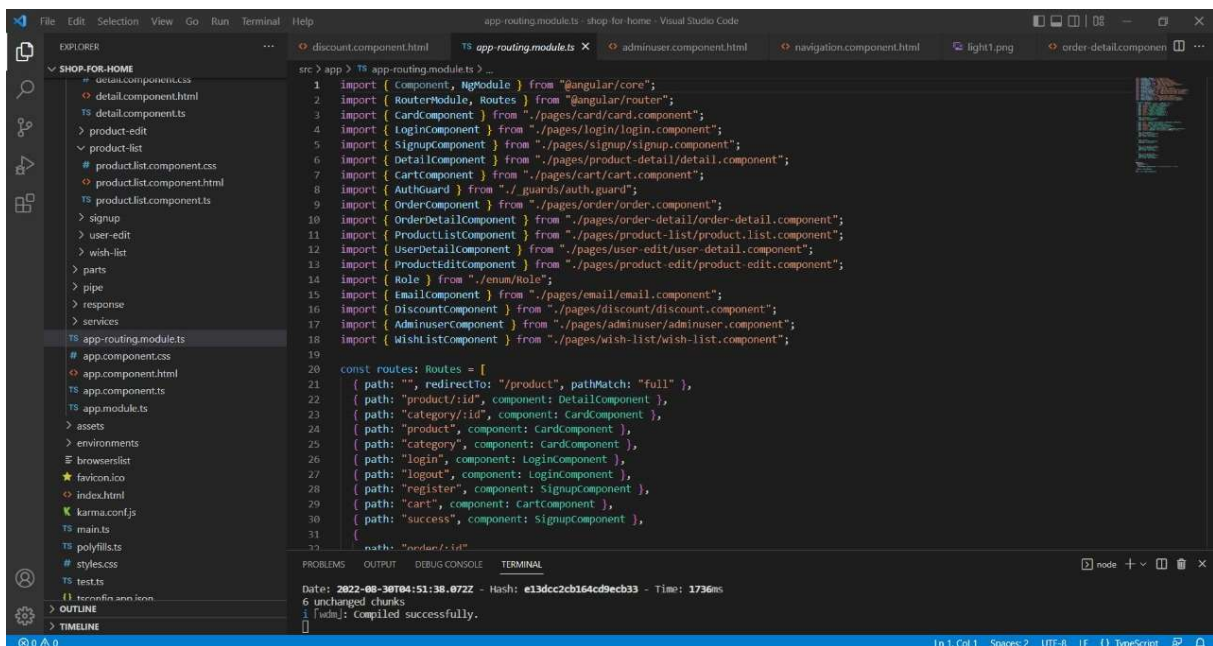
SOFTWARE USED

3.1 Front End

From your local FrontEnd code path -> open cmd

Eg -> {local path}\ecommerce-eshop\frontend

- Run this command -- code.
- Run Npm install from vs code terminal
- Run npm start
- After the successful compiling you got this link in terminal localhost/4200.
- Open this link in google chrome/Microsoft Edge.



The screenshot shows the Visual Studio Code interface with the 'app-routing.module.ts' file open. The file contains the following code:

```
1 import { Component, NgModule } from '@angular/core';
2 import { RouterModule, Routes } from '@angular/router';
3 import { CardComponent } from './pages/card/card.component';
4 import { LoginComponent } from './pages/login/login.component';
5 import { SignupComponent } from './pages/signup/signup.component';
6 import { DetailComponent } from './pages/product-detail/detail.component';
7 import { CartComponent } from './pages/cart/cart.component';
8 import { AuthGuard } from './guards/auth.guard';
9 import { OrderComponent } from './pages/order/order.component';
10 import { OrderDetailComponent } from './pages/order-detail/order-detail.component';
11 import { ProductListComponent } from './pages/product-list/product-list.component';
12 import { UserDetailComponent } from './pages/user-edit/user-detail.component';
13 import { ProductEditComponent } from './pages/product-edit/product-edit.component';
14 import { Role } from './enum/Role';
15 import { EmailComponent } from './pages/email/email.component';
16 import { DiscountComponent } from './pages/discount/discount.component';
17 import { AdminuserComponent } from './pages/adminuser/adminuser.component';
18 import { WishListComponent } from './pages/wish-list/wish-list.component';
19
20 const routes: Routes = [
21   { path: '', redirectTo: '/product', pathMatch: 'full' },
22   { path: 'product/:id', component: DetailComponent },
23   { path: 'category/:id', component: CardComponent },
24   { path: 'product', component: CardComponent },
25   { path: 'category', component: CardComponent },
26   { path: 'login', component: LoginComponent },
27   { path: 'logout', component: LoginComponent },
28   { path: 'register', component: SignupComponent },
29   { path: 'cart', component: CartComponent },
30   { path: 'success', component: SignupComponent },
31   {
32     path: 'admin/:id',
33     component: AdminuserComponent,
34     canActivate: [AuthGuard],
35   },
36 ];
37
38 @NgModule({
39   declarations: [],
40   imports: [RouterModule.forRoot(routes)],
41   exports: [RouterModule],
42 })
43 export class AppRoutingModule {}
```

The terminal output shows the following message:

```
Date: 2022-08-30T04:51:38.072Z - Hash: e13dce2cb164cd9ecb33 - Time: 1736ms
6 unchanged chunks
i [vite] Compiled successfully.
```


3.2 Back End

Spring Tool Suite (STS) is a java IDE tailored for developing Spring-based enterprise applications. It is easier, faster, and more convenient. And most importantly it is based on Eclipse IDE. STS is free, open-source, and powered by VMware.

Project Explorer:

```
> shop-for-home [boot] [devtools]
> shop-for-home-discount-service [boot] [devtools]
> shop-for-home-server [boot]
```

1) Shop-for-home

The Shop-for-home spring project packages and all files are shown in below:

```
▼ shop-for-home [boot] [devtools]
  ▼ src/main/resources
    application.properties
  > JRE System Library [JavaSE-11]
  > Maven Dependencies
  > target/generated-sources/annotations
  ▼ java
    > com.wipro.springboot
    > com.wipro.springboot.controller
    > com.wipro.springboot.entity
    > com.wipro.springboot.enums
    > com.wipro.springboot.exception
    > com.wipro.springboot.form
    > com.wipro.springboot.repository
    > com.wipro.springboot.security
    > com.wipro.springboot.security.JWT
    > com.wipro.springboot.service
    > com.wipro.springboot.service.impl
    > com.wipro.springboot.vo.helper
    > com.wipro.springboot.vo.request
    > com.wipro.springboot.vo.response
  > src/test/java
  > target/generated-test-sources/test-annotations
  > src
  > target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

Spring Boot Annotations Used for Each package:

i) com.wipro.springboot

This is the main package that we need to run the class by the spring boot app.

```
└─ com.wipro.springboot
   └─ ShopForHomeApplication.java
```

- **@SpringBootApplication**

The @SpringBootApplication annotation is a combination of the following three Spring annotations and provides the functionality of all three with just one line of code. This annotation marks a class as a Configuration class for Java-based configuration.

- **@EnableEurekaClient**

@EnableEurekaClient makes the app into both a Eureka "instance" (i.e. it registers itself) and a "client" (i.e. it can query the registry to locate other services).

ii) com.wipro.springboot.controller

```
└─ com.wipro.springboot.controller
   ├── CartController.java
   ├── CSVController.java
   ├── EmailController.java
   ├── OrderController.java
   ├── ProductCategoryController.java
   ├── ProductController.java
   ├── UserController.java
   └── WishListController.java
```

- **@CrossOrigin**

@CrossOrigin annotation only allows cross-origin HTTP requests from a single origin. We can take a less restrictive approach and specify multiple origins, on a per-use-case need. UserController.java

- **@Controller**

The @Controller annotation is a specialization of the generic stereotype @Component annotation, which allows a class to be recognized as a Spring-managed component. The @Controller annotation extends the use case of @Component and marks the annotated class as a business or presentation layer.

- **@RestController**

The `@RestController` annotation in a Spring application to build a Restful controller. Spring is a popular Java application framework and Spring Boot is an evolution of Spring that helps create stand-alone, production-grade Spring-based applications easily.

- **@RequestMapping**

The process of mapping web requests to handler methods is also called routing. `@RequestMapping` has the following specializations: The annotation can be used both at the class and at the method level.

- **@GetMapping**

`@GetMapping` annotation maps HTTP GET requests onto specific handler methods. It is a composed annotation that acts as a shortcut for `@RequestMapping (method = RequestMethod.GET)`. The following application uses `@GetMapping` to map two request paths onto handler methods.

- **@PutMapping**

Annotation for mapping HTTP PUT requests onto specific handler methods. Specifically, `@PutMapping` is a composed annotation that acts as a shortcut for `@RequestMapping (method = RequestMethod.PUT)`. consumes – Narrows the primary mapping by media types that can be consumed by the mapped handler.

- **@PostMapping**

The POST HTTP method is used to create a resource and `@PostMapping` annotation for mapping HTTP POST requests onto specific handler methods. Specifically, `@PostMapping` is a composed annotation that acts as a shortcut for `@RequestMapping (method = RequestMethod.POST)`.

- **@DeleteMapping**

It is a composed annotation that acts as a shortcut for `@RequestMapping (method = RequestMethod.DELETE)`. The following application uses `@DeleteMapping` to delete a resource. We use annotations to set up a Spring web application.

This Package is act as hibernate which helps to store data in the database

```

  ▾ com.wipro.springboot.entity
    > Cart.java
    > Email.java
    > Order.java
    > Product.java
    > ProductCategory.java
    > ProductInOrder.java
    > User.java
    > WishList.java

```

- **@Entity**

This annotation specifies that the class is an entity. This annotation can be applied to Class, Interface of Enums. Let's create a Spring boot project from the scratch and demonstrates the usage of @Entity annotation. We will use Spring Data JPA to develop a repository layer and MySQL database at the backend.

- **@Id**

The @Id annotation is inherited from javax.persistence.Id , indicating the member field below is the primary key of current entity. Hence your Hibernate and spring framework as well as you can do some reflect works based on this annotation. for details please check javadoc for Id

- **@GeneratedValue**

The @GeneratedValue annotation specifies how to generate values for the given column. This annotation will help in creating primary keys values according to the specified strategy. The only thing we need to do is to add @GeneratedValue annotation in the POJO class.

iv) **com.wipro.springboot.repository:**

```

  ▾ com.wipro.springboot.repository
    > ICartRepository.java
    > IOrderRepository.java
    > IProductCategoryRepository.java
    > IProductInOrderRepository.java
    > IProductRepository.java
    > IUserRepository.java
    > IWishListRepository.java
    > WishListCustomRepository.java

```

- **@Repository**

Spring is a popular Java application framework and Spring Boot is an evolution of Spring that helps create stand-alone, production-grade Spring-based applications easily. @Repository is a Spring annotation that indicates that the decorated class is a repository.

v) **com.wipro.springboot.service**

This package consists of several interfaces for each service of the spring boot application project

```

  ▾ com.wipro.springboot.service
    > ICartService.java
    > IOrderService.java
    > IProductCategoryService.java
    > IProductInOrderService.java
    > IProductService.java
    > IUserService.java

```

vi) **com.wipro.springboot.service.impl:**

This package is consists the various services for the each controller.

```

  ▾ com.wipro.springboot.service.impl
    > CartServiceImpl.java
    > CSVServiceImpl.java
    > EmailServiceImpl.java
    > OrderServiceImpl.java
    > ProductCategoryServiceImpl.java
    > ProductInOrderServiceImpl.java
    > ProductServiceImpl.java
    > UserServiceImpl.java
    > WishListServiceImpl.java

```

- **@Service**

Spring Service annotation can be applied only to classes. It is used to mark the class as a service provider. Spring @Service annotation is used with classes that provide some business functionalities. Spring context will autodetect these classes when annotation-based configuration and classpath scanning is used.

- **@AutoWired**

The @Autowired annotation provides more fine-grained control over where and how autowiring should be accomplished. The @Autowired annotation can be used to autowire bean on the setter method just like @Required annotation, constructor, a property or methods with arbitrary names and/or multiple arguments.

- **@Bean**

The `@Bean` annotation which is applied on a method to specify that it returns a bean to be managed by the Spring context. Spring Bean annotation is usually declared in Configuration classes methods. This annotation is also a part of the spring core framework.

- **`@Configuration`**

Spring `@Configuration` annotation is part of the spring core framework. Spring Configuration annotation indicates that the class has `@Bean` definition methods. So Spring container can process the class and generate Spring Beans to be used in the application.

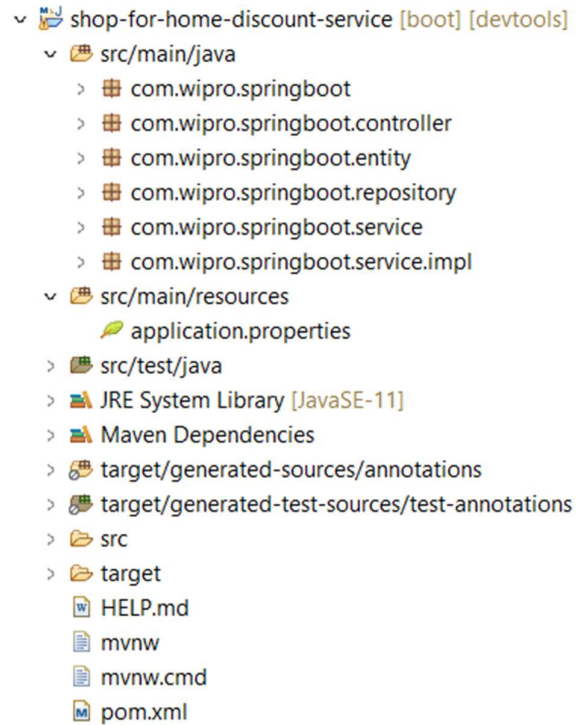
- **`@Transactional`**

`@Transactional` Spring makes sure all the required data is safe until the Transaction ends. Once it ends, the delete request from the partner would take place, and then you'll have some logic to remove it from the new Person object. But that would take place afterward. You use `@Transactional` to ensure safety on your "Transactions".

Application.properties

```
1 server.port=8081
2 eureka.instance.hostname=localhost
3 spring.application.name=shop-for-home
4
5 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
6 spring.datasource.url=jdbc:mysql://localhost:3306/ecommerce
7 spring.datasource.username=root
8 spring.datasource.password=Dpja@831
9
10 spring.mail.default-encoding=UTF-8
11 spring.mail.protocol=smtp
12 spring.mail.test-connection=false
13 spring.mail.host=smtp.gmail.com
14 spring.mail.username=t31093931@gmail.com
15 spring.mail.password=123ABCabc
16 spring.mail.port=587
17 spring.mail.properties.mail.smtp.auth=true
18 spring.mail.properties.mail.smtp.starttls.enable=true
19
20 spring.jpa.show-sql=false
21 spring.jpa.hibernate.ddl-auto=update
22 spring.jpa.database=mysql
23 spring.jpa.properties.hibernate.temp.use_jdbc_metadata_defaults= false;
24 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
25
26 spring.queries.users-query=select email, password, active from users where email=?
27 spring.queries.roles-query=select email, role from users where email=?
28
29 spring.sql.init.platform=mysql
30 spring.sql.init.mode=always
31 spring.sql.init.continue-on-error=true
32
33 server.servlet.contextPath=/api
34
35 jwtSecret=me.shop
36 jwtExpiration=86400
```

2) Shop-for-home-discount-service:



Application.properties :

```
1 server.port=8082
2 eureka.instance.hostname=localhost
3 spring.application.name=shop-for-home-discount-service
4
5 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
6 spring.datasource.url=jdbc:mysql://localhost:3306/ecommerce
7 spring.datasource.username=root
8 spring.datasource.password=Dpja@831
9
10 spring.jpa.show-sql=false
11 spring.jpa.hibernate.ddl-auto=none
12 spring.jpa.database=mysql
13 spring.jpa.properties.hibernate.temp.use_jdbc_metadata_defaults= false;
14 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
15
16 spring.sql.init.platform=mysql
17 spring.sql.init.mode=always
18 spring.sql.init.continue-on-error=true
19
20 server.servlet.contextPath=/api
21
```


- ▼ shop-for-home-server [boot]
 - ▼ src/main/java
 - ▼ com.wipro.springboot
 - > ShopForHomeServerApplication.java
 - ▼ src/main/resources
 - application.properties
 - > src/test/java
 - > JRE System Library [JavaSE-11]
 - > Maven Dependencies
 - > target/generated-sources/annotations
 - > target/generated-test-sources/test-annotations
 - > src
 - > target
 - HELP.md
 - mvnw
 - mvnw.cmd
 - pom.xml

Com.wipro.sprinboot:

```

1 package com.wipro.springboot;
2
3 import org.springframework.boot.SpringApplication;
4
5
6
7 @SpringBootApplication
8 @EnableEurekaServer
9 public class ShopForHomeServerApplication {
10
11     public static void main(String[] args) {
12         SpringApplication.run(ShopForHomeServerApplication.class, args);
13     }
14 }
15

```

Application.properties:

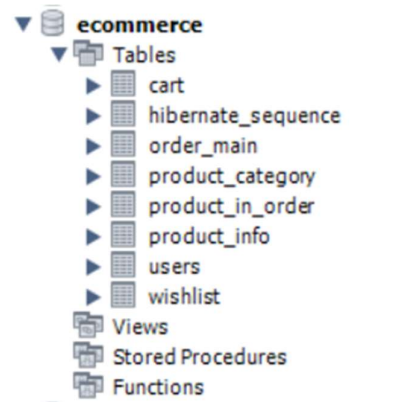
```

1 eureka.instance.hostname=localhost
2 eureka.client.register-with-eureka=false
3 eureka.client.fetch-registry=false
4 server.port=8761
5 spring.application.name=sop-for-home-server
6
7

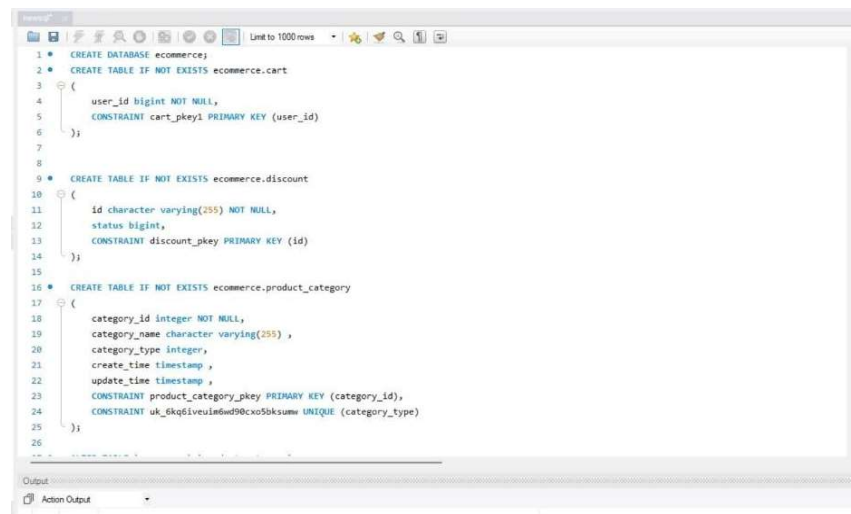
```


3.3 Database

- Install MySQL
- After installation search the MySQL command line in your computer
- Open that command line



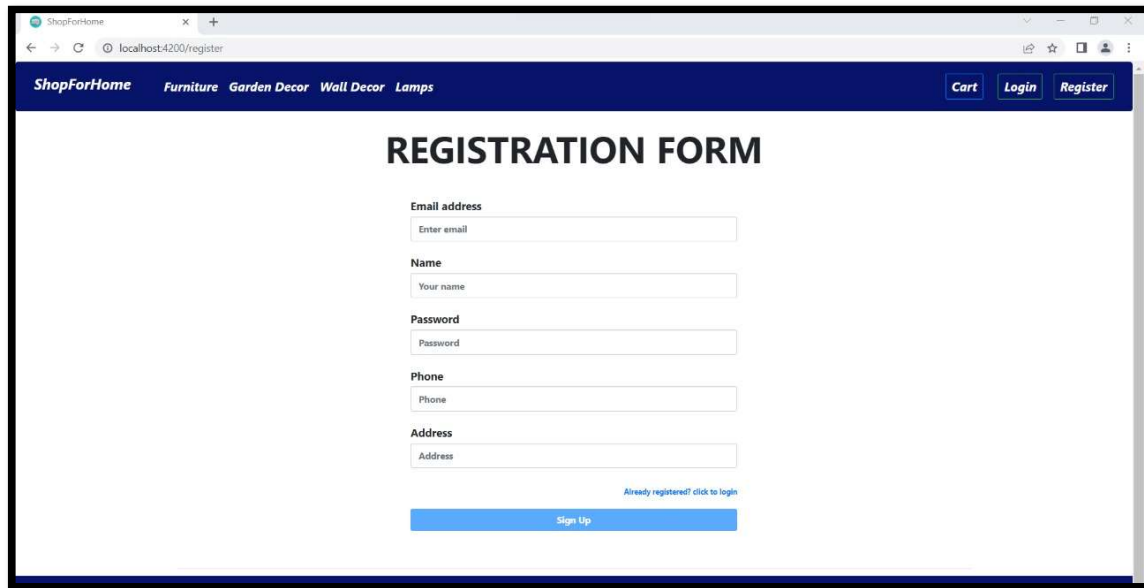
➤ MySQL Workbench



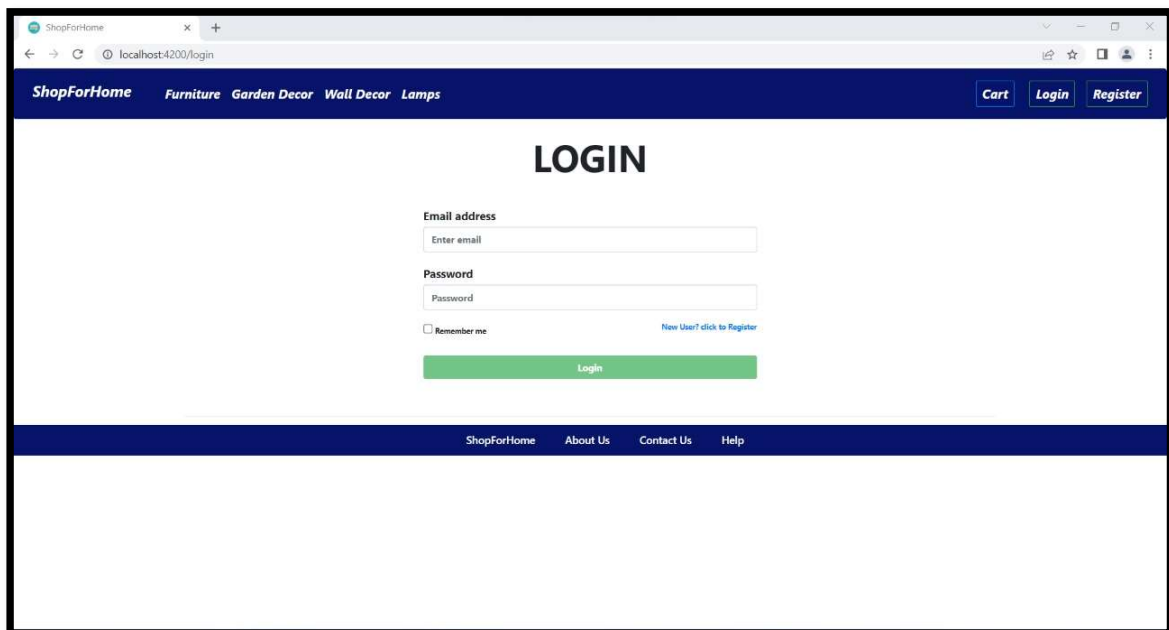
CHAPTER-4

RESULTS

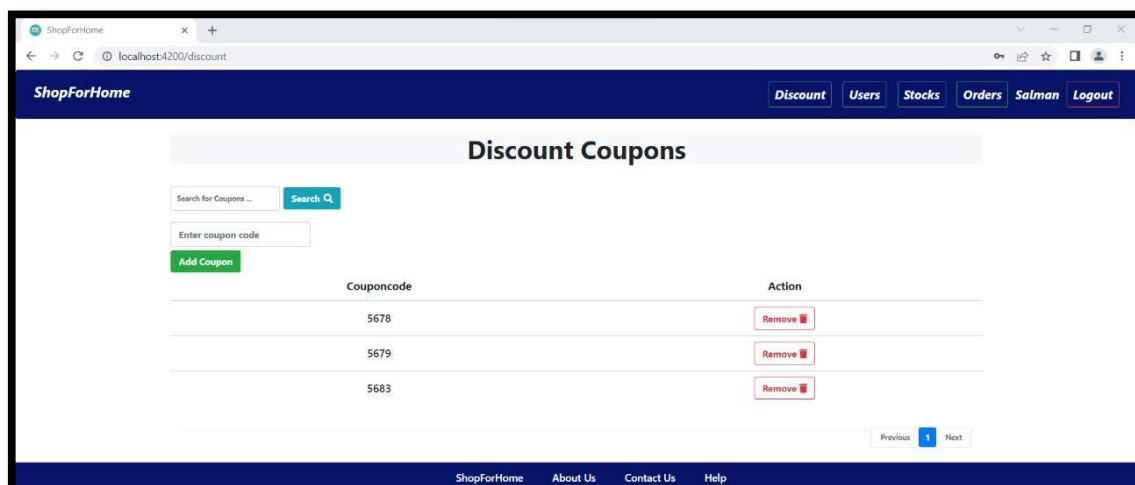
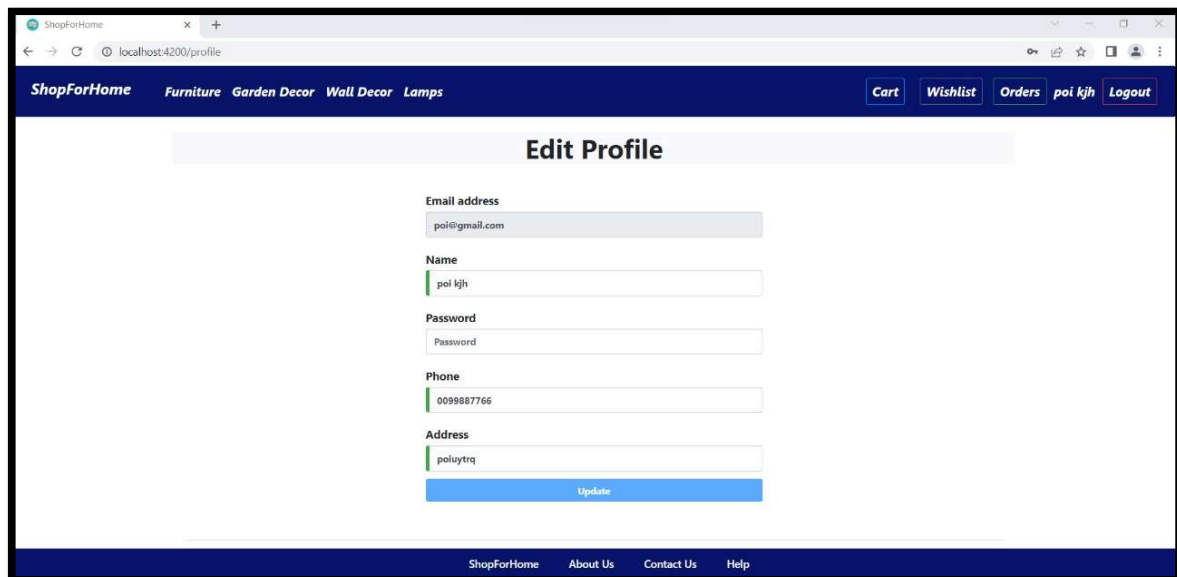
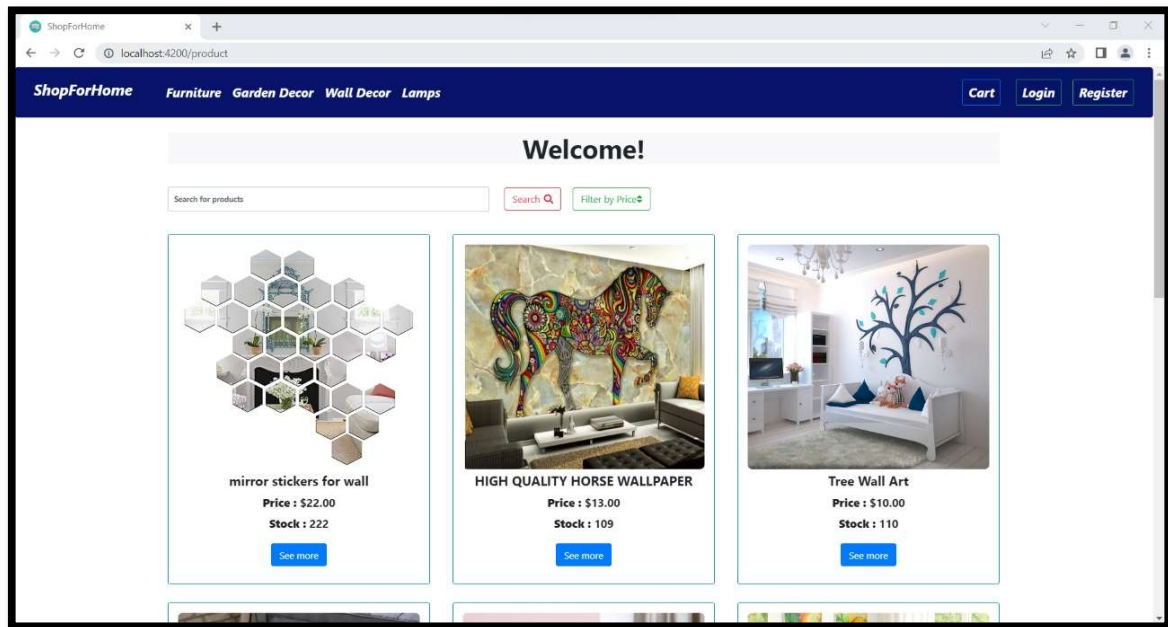
The output of the project is shown in the name of the output images.



A screenshot of a web browser displaying the 'ShopForHome' registration page. The browser's address bar shows 'localhost:4200/register'. The page has a dark blue header with the site name 'ShopForHome' and navigation links for 'Furniture', 'Garden Decor', 'Wall Decor', and 'Lamps'. On the right side of the header are buttons for 'Cart', 'Login', and 'Register'. The main content area is titled 'REGISTRATION FORM' in bold. It contains several input fields: 'Email address' (with placeholder 'Enter email'), 'Name' (with placeholder 'Your name'), 'Password' (with placeholder 'Password'), 'Phone' (with placeholder 'Phone'), and 'Address' (with placeholder 'Address'). Below these fields is a blue 'Sign Up' button. A link 'Already registered? click to login' is positioned above the button.



A screenshot of a web browser displaying the 'ShopForHome' login page. The browser's address bar shows 'localhost:4200/login'. The page has a dark blue header with the site name 'ShopForHome' and navigation links for 'Furniture', 'Garden Decor', 'Wall Decor', and 'Lamps'. On the right side of the header are buttons for 'Cart', 'Login', and 'Register'. The main content area is titled 'LOGIN' in bold. It contains input fields for 'Email address' (with placeholder 'Enter email') and 'Password' (with placeholder 'Password'). Below these fields is a 'Remember me' checkbox and a green 'Login' button. A link 'New User? click to Register' is positioned to the right of the 'Remember me' checkbox. At the bottom of the page, there is a dark blue footer with links for 'ShopForHome', 'About Us', 'Contact Us', and 'Help'.



+

ShopForHome

localhost:4200/order?page=3&size=10

ShopForHome

Discount Users Stocks Orders Salman Logout

Orders

Order #	Customer Name	Customer Email	Customer Phone	Shipping Address	Total	Order Date	Status	Action
51	Suraj Singh	surajsingh@gmail.com	8877665544	7 Akbar road , New Delhi	\$6.00	Aug 28, 2022	Cancelled	Show
48	Isha	isha@gmail.com	9999999999	delhi	\$12.00	Aug 28, 2022	Cancelled	Show
42	Suraj Singh	surajsingh@gmail.com	8877665544	7 Akbar road , New Delhi	\$104.00	Aug 28, 2022	Cancelled	Show
34	surya	sathwik@gmail.com	9999955555	delhi	\$2.00	Aug 27, 2022	Cancelled	Show
30	Suraj Singh	surajsingh@gmail.com	8877665544	7 Akbar road , New Delhi	\$16.00	Aug 27, 2022	Cancelled	Show
25	Jaswant	jaswant@gmail.com	9999955555	delhi	\$35.00	Aug 27, 2022	Cancelled	Show






ShopForHome

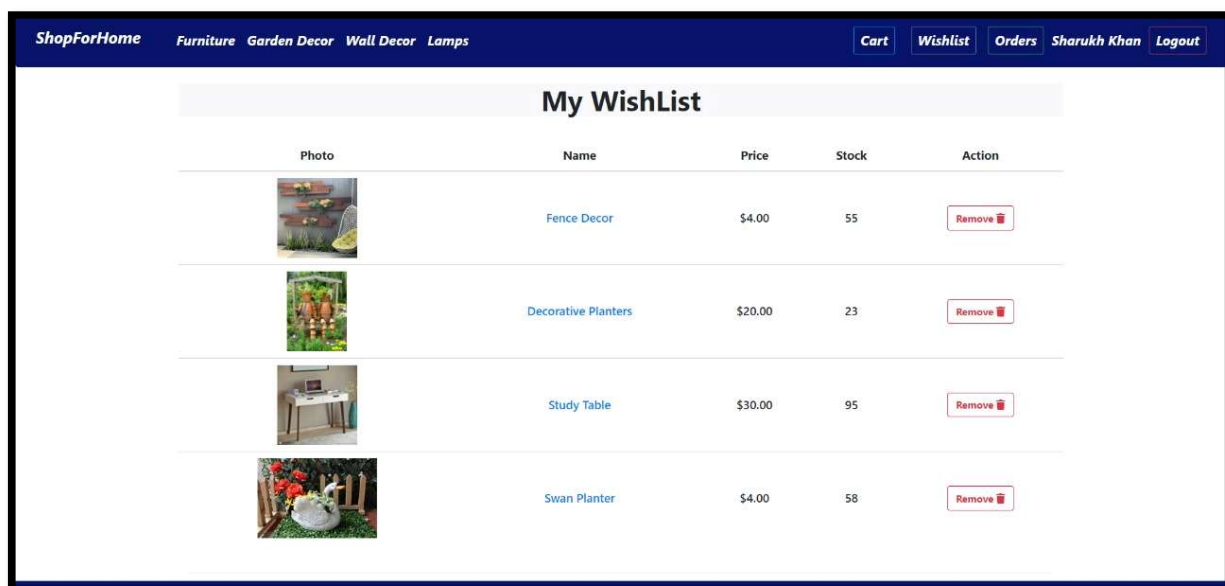
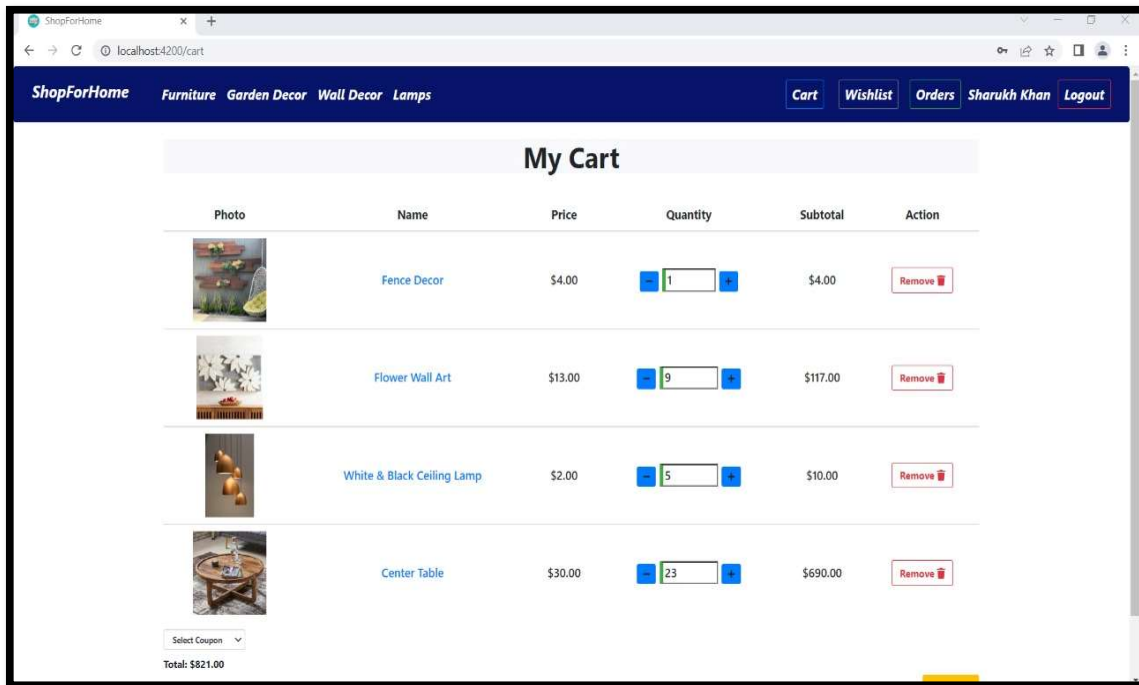
localhost:4200/seller/product

Choose File No file chosen

Upload Products

Search for products ... Search

Photo	Code	Name	Type	Description	Price	Stock	Status	Action	Mail
	B0004	Collapsible Wardrobe	Furnitures	Made from selected Non Woven cloth cover, high quality Galvanized iron pipe and PP Plastic Connectors, this portable storage closet will meet your long term storage needs.	\$30.00	1	Unavailable	Edit Remove	Email
	B0005	Wooden Wall Shelves	Furnitures	This 3 Tier Shelf is made up of Laminated Engineered Wood (MDF) which is dust or any stain and spot from the shelf and also this laminated shelf gives a stylish	\$30.00	199	Available	Edit Remove	Email
	B0011	Study Table	Furnitures	Computer Table with Drawers	\$30.00	95	Available	Edit Remove	Email
	B0012	Dining Table	Furnitures	Wooden 4 Seater Dining Table Set	\$20.00	195	Available	Edit Remove	Email
	B0013	Fabric Chair	Furnitures	Colourful Fabric Chair	\$10.00	0	Unavailable	Edit Remove	Email



CHAPTER-6

CONCLUSION

By performing or following the above-mentioned methodology we can conclude that

- We can create a web page using Angular, Spring boot, Node-JS, MySQL, etc.
- In this project, we have created the page with the name “**SHOP FOR HOME**” in which a user can buy the household items like decorating things and many others.
- We need updated or latest version software for the smooth working of the project.
- Both frontend and backend need to run frequently to the page with user interface flexibility.

In the end, we can say that the web page is created by following the path regulations and showing the output as required.