CAPSTONE PROJECT REPORT

Shop For Home (ECommerce Website)

Submitted by

Ayush Verma	ayushx16@gmail.com	Group 4 C2 batch
Mayank Parihar	pariharmayank50@gmail.com	
Indrajeet Kumar Mahato	10indrajeet02@gmail.com	
Aman Grover	gaaman56@gmail.com	
T Narasimhulu	narasimhuluptn@gmail.com	

Under the Guidance of

Ashutosh Trivedi
(Training mentor Great Learning)

ACKNOWLEDGEMENT

of various project documents made sure the project was in the right direction. We would also like to thank our **Wipro & Great Learning** for providing such opportunity and their constant efforts to keep our courses going despite all the hardships that the pandemic brought upon us.
Finally, we would also like to thank our family, friends and seniors for their constant support, trust and valuable remarks that directly or indirectly made the project a whole lot better and helped us reach where we are.

TABLE OF CONTENTS

INTRODUCTION

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

Problem Description

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 Category of CRUD (Create, Read, Update and Delete) operations of the customers. This is a role-based module where admin can perform each and every operation on data but the customer will be able to view only his/her data, so access level restrictions has also been implemented on the project. We also provide customized Angular and Java Spring Boot" Projects for beginners.

User-Requirements:

1. As a user I should be able to login, 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 into the shopping cart.

5. As a user I should be able to increase or decrease the quantity added in 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 where I can add those products which I want but don't want to order now

8. As a user I should get different discount coupon

Admin-Requirements:

1. As an Admin I should be able to login, 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 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 of users

TECHNOLOGIES REQUIRED

We have developed this project using the below technology

- **HTML**: Page layout has been designed in HTML
- **CSS**: CSS has been used for all the 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 has been used as database for the project
- **Angular:** All the frontend logic has been implemented over the 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 API's
- **Tomcat**: Project will be run over the Tomcat server
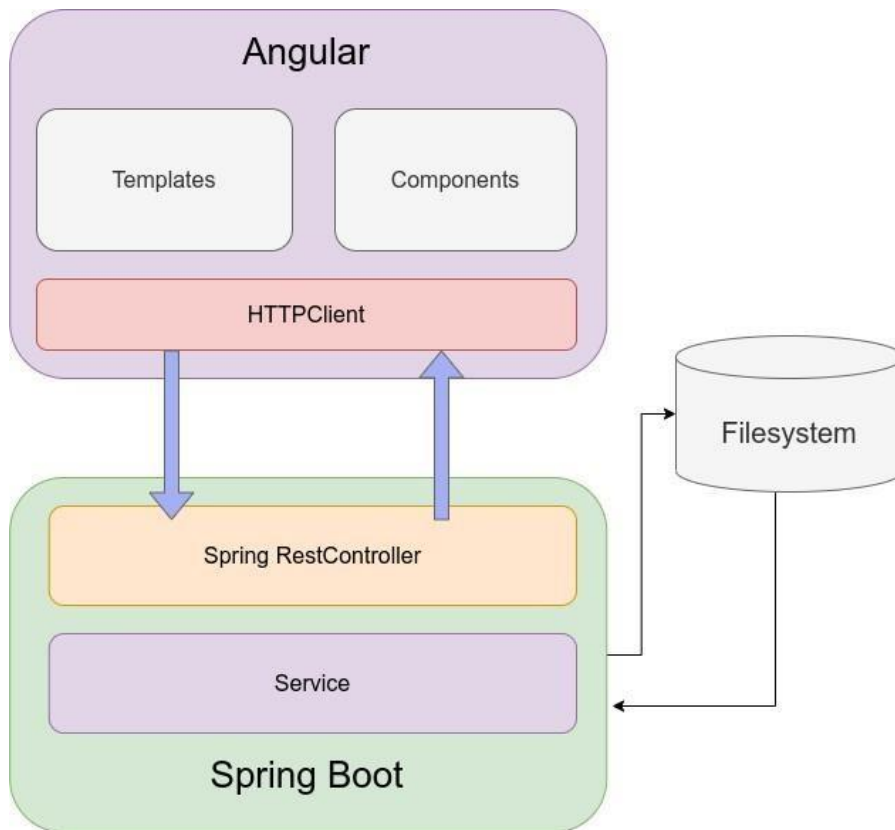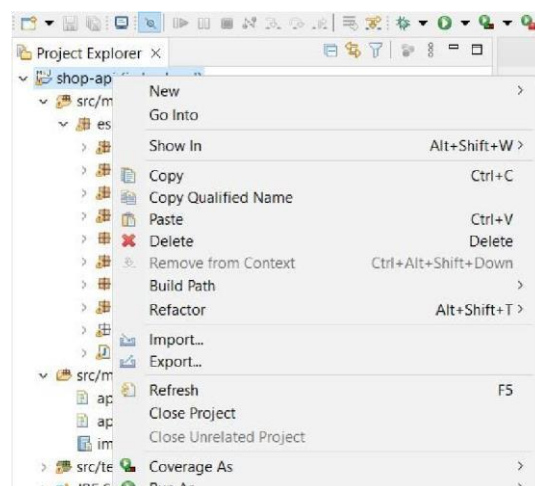
Supported Operating System

We can configure this project on following operating system.

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

INSTALLATION AND INSTRUCTIONS

Open the spring suit tool 4 or Eclipse Ide

Then import the project

Do Project clean and Build
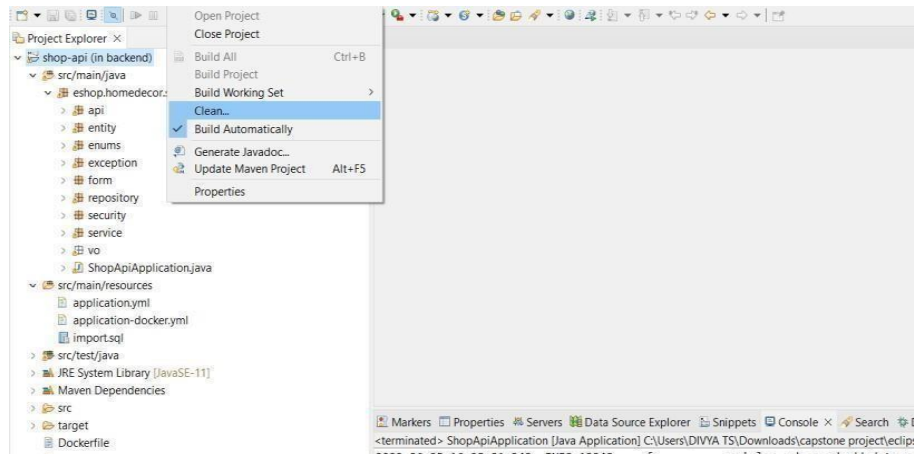
Then do maven clean
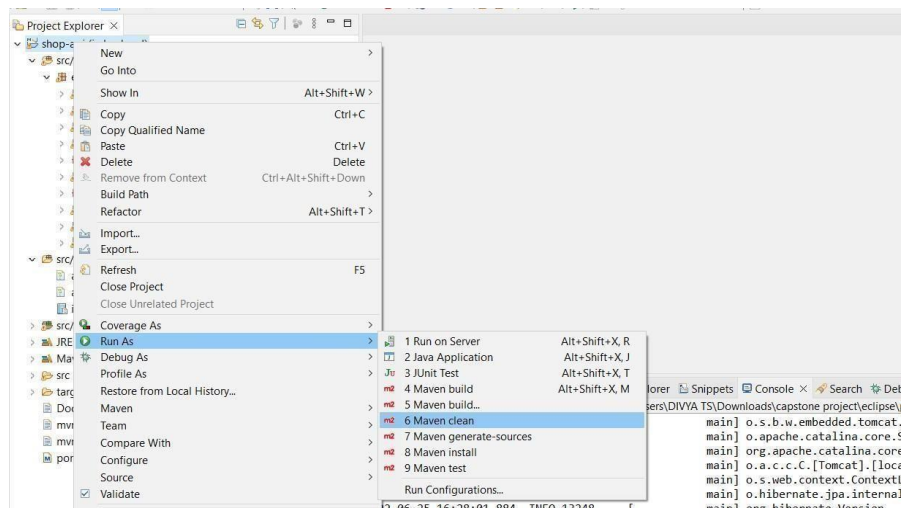
Maven install
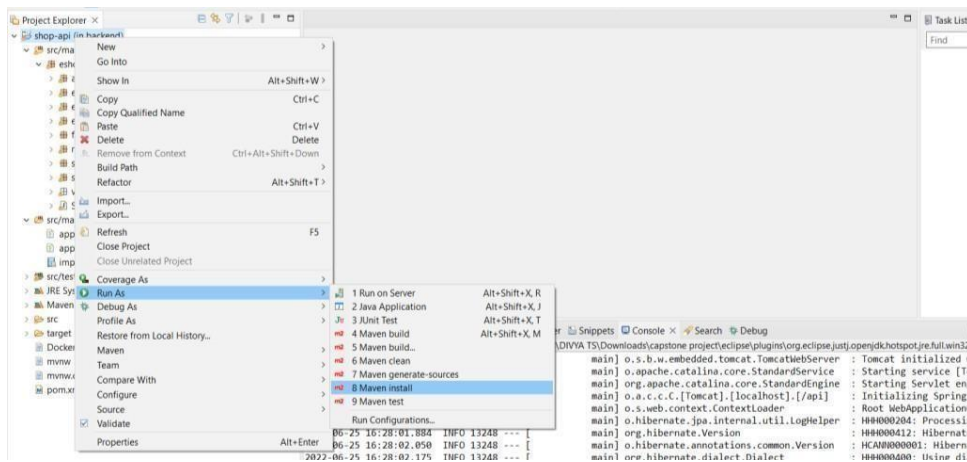
Maven Update
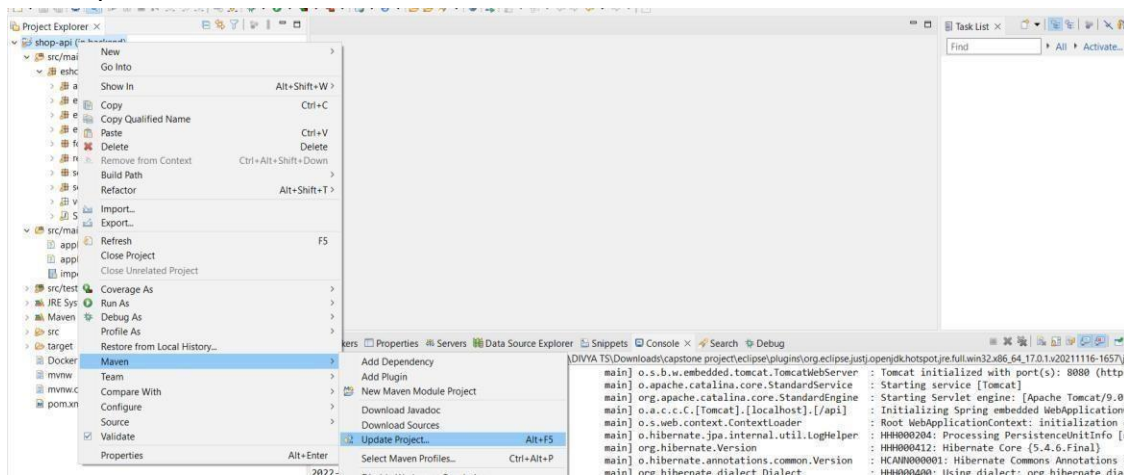
Now create Database and add required tables

- Install postgres SQL
- After installation serach PgAdmin in your computer
- Open that PgAdmin and create the database as ecommerce or any other but make sure to declare the same database name in application file

- Open this query editor and You can run the query from databse.sql file. Provided along with report and ppt.
- Run the query.

Open Visual Studio Code

Files -> open project -> locate the directory where you extracted the project and select the frontend folder

- Open terminal
- Then type **npm install**
- After that **ng serve**
- After the successful compiling you got this link in terminal http://localhost:4200/ **
- Open this link in any browser

Source Code

ANGULAR Front End

Creating Angular Part **What is Angular?**

Angular is a web application framework based on TypeScript. It's open-source and developed by Google.

That said, there have been some new developments in the Angular world that are worth a quick mention. Angular's latest major release is 12, and it boasts loads of bug fixes making it smaller, faster & easier to use. Plus, it now supports TypeScript 4.2.3 and up. It's also starting to drop support for building libraries with View Engine in favour of Angular's compiler/runtime.

Component file in Angular

The Angular component allows us to provide a way for styling the component. This means that we can provide different CSS styling, rules, and other device-specific style configuration for a specific component. For that, the Angular component has metadata properties based on your different needs and requirements.

Module File in Angular

Module in Angular refers to a place where you can group the components, directives, pipes, and services, which are related to the application. In case you are developing a website, the header, footer, left, centre and the right section become part of a module.

Service File in Angular

Services in Angular are simply typescript classes with the @injectible decorator. This decorator tells angular that the class is a service and can be injected into components that need that service. They can also inject other services as dependencies. As mentioned earlier, these services are used to share a single piece of code across multiple components. These services are used to hold business logic. Services are used to interact with the backend. For example, if you wish to make AJAX calls, you can have the methods to those calls in the service and use it as a dependency in files.

There are 2 users in this application: 1.
User

2.Admin

USER   SIGN-UP

Firstly, the user needs to create an account (or) sign-in in to the application

Events- ngoninit () - It will execute if you refresh your browser or first initialize a component but not when other events occur

LOG-IN

 Events -   ngoninit () - It will execute if you refresh your browser or first initialize a component but not when other events occur

onSubmit () - Submit directive specifies a function to run when the form is submitted

CART

Cart is a page where users can pile up what they want to buy from the website and then simply checkout by paying online

```
import    {    BrowserModule    }    from
'@angular/platformbrowser';  import  {  NgModule  }
from '@angular/core';

import { AppComponent } from './app.component';
```

Events

Events

ngOnInit() - It will execute if you refresh your browser or first initialize a component but not when other events occur ngAfterContentChecked() - It is used to check whether the product is checked or not ▪ Selected() - It is used to select the products into cart ngOnDestroy() - gets called when a component is about to be destroyed.

getCoupon() - It is used to select a coupon

remove() - The remove is used to clear all the products in a cart section

onchange() - It is used to change the product checkout() - It is used to check out from product section to payment section minusOne() - It is used to decrease the quantity of the products in a cart addOne() - It is used to increase the quantity of a product in a cart

Order Details means the details relating to the Order, including without limitation, the description of Products, details of the Seller and the Buyer, date of order placement, total amount payable by the Buyer, delivery date, mode of payment, unique order number (AWB number) etc

Events ngOnInit () - It will execute if you refresh your browser or first initialize a component but not when other events occur. Major part of the development with Angular is done in the components. Components are basically classes that interact with the .html file of the component, which gets displayed on the browser.

The file structure has the app component and it consists of the following files

app.component.css

app.component.html app.component.spec.ts

app.component.spec.ts app.module.ts

The above files were created by default when we created new project using the angular-cli command.

If you open up the **app.module.ts** file, it has some libraries which are imported and also a declarative which is assigned the app.component as follows –

src\app\new-cmp\new-cmp.component.spec.ts create src\app\new-cmp\new-cmp.component.ts update src\app\app.module.ts

@NgModule({ declarations: [

@NgModule({ declarations: [

```
AppComponent
],
imports: [
BrowserModule
],
providers: [],
bootstrap: [AppComponent]
})

export class AppModule { }
```

The declarations include the AppComponent variable, which we have already imported. This becomes the parent component. Now, angular-cli has a command to create your own component. However, the app component which is created by default will always remain the parent and the next components created will form the child components.
 The command used to create the component

ng g component new-cmp

When you run the above command in the command line, you will receive the following output C:\projectA4\Angular 4-app>ng g component new-cmp installing component create src\app\newcmp\new-cmp.component.css create src\app\new-cmp\new-cmp.component.html create

new-cmp.component.css – css file for the new component is created. new-cmp.component.html –

        html file is created.

new-cmp.component.spec.ts – this can be used for unit testing.

```
import { NewCmpComponent } from './new-
cmp/newcmp.component'; // includes the new-cmp component we
created

@NgModule({
  declarations: [
    AppComponent,
    NewCmpComponent // here it is added in declarations and will behave as a child component
  ],
  imports: [
    BrowserModule
  ],
  providers: [],   bootstrap: [AppComponent] //for bootstrap the AppComponent the main
app component is given. })

export class AppModule { }
```

        Now, if you go and cheque the same structure, we will get the new-cmp new folder created under the src/app folder.

The **new-cmp.component.ts** file is generated as follows import { Component, OnInit } from

'@angular/core'; // here angular/core is imported .

@Component({
  // this is a declarator which starts with @ sign. The component word marked in bold needs to
be the same.
selector: 'app-new-cmp', //
templateUrl: './new-cmp.component.html',   //
reference to the html file created in the new component.
styleUrls: ['./new-cmp.component.css'] // reference to the style
file. })
export class NewCmpComponent implements OnInit
{   constructor() { }
ngOnInit() {}
}

If you see the above new-cmp.component.ts file, it creates a new class called NewCmpComponent, which implements OnInit. In, which has a constructor and a method called ngOnInit(). ngOnInit is called by default when the class is executed.

Let us check how the flow works. Now, the app component, which is created by default becomes the parent component. Any component added later becomes the child component.

When we hit the URL in the **http://localhost:4200/** browser, it first executes the index.html file which is shown below −

<!doctype html>
<html lang = "en">
  <head>
    <meta charset = "utf-8">
    <title>Angular 4App</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>

```html
<app-root></app-root>
```

```html
</body> </html>
```

The above is the normal html file and we do not see anything that is printed in the browser. Take a look at the tag in the body section

`<app-root></app-root>`

This is the root tag created by the Angular by default. This tag has the reference in the **main.ts** file.

```typescript
import { enableProdMode } from '@angular/core';  import {
platformBrowserDynamic } from
'@angular/platformbrowserdynamic'; import { AppModule } from
'./app/app.module';  import { environment } from
'./environments/environment';

if (environment.production) {    enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule);
```

AppModule is imported from the app of the main parent module, and the same is given to the bootstrap Module, which makes the appmodule load.

```typescript
import { BrowserModule } from
'@angular/platform-browser'; import { NgModule } from
'@angular/core'; import { AppComponent } from
'./app.component';   import { NewCmpComponent } from './new-
cmp/new-cmp.component';

@NgModule({
  declarations: [
    AppComponent,
    NewCmpComponent
  ],
  imports: [
    BrowserModule
```

```
    ],
```

```
  providers: [],
  bootstrap: [AppComponent]
})

export class AppModule { }
```

Here, the AppComponent is the name given, i.e., the variable to store the reference of the **app. Component.ts** and the same is given to the bootstrap.

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',   templateUrl:
'./app.component.html',   styleUrls:
['./app.component.css']
})

export class AppComponent {
title = 'Angular  Project!'; }
```

Angular core is imported and referred as the Component and the same is used in the Declarator as

```
@Component({
selector: 'app-root',  templateUrl:   './app.component.html',
styleUrls:
['./app.component.css'] })
```

In the declarator reference to the selector, **templateUrl** and **styleUrl** are given. The selector here is nothing but the tag which is placed in the index.html file that we saw above.

The class AppComponent has a variable called title, which is displayed in the browser.

The **@Component** uses the templateUrl called

app.component.html which is as follows<!--The content below
is only a placeholder and can be replaced.--> <div style="textalign:center">

<!--The content below is only a placeholder and can be replaced.--> <div style="text-align:center">
<h1>
Welcome to {{title}}.

<h1>

<h1>

Welcome to {{title}}.
</h1> </div>

It has just the html code and the variable title in curly brackets. It gets replaced with the value, which is present in the **app.component.ts** file. This is called binding. We will discuss the concept of binding in a subsequent chapter. Now that we have created a new component called **new-cmp**. The same gets included in the **app.module.ts** file, when the command is run for creating a new component.

app.module.ts has a reference to the new component created.

new-cmp.component.ts

```
import { Component, OnInit } from '@angular/core';
@Component({  selector: 'app-
new-cmp',
templateUrl:
'./newcmp.component.html', styleUrls:
['./newcmp.component.css']
})

export class NewCmpComponent implements
OnInit { constructor() { } ngOnInit() {} }
```

Here, we have to import the core too. The reference of the component is used in the declarator.

The declarator has the selector called **app-new-cmp** and the **templateUrl** and **styleUrl**.

The .html called **new-cmp.component.html** is as follows

```
<p> new-comp works! <p>
```

we have the html code, i.e., the p tag. The style file is empty as we do not need any styling at present. But when we run the project, we do not see anything related to the new component getting displayed in the browser. Let us now add something and the same can be seen in the browser later. The selector, i.e., **app-new-cmp** needs to be added in the **app.component .html** file as follows

```
</h1>
</div>
```

<app-new-cmp></app-new-cmp>

When the **<app-new-cmp></app-new-cmp>** tag is added, all that is present in the .html file of the new component created will get displayed on the browser along with the parent component data. Let us see the **new component .html** file and the **new-cmp.component.ts** file.

New-cmp.component.ts

```
import { Component, OnInit } from '@angular/core';

@Component({    selector: 'app-
new-cmp',              templateUrl:
'./newcmp.component.h        tml',
styleUrls:
['./newcmp.component.c ss']
})

export class NewCmpComponent implements OnInit {
new component = "Entered in new component created";
constructor() {} ngOnInit() { }
}
```

In the class, we have added one variable called new component and the value is "**Entered in new component created**".

The above variable is bound in the **.new-cmp.component.html** file as follows –

```
<p>
{{newcomponent}}
</p>
 <p> new-cmp works! </p>
```

Now since we have included the **<app-new-cmp></app-new-cmp>** selector in the **app. component**

.html which is the .html of the parent component, the content present in the new component .html file (new-cmp.component.html) gets displayed on the browser
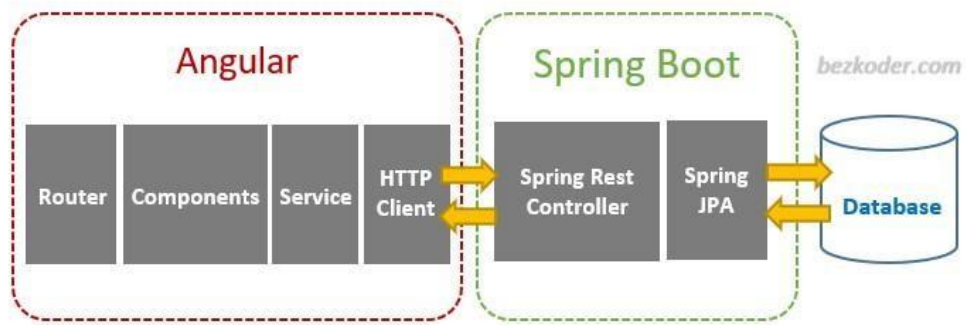
Prototype model

Spring boot – Backend

Dependency added in POM –

- Lombok
- spring security
- spring Jpa

- spring -web
- Common csv
- Jwt – (disable – for now)
- Spring-boot starter mail

LomBok Project Lombok is a Java library tool that generates code for minimizing boilerplate code. The library replaces boilerplate code with easy-to-use annotations.

@Getter,
@Setter
 @NoArgsConstructor,
@AllArgsConstructor
@Data
@NotNull

Spring security

Spring Security Web dependencies are used for providing features related to the web security of an application like:

- Restricted access to URLs in a Servlet Container environment • Managing the Spring Secuirty filters related to web

spring
Jpa

Spring Data JPA aims to significantly improve the implementation of data access layers by reducing the effort to the amount that's actually needed. As a developer you write your repository interfaces, including custom finder methods, and Spring will provide the implementation automatically.

Common csv- Apache Commons CSV is a java library that reads and writes files in Comma Separated Value (CSV) format. This library can write in all supported variations of CSV files like MSExcel, Informix, Oracle, MySQL and TSV(Tab-separated values). You can also create custom CSV formats using its fluent API.

Entities – Entities Classes are used to initialize the variables and make a model format for table
- User
- Product-Info
- Wishlist
- Product order
- Product category
- Order-main

- discount

- cart

User Entity – This class entity used for collecting data from the user for registration purpose.

public class User implements a seriliazeble - Serializability of a class is enabled by the class implementing the java.io.Serializable interface.

Annotation used –
- @Entity – for make class to table
- @Data – it contains getter setter to string method itself
- @noArgsConstructor – for no argument constructor
- @ID – make a primary key for table
- @NotEmpty – The annotated element must not be null nor empty. Supported types are:

Private Variable -ex private
Long id;

- ID
- Name
- Email Password
- Phone
- Address
- Active status
- Role - For_Customer
- Cart

ProductInfo – Class entity used for the product information

Annotation used
- @Entity
- @Id

- @DymanicUpdate - For updating, should this entity use dynamic sql generation where only changed columns get referenced in the prepared sql statement?

- @min - The annotated element must be a number whose value must be higher or equal to the specified minimum.
- @Column Default - Identifies the DEFAULT value to apply to the associated column via DDL.
- @creation TimeStamp - Marks a property as the creation timestamp of the containing entity. The property value will be set to the current VM date exactly once when saving the owning entity for the first time.
- @Update TimeStamp - Marks a property as the creation timestamp of the containing entity. The property value will be set to the current VM date exactly once when saving the owning entity for the first time.

Variables
- productid
- productDescription
- productPrice
- productStock
- productStatus
- productIcon
- createTime
- updateTime
- categoryType

WishList Entity class -
Class use for add product into wishlist table

Annotation Used–
@id -

@generatedValue – GenerationType.AUTO - used for autogenerated values ○ @Column – Specifies the mapped column for a persistent property or field

@ManyToOne - Specifies a single-valued association to another entity class that has many-to-one multiplicity . The ManyToOne annotation used within an embeddable class to specify a relationship from the embeddable class to an entity class.

@JoinColumn - Specifies a column for joining an entity association or element collection. If the JoinColumn annotation itself is defaulted, a single join column is assumed and the default values apply.

@OneToOne - Specifies a single-valued association to another entity that has one-to-one multiplicity. Ie one entity interlink with another entity

The FetchType EAGER –
Eager strategy is a requirement on the persistence provider runtime that data must be eagerly fetched.

Variables-
- ID

- User user (entity) – One to one Relationship
- ProductInfo productInfo (Entity) – manytoOne relationship

Product Order - Entity class used for the placed Product order it has manytoOne relationship with

Cart entity and Order main entity

Annotation -

- @Entity
- @Data
- @Id
- @GeneratedValue
- @ManyToOne
- @JoinColumn • @NotNull
- @NotEmpty
- @Min
- @JsonIgnore - Annotation is usually used just a like a marker annotation, that is, without explicitly defining 'value' argument (which defaults to true): but argument can be explicitly defined. This can be done to override an existing `JsonIgnore` by explicitly defining one with 'false' argument: either in a sub-class, or by using "mix-in annotations".

Variable

- Id
- Cart cart(Entity with manytoOne relationship)
- OrderMain(entity with

@many to one Relationship)

- productid
- prooductName
- productDescription
- productIcon
- CategoryType
- productPrice
- productStock
- count

Method

To string method for data display. Getter and setter methods

Product -Category

this entity class used for distinguish the product into different categories.

Annotation –

- @Entity
- @Data
- @Dynamic Update
- @Id
- @Generated value
- @naturalID - This specifies that a property is part of the natural id of the entity.

Methods -
Getter setter methods and no Argument Constructor and Required Argument Constructor

Variables

- CategoryId
- categoryName
- CategoryName
- categoryType
- createTime(Date as DataType)
- updateTime **Order-Main** –

this entity class used for the placed the order with productINorder checkout and deatials of the user

Annotation
- @Entity
- @Data
- @Dynamic update
- @Generated value
- @OneToMany
- @NotEmpty
- @CreationTimeStamp
- @UpdateTimeStamp
- @ColumnDefault – if the value is not given then its takes default column value

Methods – Getter setter , no- Argument Constructor, Required-Argument Constructor

Variables
- Order
- ID
- ProductOrder set(entity with OneToMany Relationship)

- BuyersEmail(current User Email)
- BuyersPhone
- BuyersAddress
- OrderAmount
- OrderStatus
- CreatTimeStamp
- updateTimeStamp

Discount - this Entity class used for the making discount coupon

Annotation

- @Entity
- @Data
- @ID
- @Table
- @NoArgsData

Methods – Getter Setter

Cart – this class entity used for the add product into cart by each user

Annotation -

- @Entity
- @Data
- @Dynamic Update
- @Id
- @Generated value
- @OneToMany
- @JsonIgnore
- @MapsId - The value element specifies the attribute within a composite key to which the relationship attribute corresponds. If the entity's primary key is of the same Java type as the primary key of the entity referenced by the relationship, the value attribute is not specified.

Variables
- Id
- User user(Entity with OnetoOne Relationship) mainly for is user has its separate cart reference column is email)
- Set ProductOrder (entity with OneToMany mapping)

Methods

- @Data
- @toString Method   •   @NoArgsConstrutor.

Repository

Repository - A repository is **a mechanism for encapsulating storage, retrieval, and search behaviour which emulates a collection of objects**. It is a specialization of the

- @Component annotation allowing for implementation classes to be autodetected through classpath scanning.

Annotation Used in All classes -

- @Repositroy
- @Repository Annotation is a specialization of
- @Component annotation which is used to indicate that the class provides the mechanism for storage, retrieval, update, delete and search operation on objects. Though it is a specialization of
- @Component annotation, so Spring Repository classes are autodetected by spring framework through classpath scanning. This annotation is a general-purpose stereotype annotation which very close to the DAO pattern where DAO classes are responsible for providing CRUD operations on database tables.

All classes Extends a JpaRespository
JpaRepository is a JPA (Java Persistence API) specific extension of Repository. It contains the full API of CrudRepository and PagingAndSortingRepository. So, it contains API for basic CRUD operations and also API for pagination and sorting.

Repository Classes -

- CartRepository – used for performing CRUD Operation on Cart Entity
- DiscountRepositroy – used for performing CRUD Operation on DiscountEntity
- OrderRepository - used for performing CRUD Operation on OderEntity
- ProductCategoryRepository -used for performing CRUD Operation on ProductCategory entity
- ᴘroductInOrderRepository - used for performing CRUD Operation on ProductOrder
- Entity
- UserRepository- used for performing CRUD Operation on User Entity

- WishListCustomRepository-   used for   performing   CRUD Operation   on
- WishListCustom Entity
- WishListRepository - used for performing CRUD Operation on WishList Entity.

Controller class - In Spring Boot, the controller class is responsible for processing incoming REST API requests, preparing a model, and returning the view to be rendered as a response.

Annotation -

- @Controller annotation is a specialization of the generic stereotype
- @Component annotation, which allows a class to be recognized as a Springmanaged component.
- @Controller annotation extends the use-case of
- @Component and marks the annotated class as a business or presentation layer
- @RestContoller – The @RestController annotation in Spring is essentially just a combination of @Controller and @ResponseBody. This annotation was added during Spring 4.0 to remove the redundancy of declaring the @ResponseBody annotation in your controller
- @RequestMapping -Annotation for mapping web requests onto methods in requesthandling classes with flexible method signatures.

- @CrossOrigin - Annotation for permitting cross-origin requests on specific handler classes and/or handler methods. Processed if an appropriate HandlerMapping is configured.

- @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.

- @GetMapping - Annotation for mapping HTTP GET requests onto specific handler methods. Specifically, @GetMapping is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.GET).

- @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).

- @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).

- @DeleteMapping - Annotation for mapping HTTP DELETE requests onto specific handler methods. Specifically, @DeleteMapping is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.DELETE).

- @PathVariable - Annotation which indicates that a method parameter should be bound to a URI template variable. Supported for RequestMapping annotated handler methods.

- @RequestBody - Annotation indicating a method parameter should be bound to the body of the web request. The body of the request is passed through an HttpMessageConverter to resolve the method argument depending on the content type of the request. Optionally, automatic validation can be applied by annotating the argument with @Valid.

- @RequestParam - Annotation which indicates that a method parameter should be bound to a web request parameter.

Controller Classes -
- CartController
- CategoryController
- CSVController
- DISCountController
- EmailController
- OrderController
- ProductController
- ProductController
- USerController
- WishListController

Service Class –
A service interface is a published interface used to invoke a service. An interface can be implemented using any number of technologies.
In service Interface We Just Write a Method and thir logic code will be Write in Service Impl.

Service Interface –
- CartService
- CategoryService
- DiscountService
- OrderServic

- ProductService
- UserService

SERVICE IMPL CLASS

The service implementation should be a Spring bean (it either has to have a @Component or @Service annotation, or should be defined in a Spring XML configuration file), so that Spring will find it and register it in the Spring application context.

The main function of the Service Impl is to provide services to the Admins and the Users.

Cart Service Impl class: - In this class we will able to add the products to the cart.

- @Service annotation: These class files are used to write business logic in a different layer, separated from @RestController class file.

- @Autowired annotation: provides more fine-grained control over where and how auto wiring should be accomplished.

- @Override annotation: informs the compiler that the element is meant to override an element declared in a superclass.

- @Transactional annotation: is the metadata that specifies the semantics of the transactions on a method.

- Category Service Impl class: - In this all the products that added to the application are being categorized with the required category.

- @Service

- @RestController annotation: in order to simplify the creation of RESTful web services. But unfortunately, we haven't used these annotations in this spring boot application.

- @Autowired

- @Override

- @Transactional

- CSV Service class: - In this file it will able to store the products in column by column, and split it by the seperater.

  - @Service annotation

  - @Autowired

- Discount Service Impl class: - In this we will create the discount column, and able to add and delete the discount coupons to the products.

  - @Service

  - @Autowired

  - @Transactional

  - @Override

- Email Service class: - In this we will able to add the email service to the customers like the update of the receiving the details of the products.

  - @Service

  - @Autowired

  - @Value Spring annotation: This annotation can be used for injecting values into fields in Spring-managed beans, and it can be applied at the field or constructor/method parameter level.

- Order Service Impl class: - In this class we will able to see the details of the order of the products that ordered by the customers.

- @Service

- @Autowired

- @Override

- @Transactional

- Product In Order Service Impl: - In this it will show that the product is ordered or not and we will able to increase the number.

- @Service

- @Autowired

- @Override

- @Transactional

- Product Service Impl class: - In this we will able to add the details of the products.

- @ Service

- @Autowired

- @Override

- @Transactional

- User Service Impl class: - In this the user can able to set the password for the application or the account of the website.

- @ Service

- @DependsOn: We should use this annotation for specifying bean dependencies. Spring guarantees that the defined beans will be initialized before attempting an initialization of the current bean.

- @Autowired

- @Override

- @Transactional

- WishList Service class: - In this creation of the wishlist bar or menu, and able to add the products to it.

- @ Service

- @Transactional

- @Autowired

SECURITY JWT AUTHENTICATION

JWT, or JSON Web Token, is an open standard used to share security information between two parties — a client and a server.

The main function of this is to provide the security User data and the customes data that have

stored in the server.

1. Jwt Entry Point class: - taking the entry of the users.

- @Component: is an annotation that allows Spring to automatically detect our custom beans.

- @Override

  2. Jwt Filter class: - checks if the incoming request has a valid JSON Web Token (JWT).

- @Component

- @Autowired

- @Override

3. Jwt Provider class: - It will allow to see the user details.

- @Component

- @Value

Other Annotation Used are: -

- @Qualifier annotation: we can eliminate the issue of which bean needs to be injected.

- @Configuration annotation: indicates that a class declares one or more @Bean methods and may be processed by the Spring container to generate bean definitions and service requests for those beans at runtime.

- @EnableWebSecurity annotation: is annotated at class level with @Configuration annotation to enable web securities in our application defined by Web Security Configured implementations.

- @Bean annotation: is applied on a method to specify that it returns a bean to be managed by Spring context.

REQUEST

Defines a builder that adds a body to the response entity. Defines a builder that adds headers to the request entity. Request Entity initialized with a URI template and variables instead of a URI.

Login Form class: - It is used to create the login details of the customers.   Annotation Used: -

- @Data: It is a convenient shortcut annotation that bundles the features of @ToString.

- @NotBlank: to say that a string field must not be the empty string (i.e., it must have at least one character.

- @Test annotation: is useful when we need to bootstrap the entire container.

RESPONSE

Response Entity represents the whole HTTP response: status code, headers, and body. As a result, we can use it to fully configure the HTTP response. If we want to use it, we have to return it from the endpoint; Spring takes care of the rest.

This is the response class which are used to see the responses of the customers which will stores the data. We need responses of only these categories.

1. Category Page class

2. Jwt Response class

3. Product Info Response class

4. User List Response class   Annotation Used: -

 • @Data: It is a convenient shortcut annotation that bundles the features of @ToString.

Application.yml

in this file we defined some method for connecting Database to the spring.

Like databases URL , dialect username password properties mail etc

Properties

Spring DATASOURCE (DataSourceAutoConfiguration & DataSourceProperties)

spring:

 datasource:

   driver-class-name: org.postgresql.Driver   username:
postgres   password: admin   url:
jdbc:postgresql://localhost:5432/ecommerce
platform: postgres   initialization-mode: always
continue-on-error: true

```yaml
  mail:    default-encoding: UTF-8

host: smtp.gmail.com    username:

ayushx16@gmail.com    password:

134153@#

   port: 587

properties:    mail:

smtp:

     auth: true       starttls:

enable: true

protocol: smtp

testconnection: false


 jpa:

  show-sql: false #   Generate-

ddl: false    hibernate:

   ddl-auto: none    database:

postgresql

properties:

hibernate:      temp:

    use_jdbc_metadata_defaults: false;    databaseplatform:

org.hibernate.dialect.PostgreSQL9Dialect


 queries:

  users-query: select email, password, active from users where email=?    rolesquery:

select email, role from users where email=?
```
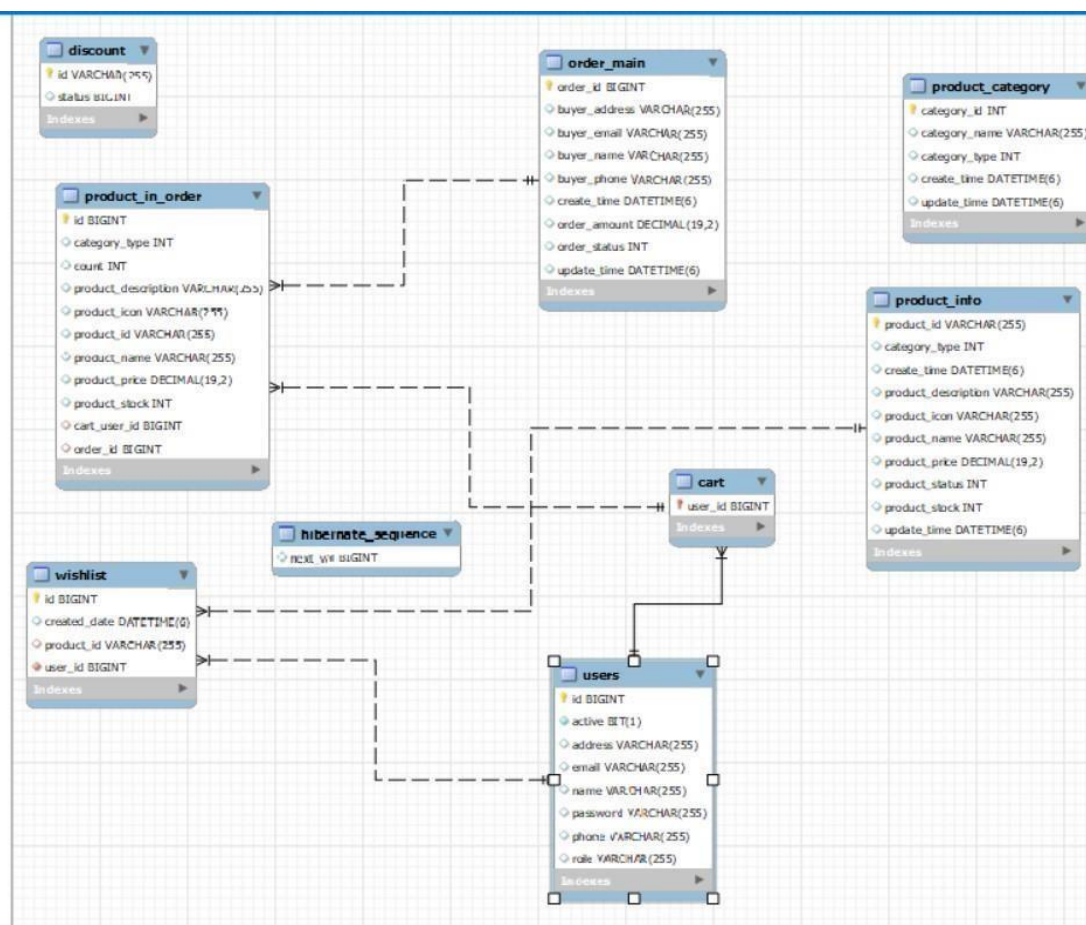
server: servlet:

  contextPath: /api

jwtSecret: me.zhulin jwtExpiration:

86400

DATABASE PostgreSQL databases

PostgreSQL, also known as Postgres, is a free and open-source relational database management system emphasizing extensibility and SQL compliance. It was originally named POSTGRES

Final Result

Sign In page

Signup Page
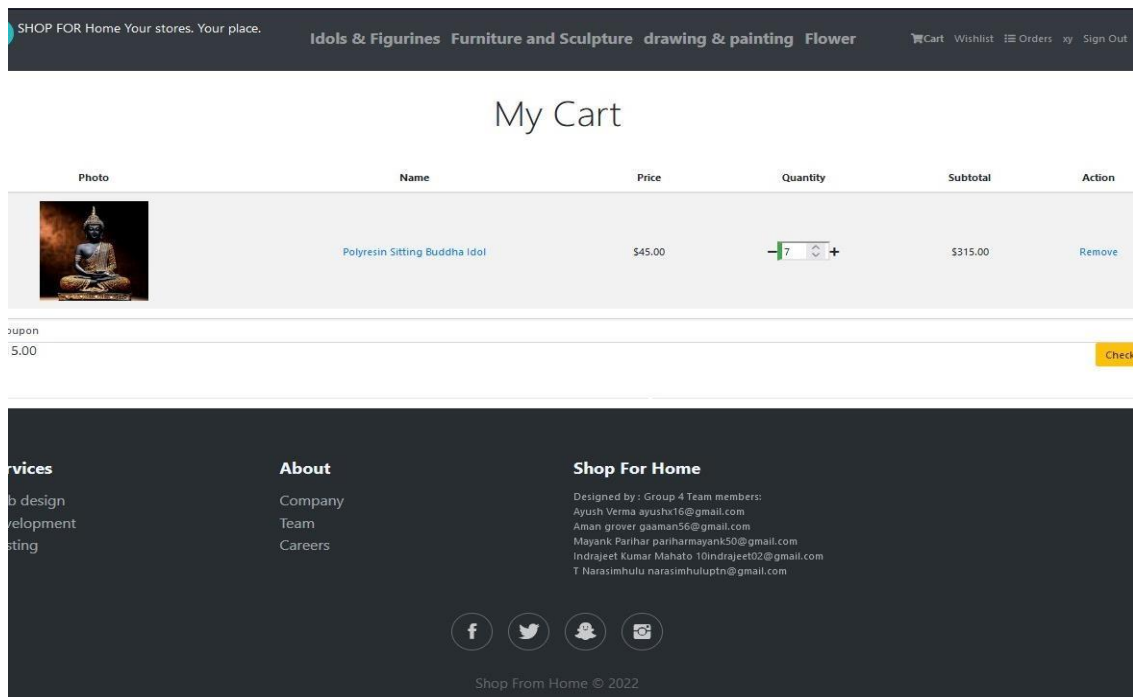
Cart

Conclusion

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

We can create a web page using the Angular, Spring boot, Node-JS, MySQL, Eclipse etc..

In this project, we have created the page of name "**SHOP FOR HOME**" in which a user can buy the house hold items like decorating things and many other.

We are needed updated or latest version software for smooth working of the project.

Both frontend and back end needs to run in a frequent manner for to have the page with user interface flexibility.

At the end we can say that web page is created by following the path regulations and shown the output as required.

Reference

Project references are references to projects that contain assemblies; you add project references by using the Projects tab of the Reference Manager dialog box. Visual Studio can find an assembly when given a path to the project

And From YouTube plus from Recorded lectures On GreatLearning

Properties for BookyardClient

type filter text

Project References

▷ Resource
  Builders
  Git
  Java Build Path
▷ Java Code Style
▷ Java Compiler
▷ Java Editor
  Javadoc Location
  Project Facets
  Project References
  Run/Debug Settings
▷ Task Repository
  Task Tags
▷ Validation
  WikiText

Projects may refer to other projects in the workspace.
Use this page to specify what other projects are referenced by the project.

Project references for 'BookyardClient':

- ☑ > Bookyard.Contracts [Practice master]
- ☐ > Bookyard.server [Practice master]
- ☐ jjwt
- ☐ MyFirstSwingGuiProject [Practice master]
- ☐ RemoteSystemsTempFiles [Practice master]
- ☐ > Servers [Practice master]
- ☐ ServletTutorial [Practice master]

OK   Cancel