

Online Food Delivery System

FOOD BOX

Team Members

- 1. Gayatri Sulkar**
- 2. Mayuri Dilip Yadav**
- 3. Sunena Saikia**
- 4. Jammuladinne Vineela**
- 5. Pratiksha Thorbole**
- 6. Sujal Upadhyay**
- 7. Mule Kalyanreddy**
- 8. Gnana Kumar R**
- 9. Duggempudi R**
- 10. Dinesh S**

Project Code:	6
Project Name:	Online Food Delivery System

Table of Contents

1.	INTRODUCTION	3
2.	SYSTEM OVERVIEW	3
3.	SUB-SYSTEM DETAILS	9
4.	DATA ORGANIZATION	11
5.	REST APIs to be Built	25
6.	IMPLEMENTATION DETAILS	20
7.	UNIT TESTING	23
8.	FUNCTIONAL TESTING	25
9.	CONCLUSION	28

1. Introduction

The project Online Food ordering system is a web-based application that allows the administrator to handle all the activities online quickly and safely. Using Interactive GUI anyone can quickly learn to use the complete system.

Using this, the administrator doesn't have to sit and manage the entire activities on paper, and at the same time, the head will feel comfortable to keep check of the whole system. This system will give him power and flexibility to manage the entire system from a single online portal.

Scope and Overview:

The scope of the “**Online Food Delivery System**” will be to provide the functionality as described below. The system will be developed on a Windows operating system using Java/J2EE, Hibernate, Spring.

2. System Overview

The “**Online Food Delivery System**” supports basic functionalities (explained in section 2.1) for all below listed users.

- Administrator (A)
- Customer (C)

2.1 Authentication & Authorization

2.1.1 Authentication:

1. Any Customer or End-user may login using a unique **User id** and **Password**.
2. Admin can also do login using his/her unique **User id** and **Password**.

2.1.2 Authorization

The below listed Operations can be done by both Administrator and Customer.

Administrator:

1. Able to log in as admin in Online Food Delivery System with his/her Unique user id and Password.
2. Able to update or Remove Food item Price, Name, Category, Description and its Status.
3. Able to view Customer details and orders.

Customer:

1. Able to login as Customer in Online Food Delivery System with his/her unique user id and Password.
2. Able to Register in to System using his/her Name, Email, Phone Number, Address and Password.
3. Able to add or Remove a Food item in the cart.
4. Able to order a Food item from the cart and purchase using UPI Payment method.

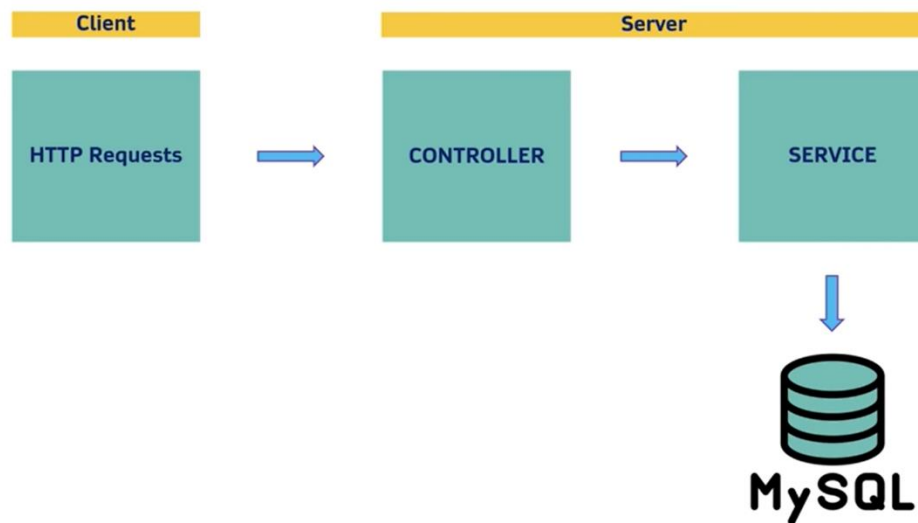
2.2 Functional Flow

The functional flow of the messages across different application components is shown below.

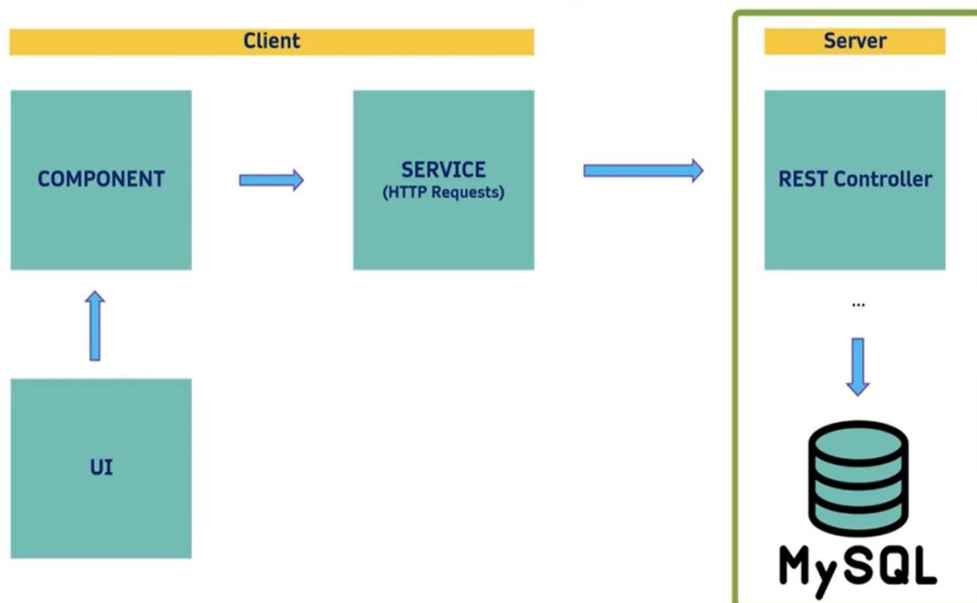
Ex. – Working Flow of a web application



API Design



API Design



2.3 Environment:

A. Hardware Requirements:

1. Intel hardware machine (PC with Pentium 1 and above, PC with 2 GB RAM, PC with 250 GB HDD or more).
2. Preferable Operating System -Windows.

B. Software Requirements:

Number	Description	Type
1	Front-End	Angular Framework by Google
2	Back-End	Spring Boot
3	Database	MySQL
4	Server	Apache Tomcat
5	IDE	Eclipse and Visual Studio Code
6	Testing Tools	Postman and Junit Framework

Front-End:



Angular is a TypeScript-based free and open-source web application framework led by the Angular Team at Google and by a community of individuals and corporations. Angular is a complete rewrite from the same team that built AngularJS.

Back-End:



Spring boot is a very convenient java-based framework comparing with others to develop a REST API for the backend. We don't need to waste our time doing a lot of coding for the configuration since spring boot has the feature of autoconfiguration and an embedded inbuilt Tomcat server.

Database:

MySQL is an open-source relational database management system to manage databases efficiently. It is a very flexible DBMS that provides advanced features and reliability.

Unit Testing:

We used Postman to initially test the backend REST API endpoints. Postman provides a sleek user interface with which to make HTML requests, without the difficulty of writing a bunch of code to test an API's functionality.

JUnit5 is used for the unit testing of the backend. It is a unit testing framework for java that helps to define the flow of execution of our code using different annotations.

3. Sub-system Details

The Online Food Delivery System is defined, wherein all users need to login successfully before performing any of their respective operations.

Find below (section 3.1 & 3.2) tables that provides functionality descriptions for each type of user /Customer.

3.1 Administrator

The administrator as a user is defined to perform below listed operations after successful login.

ID	Objects	Operations	Data to include	Remarks
AD-001 To AD-005	<i>Product</i>	Add View Delete Modify	Product Id, Product Name, Product Price, Description	
AD-005 To AD-0010	<i>Customer</i>	View	Username, email, phone number, Address.	
AD – 0011 to AD - 0013	<i>Order</i>	View	Order Id, User Id, Product Id, Product Price, Quantity, Tracking Number, Address	

3.2 Customer

The customer as a user is defined to perform below listed operations after successful login.

ID	Objects	Operations	Data to include	Remarks
US-001	<i>User</i>	Register and Edit	User Id, Username, Password, Email, Phone Number, Address etc.	
US-002	<i>Product</i>	Add to Cart. Delete from Cart.	Product Id, Product Name, Price, Quantity ,Availability	
US-003	<i>Order</i>	Add Product	OrderId, UserId, Product Id and Quantity, Price	

3.3 Login | Logout

[Web Application - J2EE, Hibernate, Spring]

- Go to Registration screen when you click on Register link.
- Go to Success screen when you login successfully after entering valid username & password fetched from the database.
- Redirect back to same login screen if username & password are not matching.

4. Data Organization

This section explains the data storage requirements of the Product Order Entry System and **indicative** data description along with suggested table (database) structure. The following section explains few of the tables (fields) with description. However, in similar approach need to be considered for all other tables.

4.1 Table: User_Registration_Details(Database table name :reg)

The user specific details such as username, email, phone etc. Authentication, and authorization / privileges should be kept in one or more tables, as necessary and applicable.

Field Name	Description
<i>User ID</i>	User ID is auto generated after registration and it is used as Login ID.
<i>F_name</i>	First Name of the Customer
<i>L_name</i>	Sur/Last Name of Customer
<i>Email</i>	Customer Email Id
<i>Phone Number</i>	10-digit contact number of users
<i>Password</i>	User Password
<i>Address</i>	Full Address of Customer including State and City
<i>Pin</i>	Zipcode / Pincode of current location

4.2 Table: Product_Details (Database table name :product)

This table contains information related to a product.

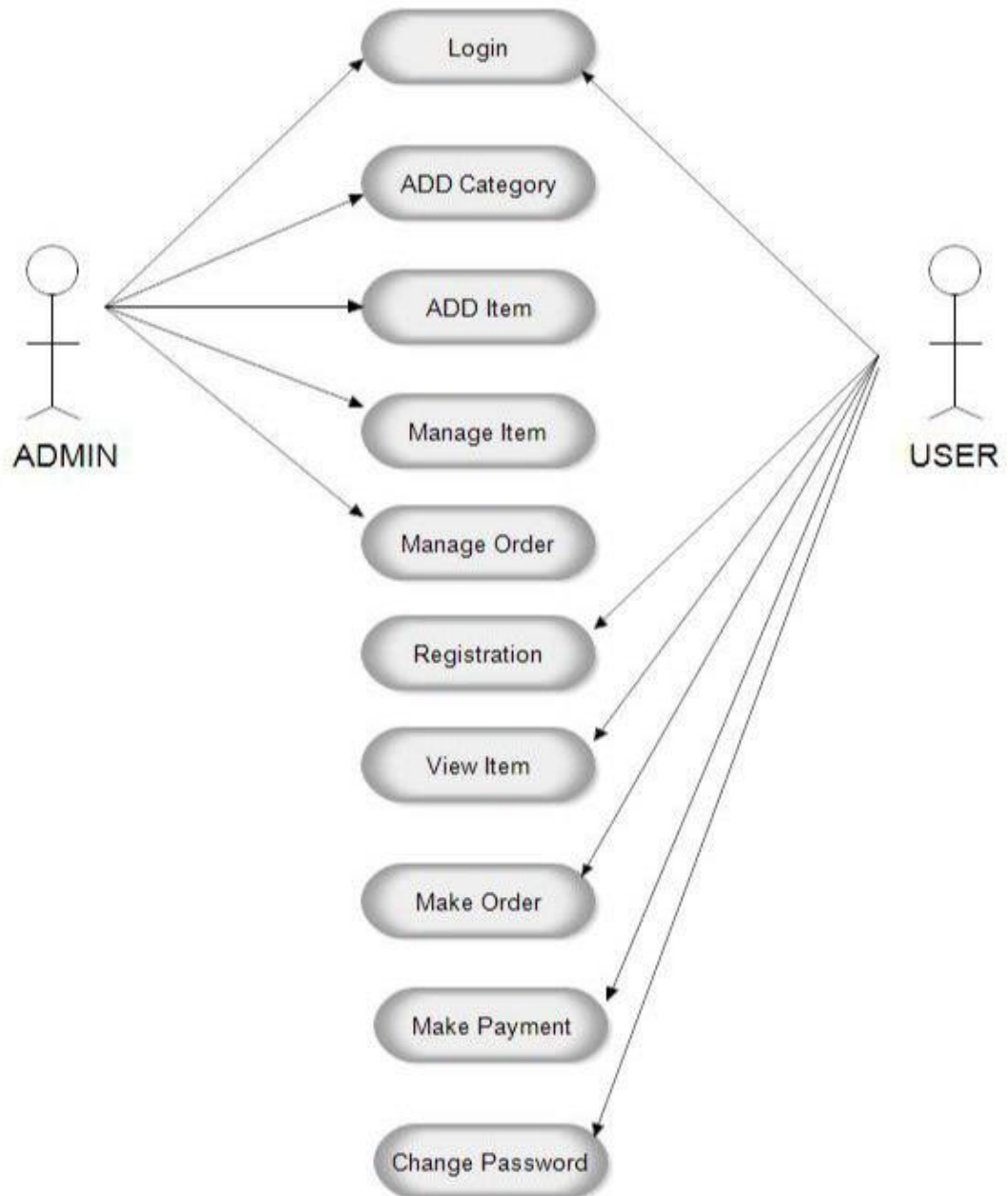
Field	Description
<i>Product_id</i>	Unique Product Id which is the Primary key to identify the product.
<i>P_name</i>	Name of the product {Eg;Gulab Jamun,Halwa}
<i>Description</i>	Detailed Description of the product
<i>Price</i>	Cost of the product
<i>Category</i>	Belongs to which category the described Food item
<i>Available</i>	Status/Availability of the Product
<i>Free Delivery</i>	Free /paid delivery
<i>Image Url</i>	Image of the product

4.3 Table: Order_Details (Database table name :order)

This table contains information related to cart details.

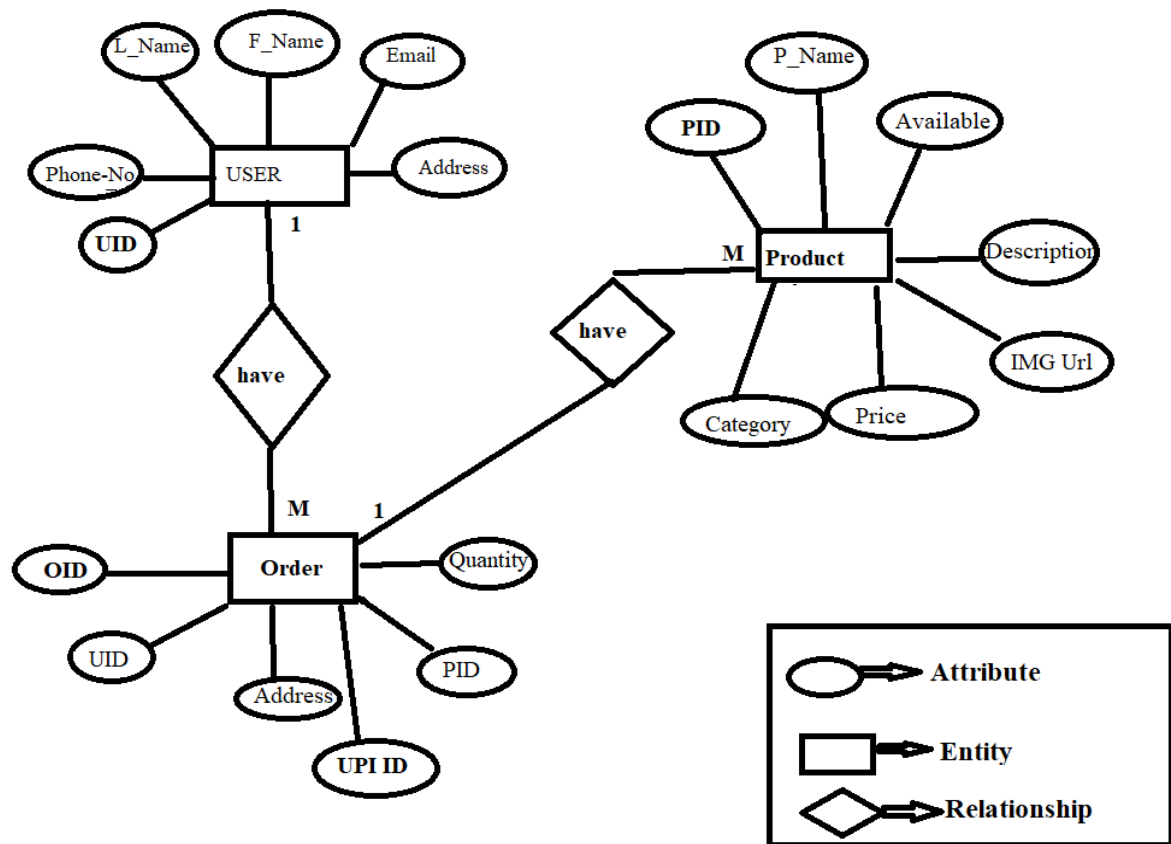
Field Name	Description
<i>Order_Id</i>	Unique Order ID Auto Generated (Primary Key)
<i>User_Id</i>	User Id corresponding to logged in user. (Foreign Key)
<i>Product_Id</i>	Product Id of the Selected product (Foreign Key)
<i>Quantity</i>	Quantity of the product in (E.g., 2 or 3 number of products)
<i>Address</i>	Address to be Delivered (Customer may change his location or may order his/her dear ones)
<i>Tracking Number</i>	Tracking Number will be created for the Order ID
<i>Upi_ID</i>	Customer can give his Upi ID to pay via online (Cashless Transaction)

4.4 UML Diagram

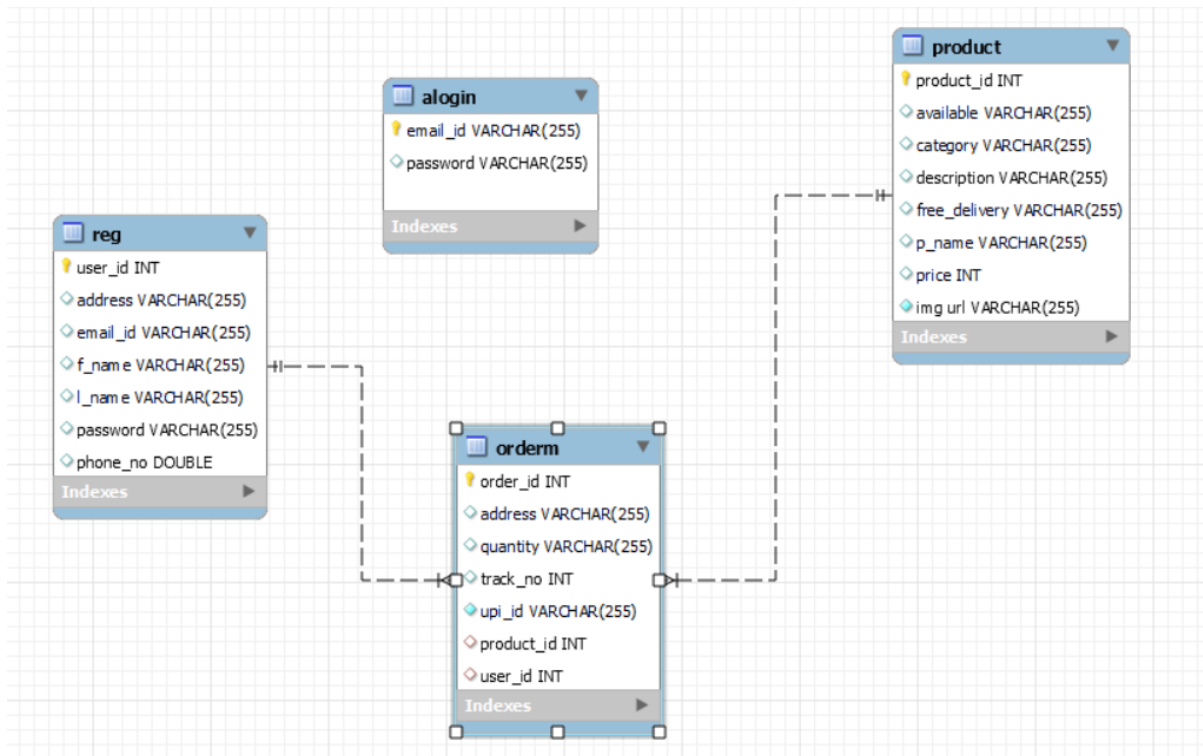


4.5 Entity Relationship Diagram

Entity Relationship Diagram



4.6 Database Schema



5. REST APIs to be Built.

Technology stack:

- Spring Boot
- Spring REST
- Spring Data JPA

5.1. Steps for creating a project in **Spring Boot**:

- In Eclipse IDE, we have to create a new Maven Project using required Group Id , Artifact Id ,Packaging and version.
- In POM.xml we need dependencies such as Spring Data JPA, Spring Boot Devtools , Mysql Driver and Spring web to support Spring application.
- Later we have to configure application.properties to connect with Mysql database.
- By creating Model, Service, Repository, Controller Packages we can perform the required Business Logics.

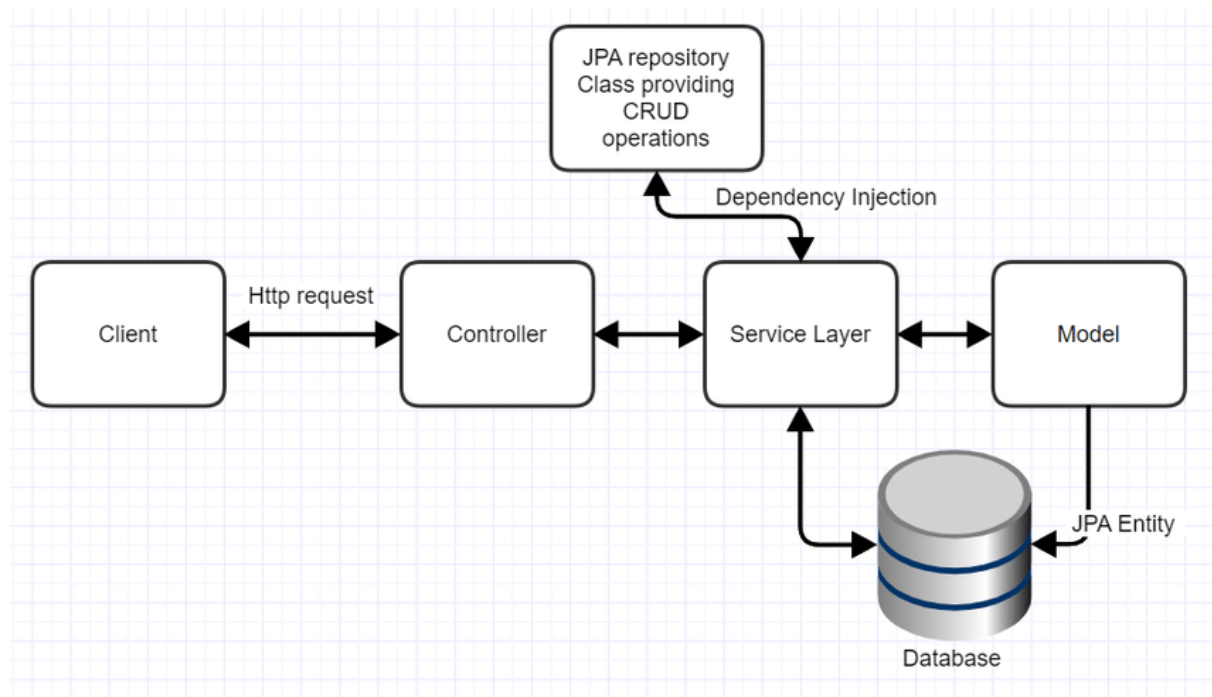


Fig A : Representation of Spring MVC pattern

5.2 Steps to create entities to perform Business logics

1. Creating *User* Entity:

Here will have multiple layers into the application:

1. In Eclipse IDE, We need to create an (Model class)
Entity: *User*
2. By Creating a UserRepository interface and will make use of Spring Data JPA.
 - a. Will have a query to validate user.
 - b. Add the User details by extending JPA Repository.
3. By Creating a UserService class we can perform the required Business logics and exposes all Services.
4. Finally, by creating a UserController class will handle all http requests and also will have following URI 'S:

URI	METHODS	Description	Format
<u>/api/v1/registers</u>	GET	Give a single user description searched based on username	JSON
/api/v1/register	POST	Add the user details	JSON
/api/v1/register	PUT	Update the user details	JSON
/api/v1/register	DELETE	Delete user by id	String

2. Creating Product Entity:

Build a RESTful resource for **Product** manipulations, where CRUD operations to be carried out. Here will have multiple layers into the application:

1. In Eclipse IDE, we need to create an Entity : *Product*
2. By Creating a ProductRepository interface and will make use of Spring Data JPA
 - a. Will have findByProductName method.
 - b. Add the Product details method.
 - c. Will have deleteProductById method.
 - d. Will have findAllProducts method.
 - e. Will have list method to view all products.
3. By Creating a ProductService class will perform the required Business logics.
4. Finally,by creating a ProductController will handle http requests using Request Mapping Annotation will have the following Uri's:

URI	METHODS	Description	Format
/api/v1/products	GET	Get all the products	JSON
/api/v1/product	GET	Give a single product description searched based on product name	JSON
/api/v1/Product	POST	Add the product details	JSON
/api/v1/product	DELETE	Delete a Product based on product id	JSON
/api/v1/product	PUT	Create product item	JSON

3. Creating Order Entity:

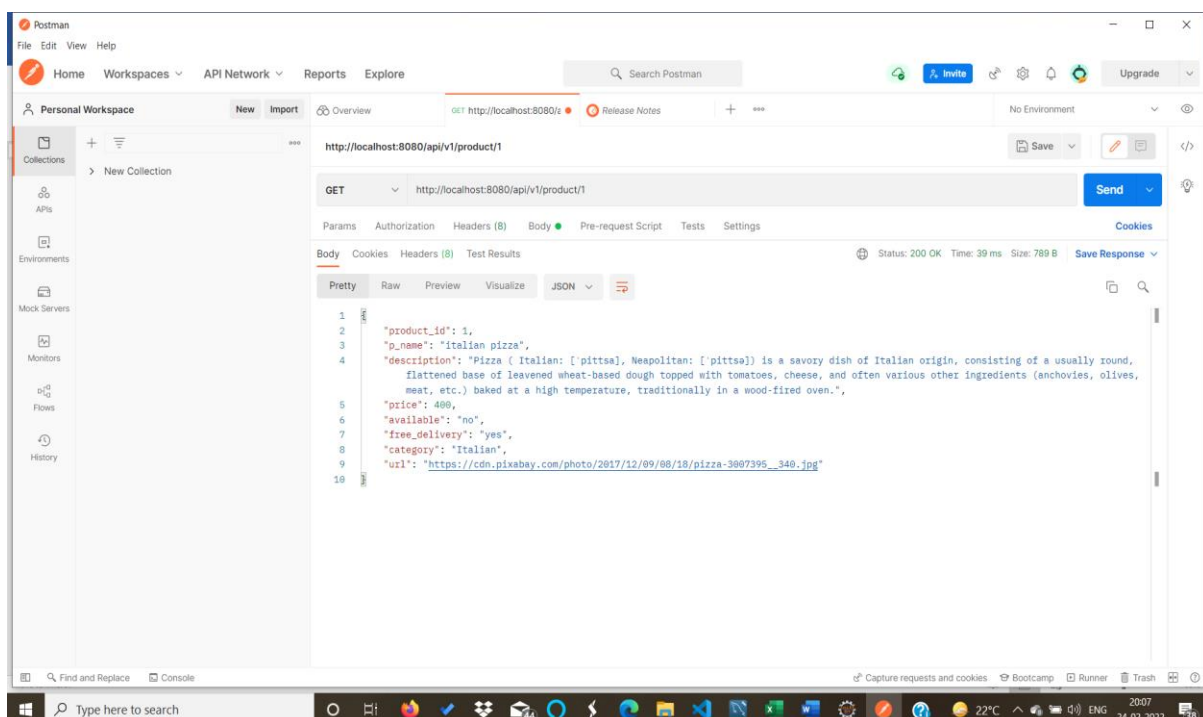
Build a RESTful resource for **ProductCart** manipulations, where following operations to be carried out. Here will have multiple layers into the application:

1. In Eclipse IDE we need to Create an Entity:*Order*
2. By Creating a OrderRepository interface and will make use of Spring Data JPA
 - a. Will have save method helps to save items in cart.
 - b. Will have deleteProductById method to remove item with specific product Id from cart.
 - c. Will have update method for updating the product and its Quantities.
 - d. Will have List method to see the list of Orders made.
3. Create a ProductCartService class and will expose all these services.
4. Finally, create a ProductCartRestController will have the following Uri's:

URI	METHODS	Description	Format
/api/v1/orders	GET	List the Products from the Database	JSON
/api/v1/order	POST	Create a Product item	JSON
/api/v1/order	PUT	Update the product item and its details	JSON
<u>/api/v1/order</u>	DELETE	Delete the item using particular Product id	JSON

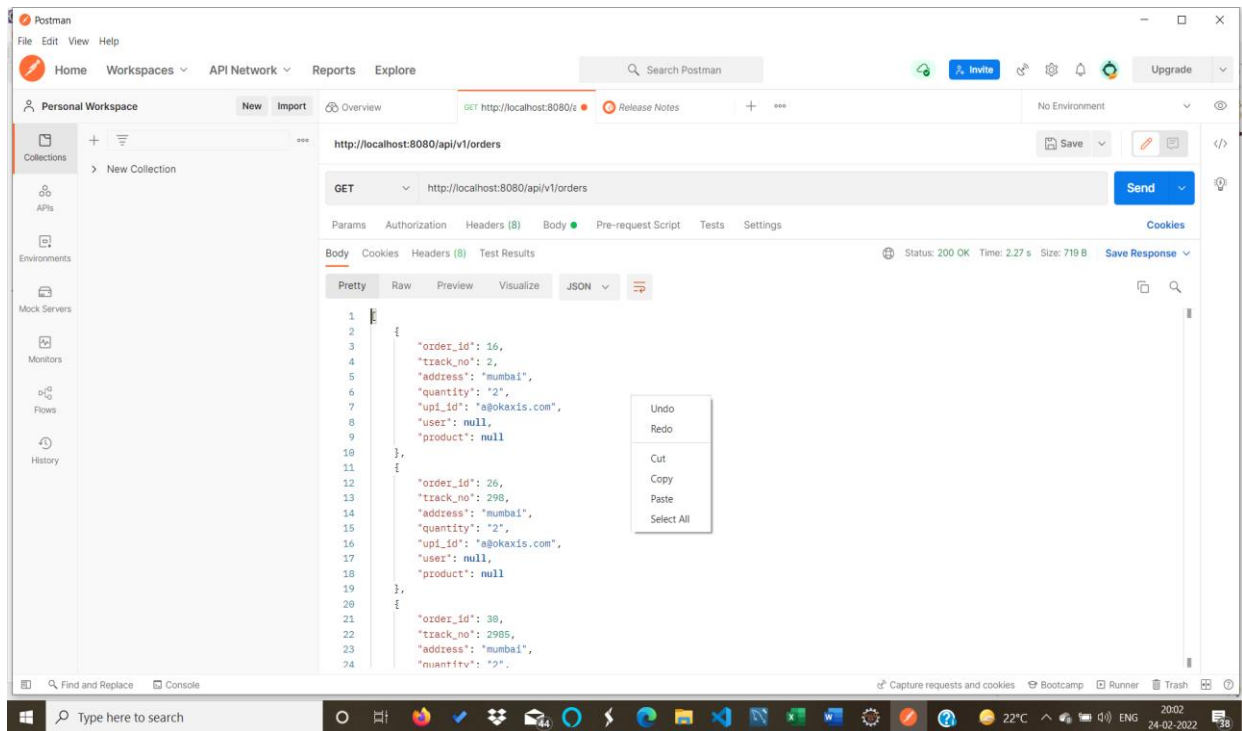
6. Implementation Details

After deciding the technologies, We started to implement our system. First, we developed the backend for our system by creating a REST API for the server-side application using java-based spring boot in IntelliJ IDEA. Then, we configured the backend API to connect to MySQL server @localhost:3306 to access the hospital management database created in the MySQL server which contains the tables for admin, doctor, user, schedule and booking. Then, we initially tested the API using postman by sending data in JSON format through HTTP GET, POST requests using the relevant URL endpoints.



Spring uses a JPA Java specification which is very useful to avoid writing native SQL queries in terms of tables and columns. It uses JPQL(Java Persistence Query Language) which is used to write queries in terms of java entities. Spring JPA helps to map data between a java application and a relational database using ORM(Object Relational Mapping).

Hence, the data sent from postman is stored to MySQL database using a POST request and retrieved from The database using GET request. After successfully testing the API, we developed the frontend using vue.js in visual studio code. We created the navigator window and designed the structure of our webpage using HTML5, then styled the web page using CSS3 and finally added dynamic and behaviours to our web page using Javascript. We also created a footer at the bottom of our home page.



Thereafter, we integrated the frontend, backend and database to create our hospital management system website. Finally, we tested our web application by performing unit tests using Junit for backend and Jest for the frontend to make sure all the required functionalities of our system are working properly as expected to be made available for the users.

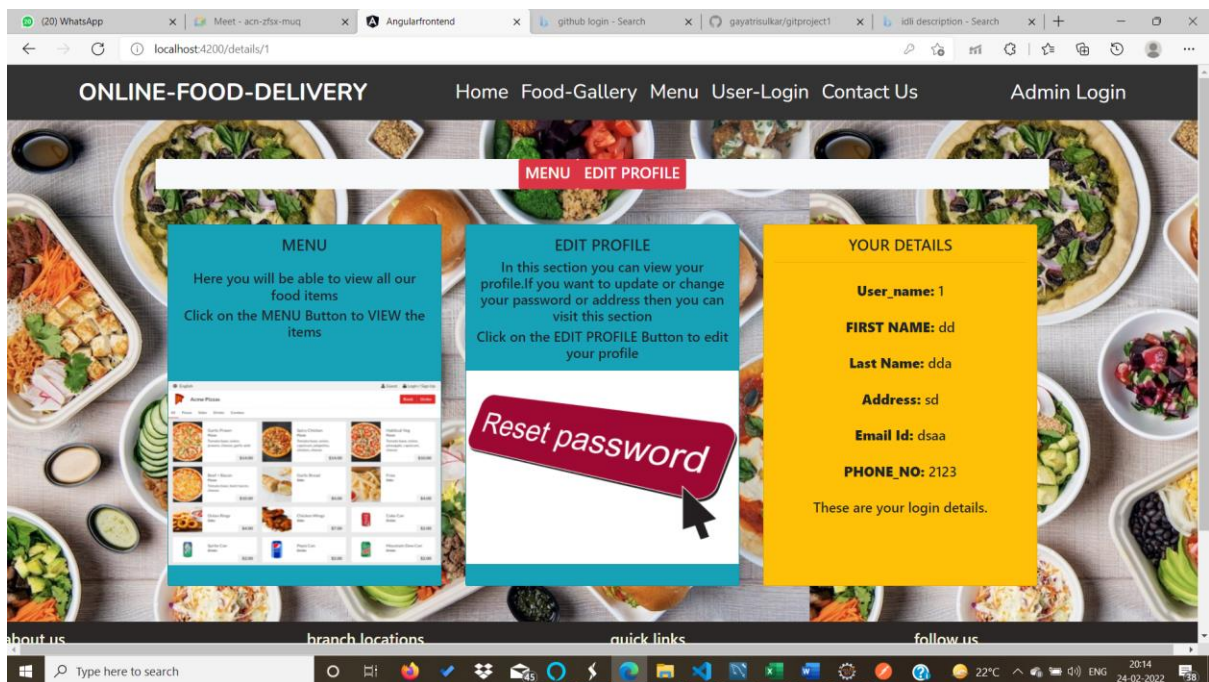
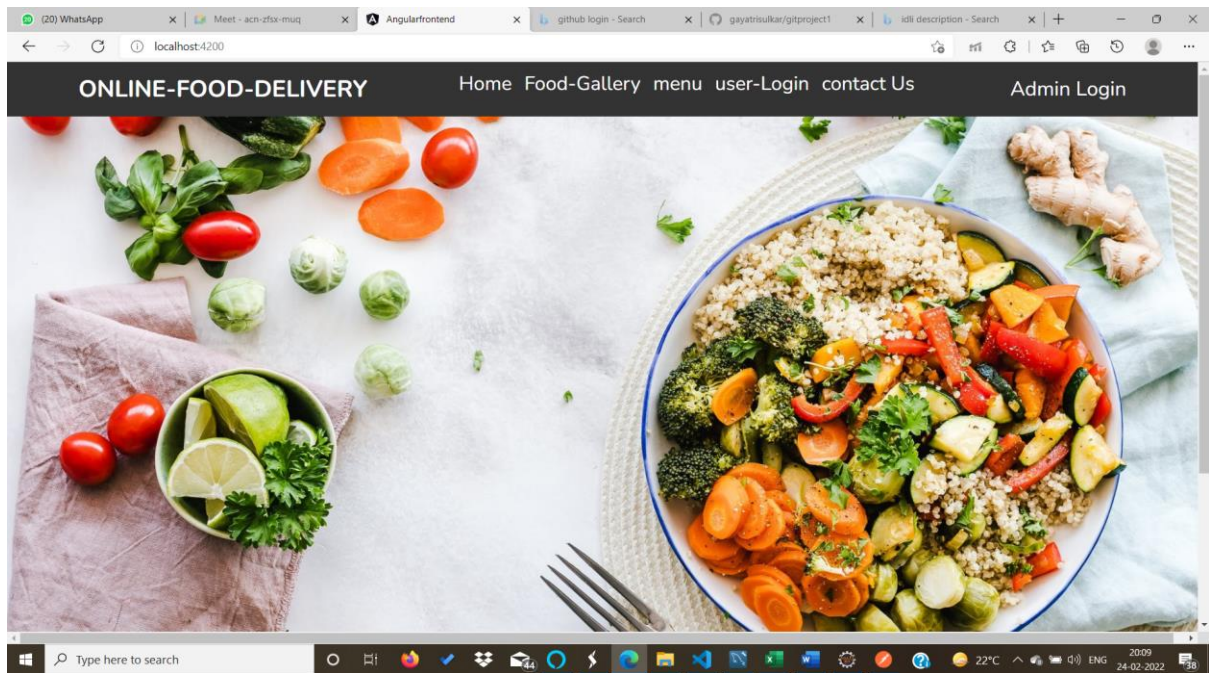
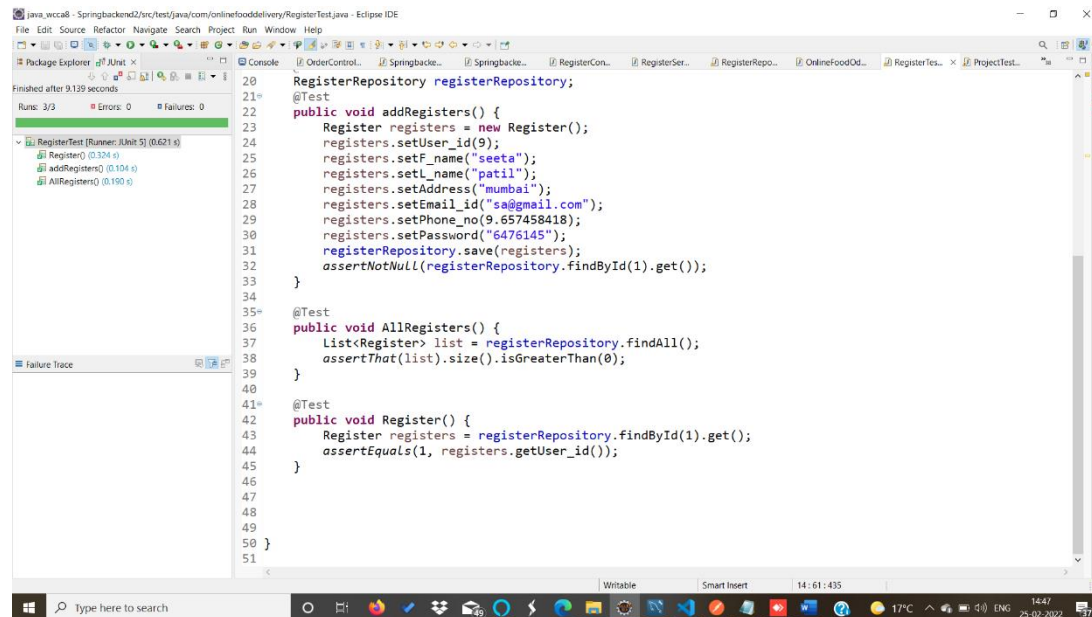


Fig B : Part of the home page of Online Food Ordering System

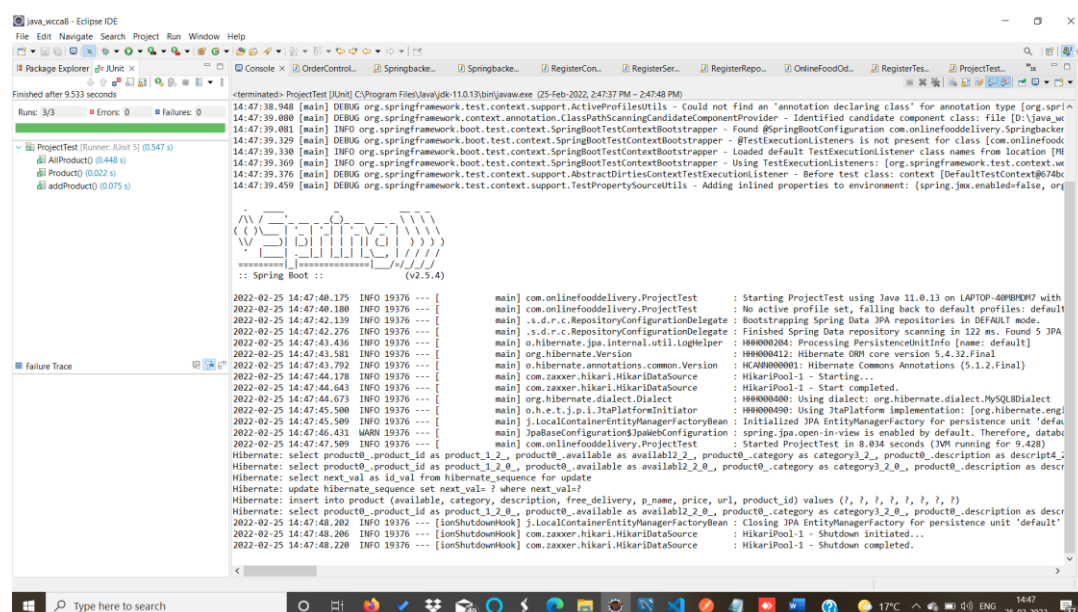
7. Unit Testing Output and Screenshots

A. Register Junit Test case



```
20 RegisterRepository registerRepository;
21 @Test
22 public void addRegisters() {
23     Register registers = new Register();
24     registers.setUser_id(9);
25     registers.setF_name("seeta");
26     registers.setL_name("patil");
27     registers.setAddress("mumbai");
28     registers.setEmail_id("sa@gmail.com");
29     registers.setPhone_no(9.657458418);
30     registers.setPassword("6476145");
31     registerRepository.save(registers);
32     assertNotNull(registerRepository.findById(1).get());
33 }
34
35 @Test
36 public void AllRegisters() {
37     List<Register> list = registerRepository.findAll();
38     assertEquals(list.size(), 1);
39 }
40
41 @Test
42 public void Register() {
43     Register registers = registerRepository.findById(1).get();
44     assertEquals(1, registers.getUser_id());
45 }
46
47
48
49
50 }
51
```

B. Product Test case

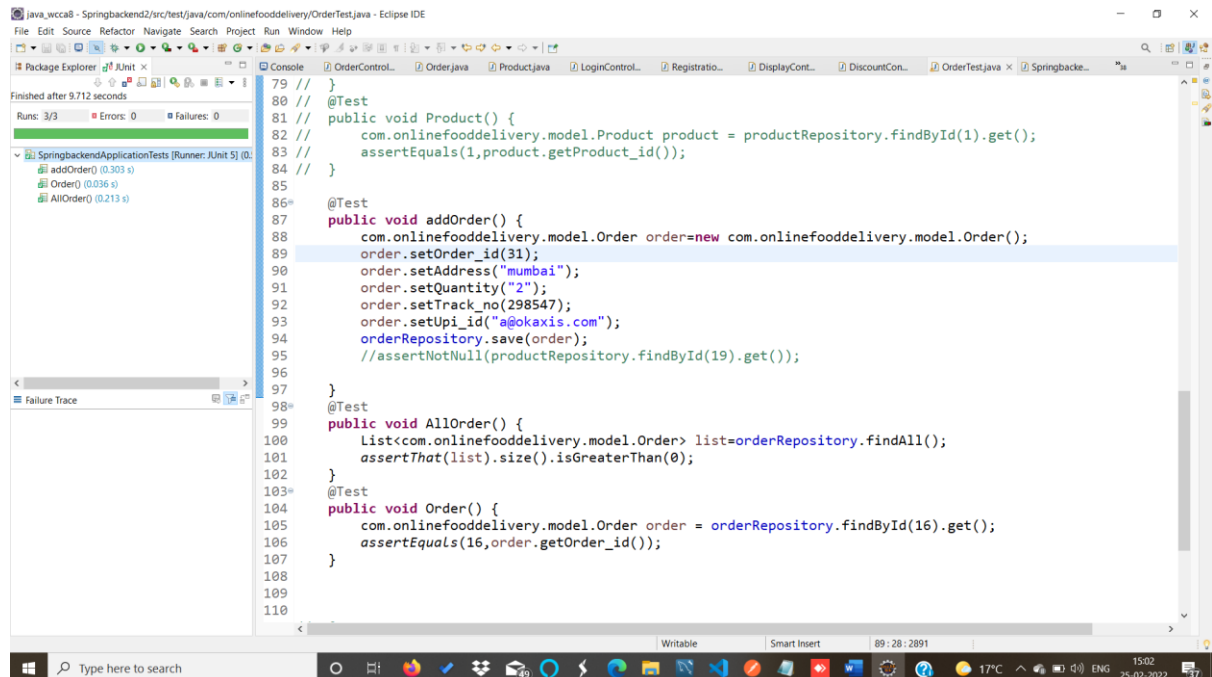


```
<terminated> ProjectTest [Unit: C:\Program Files\Java\jdk-11.0.13\bin\java.exe (25-Feb-2022 24:37 PM - 24:48 PM)]
14:47:38.548 [main] DEBUG org.springframework.test.context.support.ActiveProfilesUtils - Could not find an 'annotation declaring class' for annotation type [org.springframework.test.context.support.ActiveProfilesUtils].
14:47:39.080 [main] INFO org.springframework.context.annotation.ClassPathScanningCandidateComponentProvider - Identified candidate component class: file [D:\java\workspace\onlinefooddelivery\src\test\java\com\onlinefooddelivery\ProjectTest.class]
14:47:39.081 [main] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper - Found @SpringBootTestConfiguration on class [com.onlinefooddelivery.ProjectTest]
14:47:39.329 [main] DEBUG org.springframework.boot.test.context.SpringBootTestContextBootstrapper - @TestExecutionListeners is not present for class [com.onlinefooddelivery.ProjectTest]
14:47:39.330 [main] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper - Loaded default TestExecutionListener class names from location [M
14:47:39.369 [main] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper - Using TestExecutionListeners: [org.springframework.test.context.u
14:47:39.376 [main] DEBUG org.springframework.test.context.support.AbstractDirtiesContextTestExecutionListener - Before test class: context [DefaultTestContext@6074bc
14:47:39.459 [main] DEBUG org.springframework.test.context.support.TestPropertySourceUtils - Adding inlined properties to environment: (spring.jpa.enabled=false, org

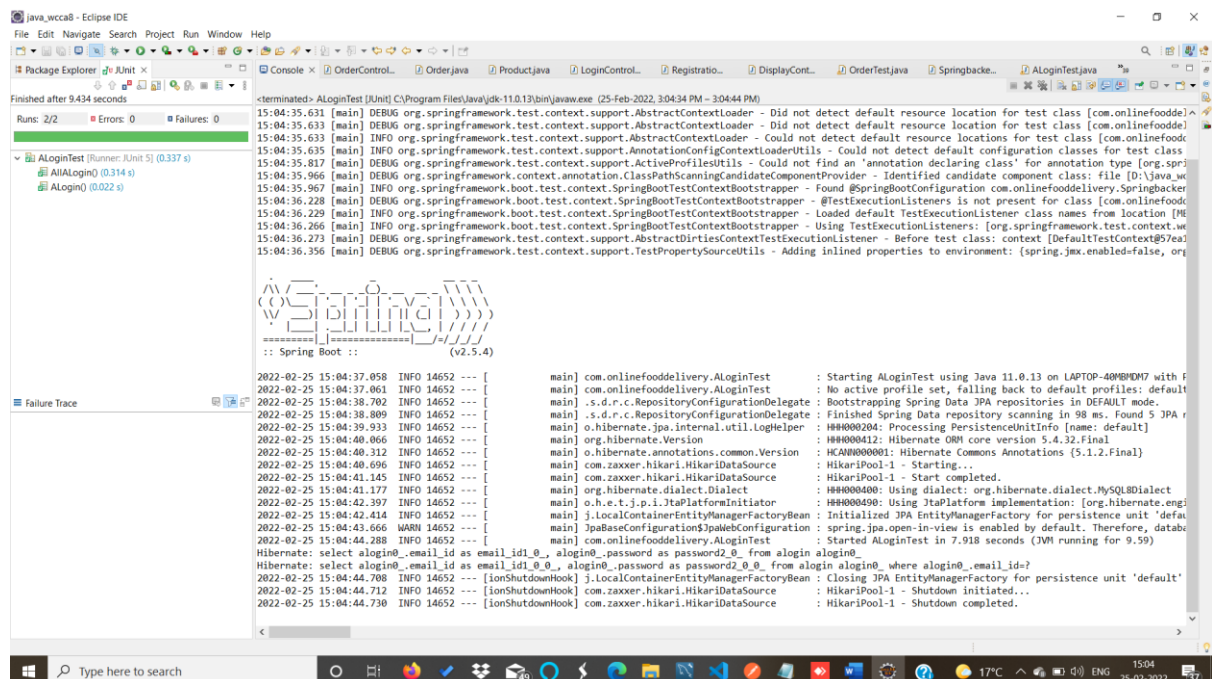
:: Spring Boot ::
(v2.5.4)

2022-02-25 14:47:40.175 INFO 19376 --- [main] com.onlinefooddelivery.ProjectTest : Starting ProjectTest using Java 11.0.13 on LAPTOP-60M8Q9M7 with
2022-02-25 14:47:40.180 INFO 19376 --- [main] com.onlinefooddelivery.ProjectTest : No active profile set, falling back to default profiles: default
2022-02-25 14:47:42.139 INFO 19376 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2022-02-25 14:47:42.276 INFO 19376 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 122 ms. Found 5 JPA
2022-02-25 14:47:43.436 INFO 19376 --- [main] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing PersistenceUnitInfo [name: default]
2022-02-25 14:47:43.581 INFO 19376 --- [main] org.hibernate.Version : HHH0000412: Hibernate ORM core version 5.4.32.Final
2022-02-25 14:47:43.792 INFO 19376 --- [main] o.hibernate.annotations.common.Version : HHA0000001: Hibernate Commons Annotations [5.1.2.Final]
2022-02-25 14:47:44.178 INFO 19376 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2022-02-25 14:47:44.643 INFO 19376 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2022-02-25 14:47:44.673 INFO 19376 --- [main] org.hibernate.dialect.Dialect : HHH0000400: Using dialect: org.hibernate.dialect.MySQL8Dialect
2022-02-25 14:47:45.580 INFO 19376 --- [main] o.h.e.t.j.p.i.TrasPlatformInitiator : HHH0000490: Using JPAPlatform implementation: [org.hibernate.engi
2022-02-25 14:47:45.589 INFO 19376 --- [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2022-02-25 14:47:46.431 WARN 19376 --- [main] jpaBaseConfigurationJpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, datab
2022-02-25 14:47:47.589 INFO 19376 --- [main] com.onlinefooddelivery.ProjectTest : Started ProjectTest in 8.834 seconds (JVM running for 9.428)
Hibernate: select product0_product_id as product1_2_0, product0_available as availab12_2_0, product0_category as category3_2_0, product0_description as descr
Hibernate: select product0_product_id as product1_2_0, product0_available as availab12_2_0, product0_category as category3_2_0, product0_description as descr
Hibernate: update hibernate_sequence set next_val= ? where next_val=?
Hibernate: insert into product (available, category, description, free_delivery, p_name, price, url, product_id) values (?, ?, ?, ?, ?, ?, ?, ?)
Hibernate: select product0_product_id as product1_2_0, product0_available as availab12_2_0, product0_category as category3_2_0, product0_description as descr
2022-02-25 14:47:48.282 INFO 19376 --- [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory for persistence unit 'default'
2022-02-25 14:47:48.286 INFO 19376 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown initiated...
2022-02-25 14:47:48.228 INFO 19376 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown completed.
```

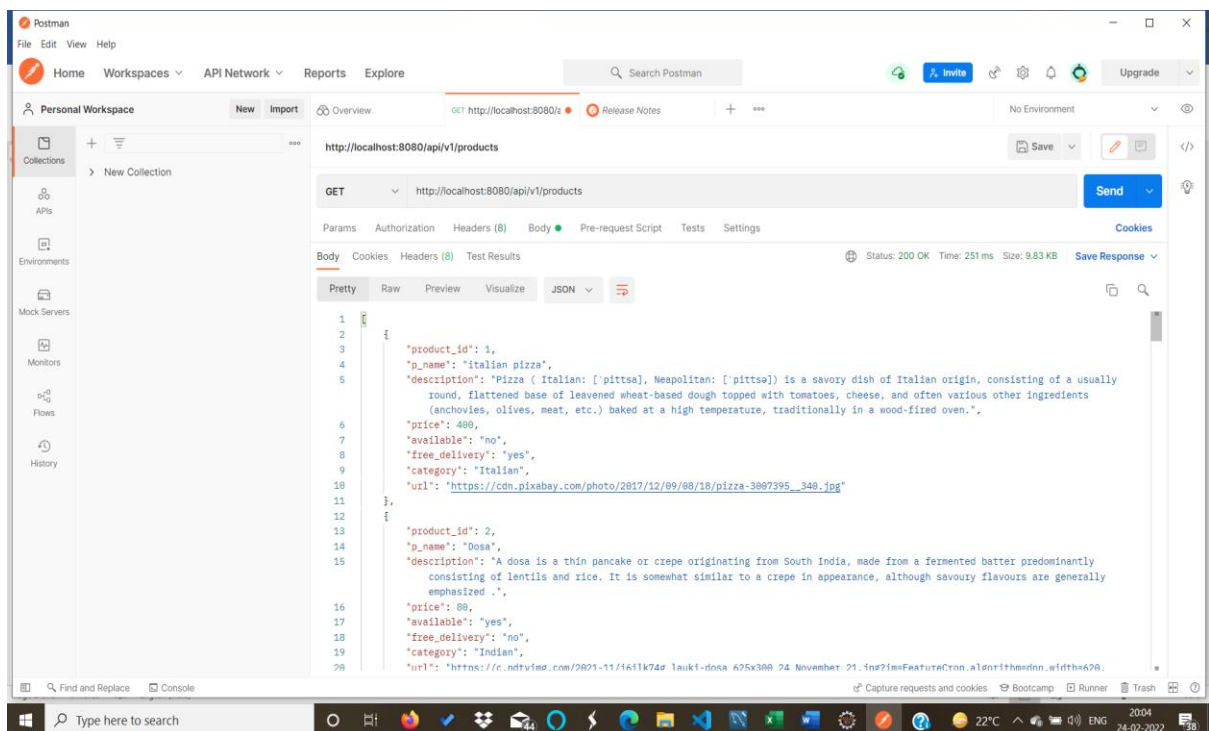
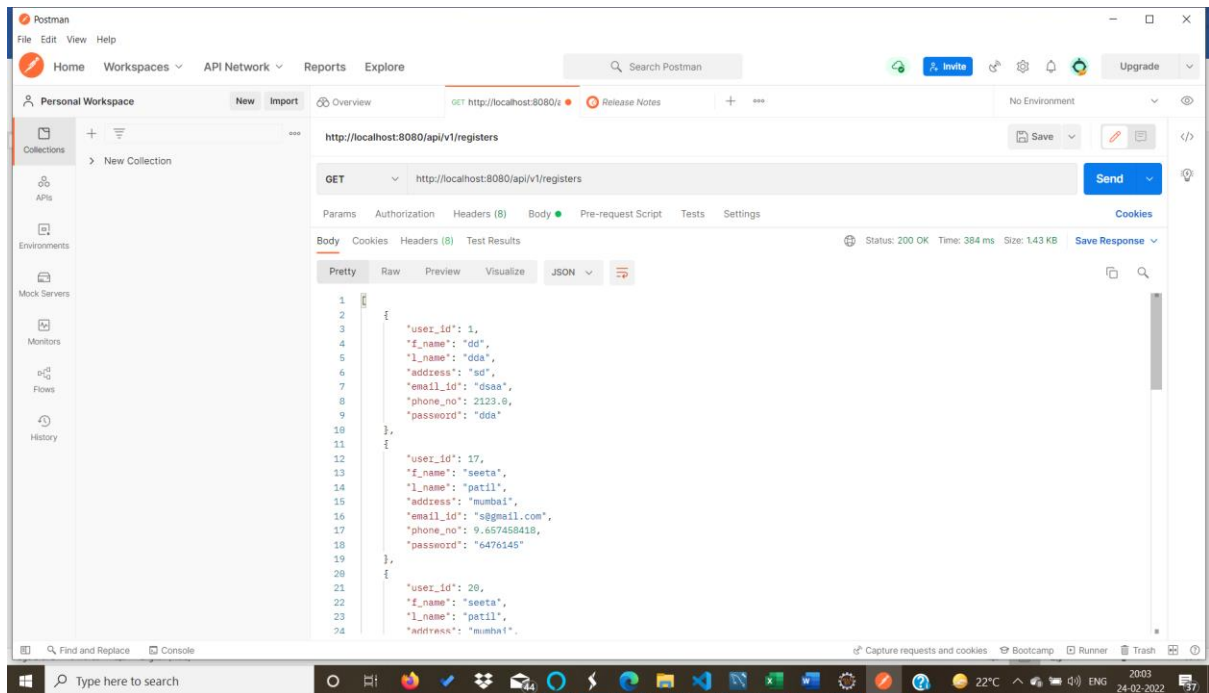
C.Order Test case



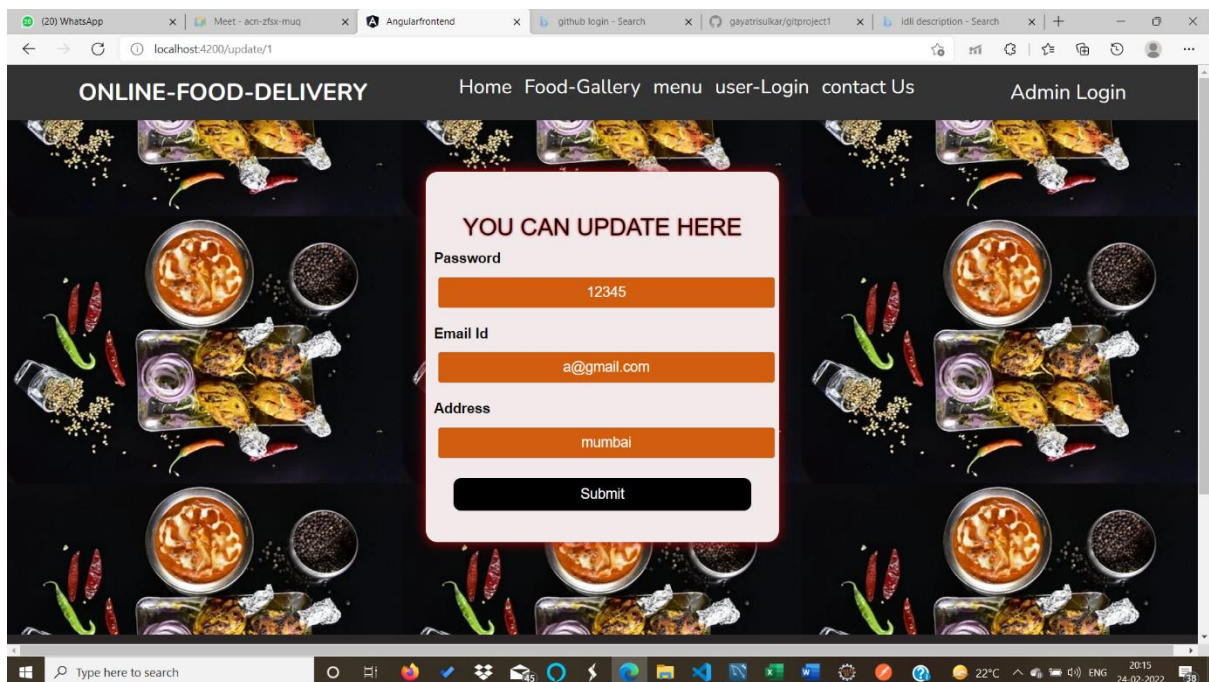
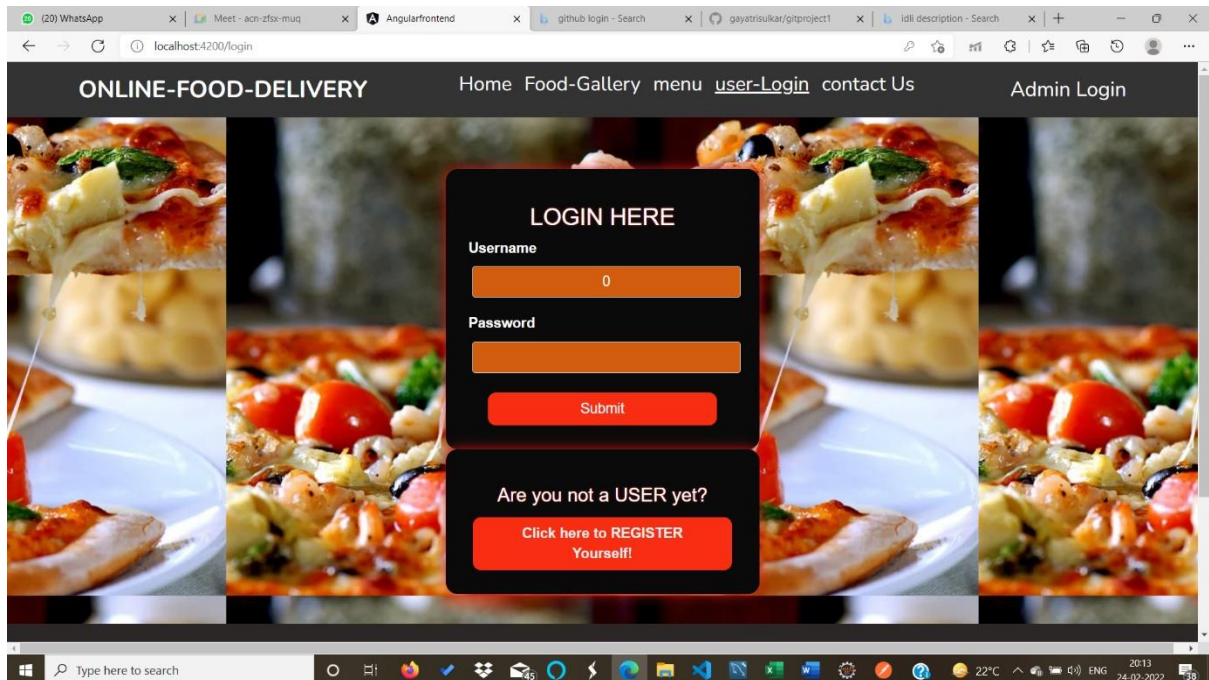
D.Login Test case

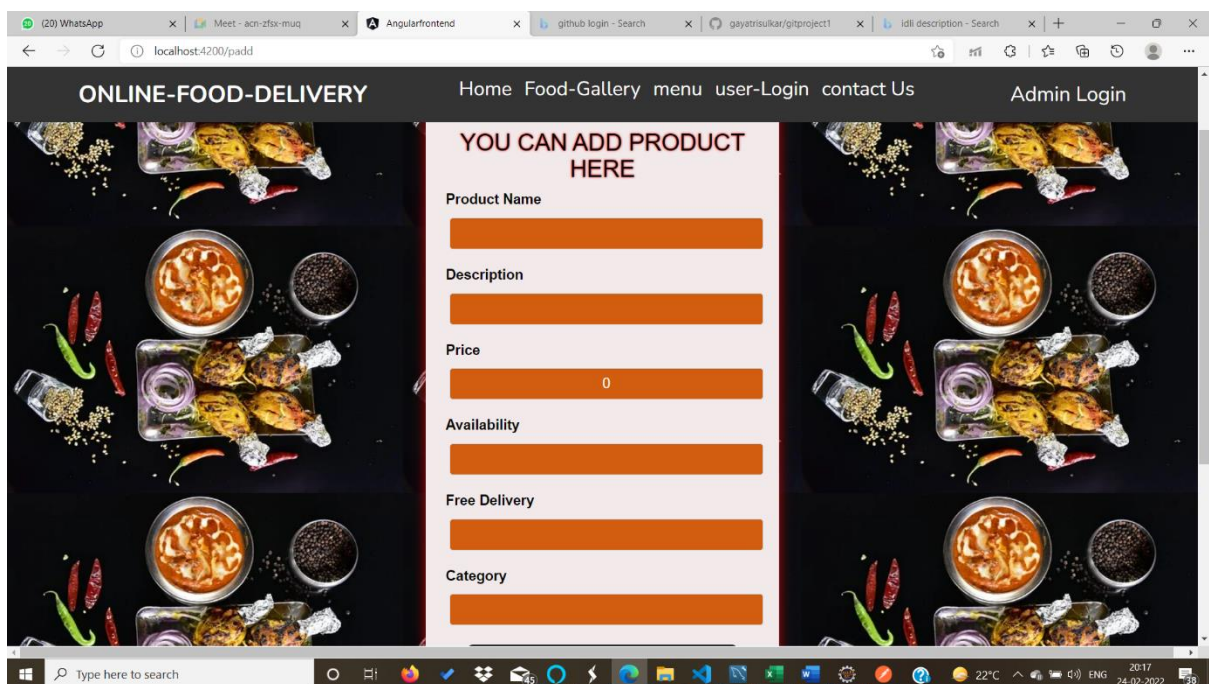
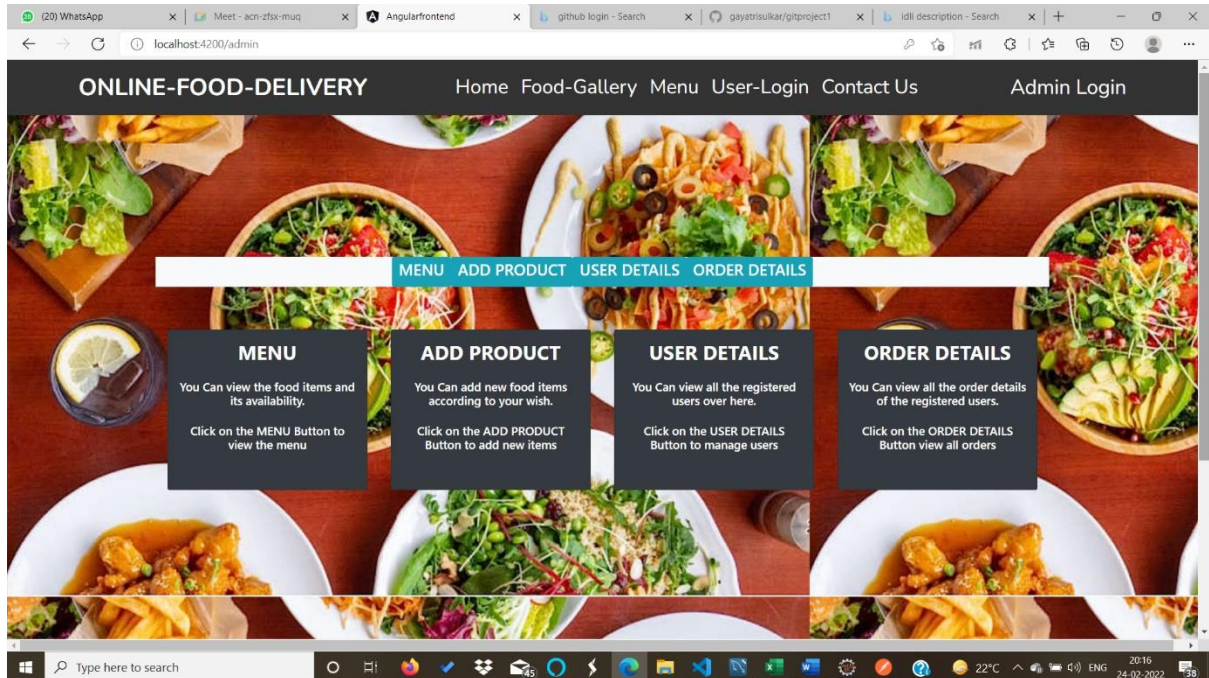


8.Functional Testing Screenshots (Postman Testing)



More Screenshots of User Interface :





9.CONCLUSION

- With online ordering on board you will enrichen your customer experience by making the process of ‘placing orders’ a lot easier. It will show that you value your customer’s time.
- Online ordering will guarantee a ‘level up’ to your web presence. And a good web presence will make you stand out in the search engine rankings and bring more customers to you.
- Online ordering will boost your productivity by eliminating the inefficient process of taking orders. It will help you to plan and implement an adaptive marketing campaign.
- Utilising the latest online ordering technology for your restaurant will also help you to tap into a massive customer base which is tech-savvy and believes in ‘online way’.