ONLINE FOOD DELIVARY SYSTEM

Team Lead:

Koppolu Venkata Teja

Frontend Lead:

Shrinath Shivaji Magdum

Backend Lead:

Koppolu Venkata Teja

Team Members:

1. Alugubelly Madhuri
2. Pidugu Pravalika
3. S Bindhu
4. Shrinath Shivaji Magdum
5. Devarinti Anusha
6. Kamsala Sai
7. Pandey Aishwarya Jyothi
8. Pragya Upadhyay
9. Reddem Himabindu
10. Koppolu Venkata Teja

Project Code	6
Project Name	Online Food Delivery System

Table of Contents

1. Introduction

It is known globally that, in today's market, it is extremely difficult to start a new small-scale business and live-through the competition from the well-established and settled owners. In fast paced time of today, when everyone is squeezed for time, the majority of people are finicky when it comes to placing a food order. The customers of today are not only attracted because placing an order online is very convenient but also because they have visibility into the items offered, price and extremely simplified navigation for the order.

Online ordering system that I am proposing here, greatly simplifies the ordering process for both the customer and the restaurant. System presents an interactive and up-to-date menu with all available options in an easy to use manner. Customer can choose one or more items to place an order which will land in the Cart. Customer can view all the order details in the cart before checking out. At the end, customer gets order confirmation details. Once the order is placed it is entered in the database and retrieved in pretty much real time. This allows Restaurant Employees to quickly go through the orders as they are received and process all orders efficiently and effectively with minimal delays and confusion.

1.1. 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 Angular, Spring-Boot, Hibernate, MySQL.

2. System Overview

The "**Online Food Delivery System**" should support basic functionalities (explained in section 2.1) for all below listed users.

- Administrator (A)
- Restaurants (R)
- Customer (c)

2.1. Authentication & Authorization

2.1.1 *Authentication*:

Any end-user should be authenticated using a unique user id and password.

2.1.2 Authorization

List of operations are done by Admin, Restaurant and Customers.

Admin:

- Admin can add Restaurants details with user name and password.
- Admin can edit and delete Restaurants details.

Restaurants:

- Restaurant can add their food category details.
- Restaurant can add their available food items with price.
- Restaurant can see Customers orders.
- Proceed for preparing food.
- Sending order to customers.

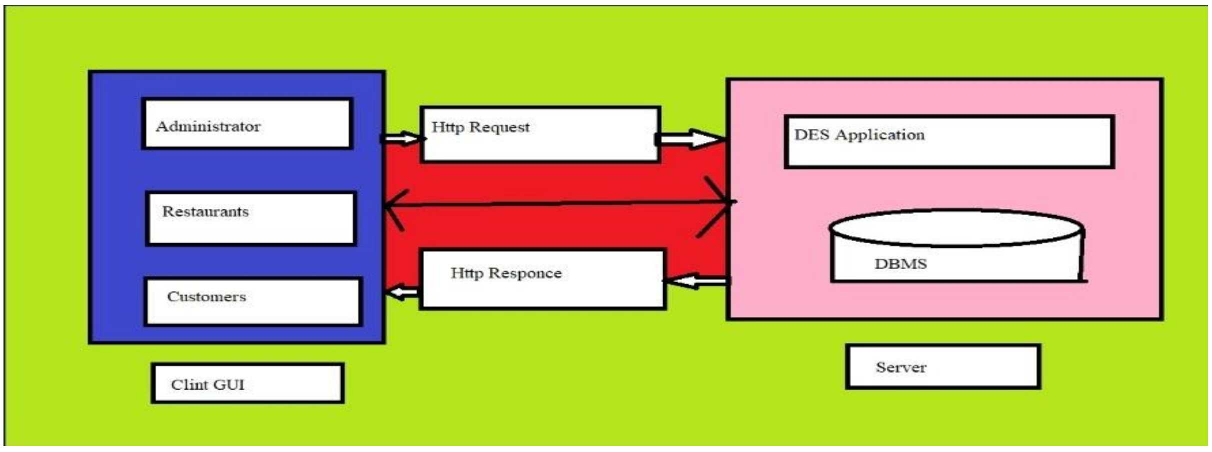
Customers:

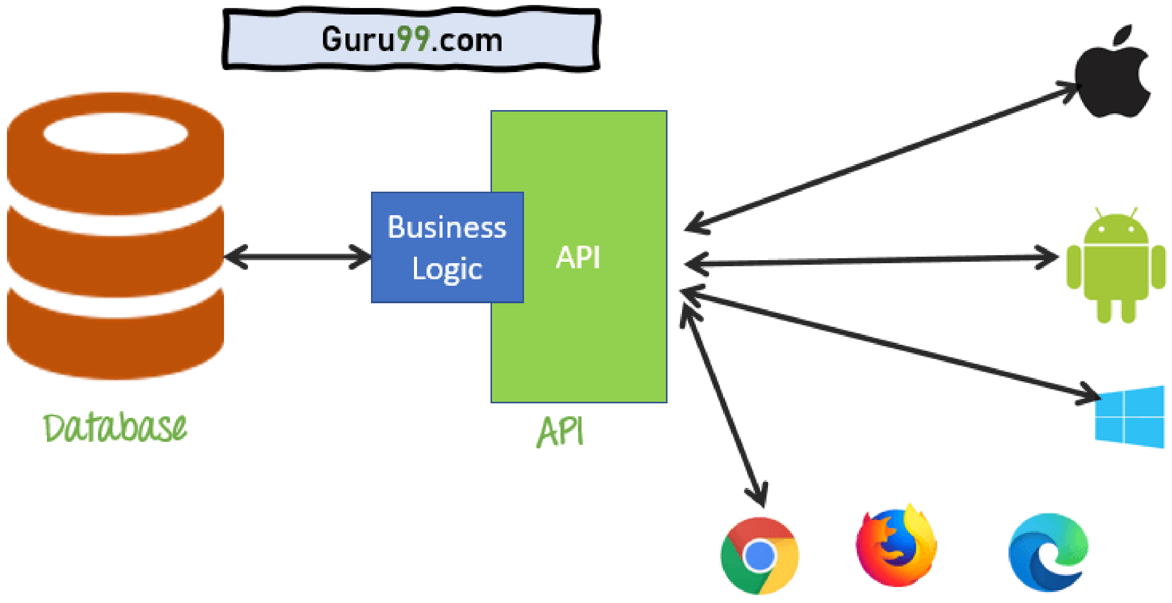
- Register their accounts in signup.
- After register successfully user can login.
- After login they can find numbers of Restaurants and select best one.
- Customers can see food items present in Restaurant.
- Add to cart or Order by proceeding amount.
- Customers can check their status once order a food.

2.2 Functional Flow

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

Ex. - Web Application.





2.3 Environment:

The system will be developed on any Windows OS machine using J2EE, Hibernate and Spring-Boot, MySQL.

- Intel hardware machine (PC P4-2.26 GHz, 512 MB RAM, 40 GB HDD)
- Server – Apache Tomcat 8 or higher
- Database – MySQL
- JRE 8
- Eclipse IDE or Spring Tool or IntelliJ IDEA.

FUNCTIONAL SPECIFICATION:

3. Sub-system Details:

The “**Online Food Delivery System**” should is defined, where in all users need to login successfully before performing any of their respective operations. Find below tables that provides functionality descriptions for each type of user / sub-system. Against each requirement, indicative data is listed in column ‘Data to include’. Further, suggested to add/modify more details wherever required with an approval from customer/faculty.

3.1 Administrator

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

ID	Objects	Operation	Data to include	Remarks
1	Manage Restaurant	Add, View, Delete, Modify	Restaurant name, Contact number, Email, Address. GST Number	

3.2 Restaurants:

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

ID	Objects	Operations	Data to include	Remarks
1	Manage Category	Add, view, Modify	Adding food Category	

2	Manage Product	Add, view, delete, Modify	Adding number of food items present in restaurants.	
3	New Orders	Showing Orders from customers	Orders from customers	
4	Preparing Orders	Prepare food	Proceed to prepare	
5	Completed Orders	Return to customers	Return to customer	

3.3 Customer:

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

ID	Objects	Operation	Data to include	Remarks
1	User	Register	Name, Contact Number, Email	
2	User	Login	Email, Password	
3	Restaurants	Choose Restaurant	View Product	
4	Product	Add to cart, Delete from cart, Delete all the products from cart, Procced to pay	Product id, Product Name, Price, Quantity, Total Price.	
5	Your Order	Check status of your food	Status	

FUNCTIONAL SPECIFICATION

3.3 Login| Logout:

{Web Application – Angular, Spring-Boot, Hibernate, MySQL}

- 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 Online Food Delivery 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: app user details (database name: appuser):

In this table we can find all users, Restaurant id's, Admin details we can find here.

Authentication, and authorization / privileges should be kept in one or more tables, as necessary and applicable.

Field Name	**Description**
AppuUser_pk	User id can automatically increase my system
Email	Admin email, Restaurant email, user email
Password	Give your password

4.2 Table: Restaurant Details (Database name: restro):

In this table contains all restaurant details.

Field Name	**Description**
Restro_pk	Restaurant id automatically genarate
Address	Location
Contact	We can add Contact Number
Gstno	We can add GST Numbers
Name	Restaurant name
App user_fk	This key taking from appuser database

4.3 Table: User Registration Details (Database name: userdata):

Field Name	**Description**
Id	Automatically generate by system
Name	Enter user name

ucontact	Enter contact number
Appuser_fk	Taking from app user data

4.4 Table: Food Category (database name: category):

Here we can add food category

Field Name	**Description**
Category_pk	System automatically generate
cname	Item name
Restro_fk	Taking from restro database

4.5 Table: Product (Database name: product):

Here we can get food items present in Restaurant

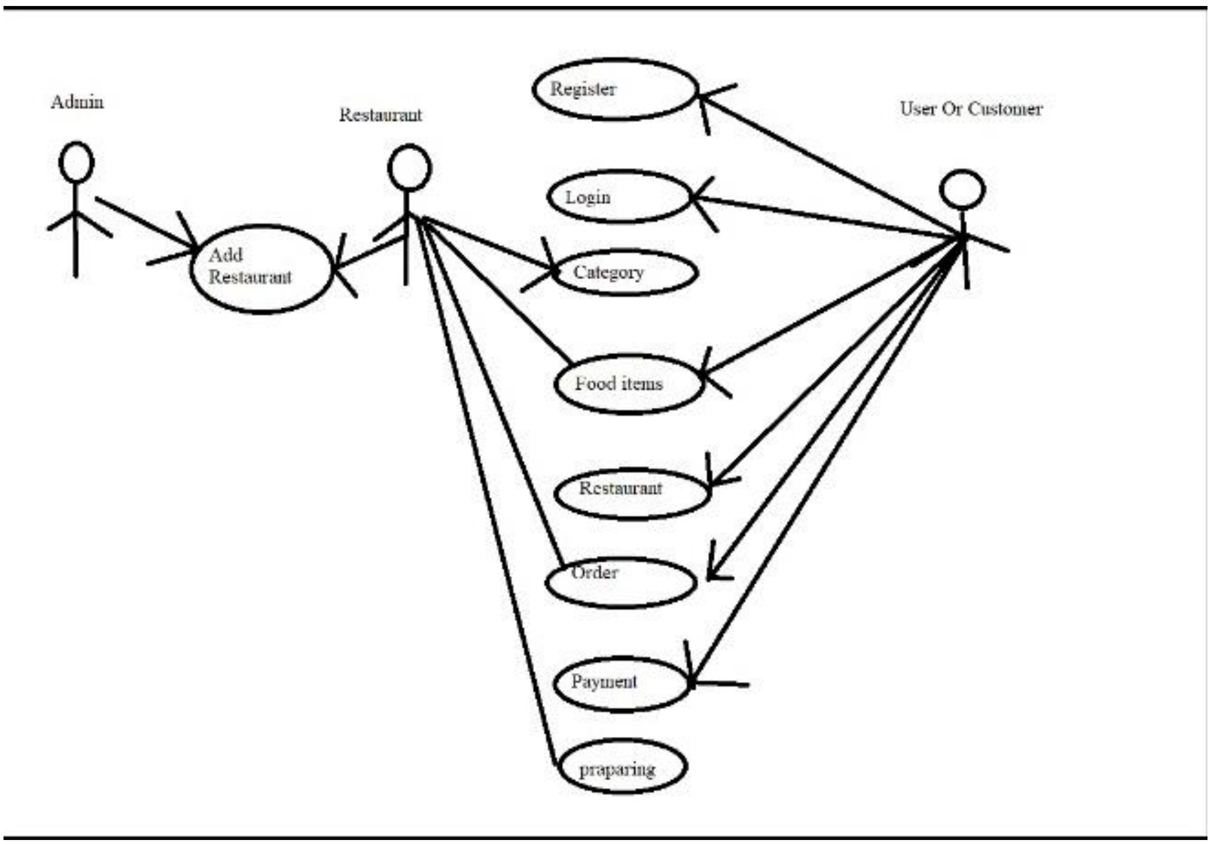
Field Name	**Description**
Id	System automatically genarate
Description	About food
Name	Food name
Price	Give price
Status	It will update automatically once ordered
Category_fk	Taking from category database

4.6 Table: User Purchase (Database name: userpurchase):

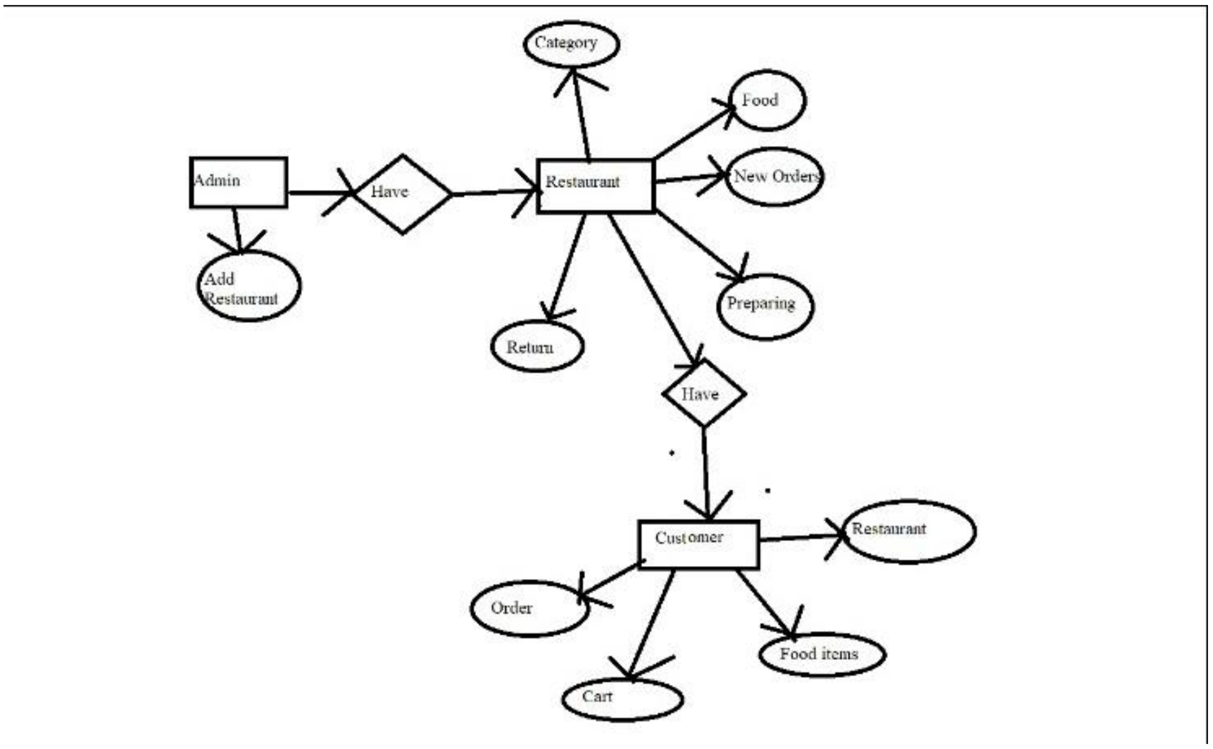
We can get in below table users transaction details.

Field Name	**Description**
Userpurchase_pk	System automatically genarate
Address	We can add address
Amount	Total amount
Pdata	It is json
Ptype	Use cash, credit card, debit card to pay
Status	We can get status
tranxid	After payment we can add transaction id
ucontact	User can add contact number
User_fk	Taking from app user database
Restro_fk	Taking from restro database

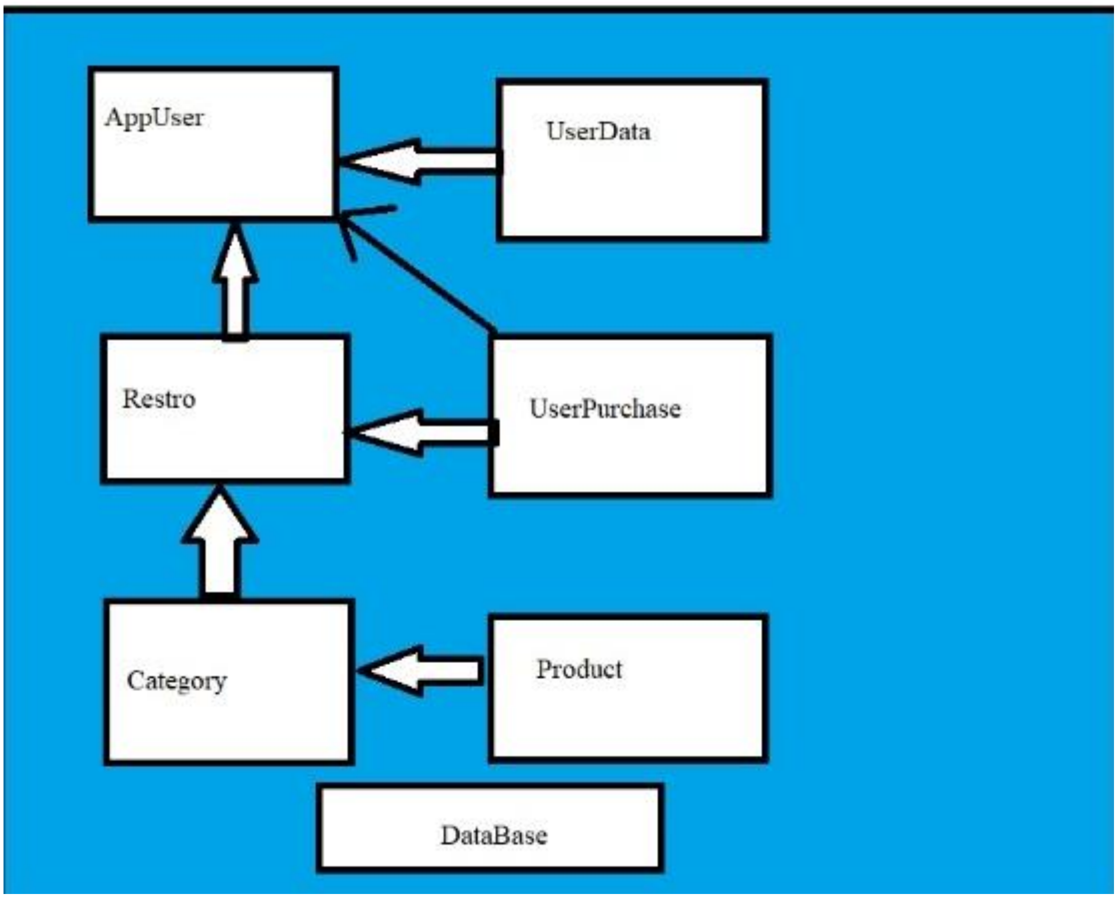
4.7 UML Diagram:



4.5 Entity Relationship Diagram:



4.6 Database Schema:



FUNCTIONAL SPECIFICATION

5 REST APIs to be Built.

Create following REST resources which are required in the application,

1. Creating **User** Entity: Create Spring Boot with Microservices Application with Spring Data JPA

Technology stack:

- Spring-Boot
- Spring REST

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.

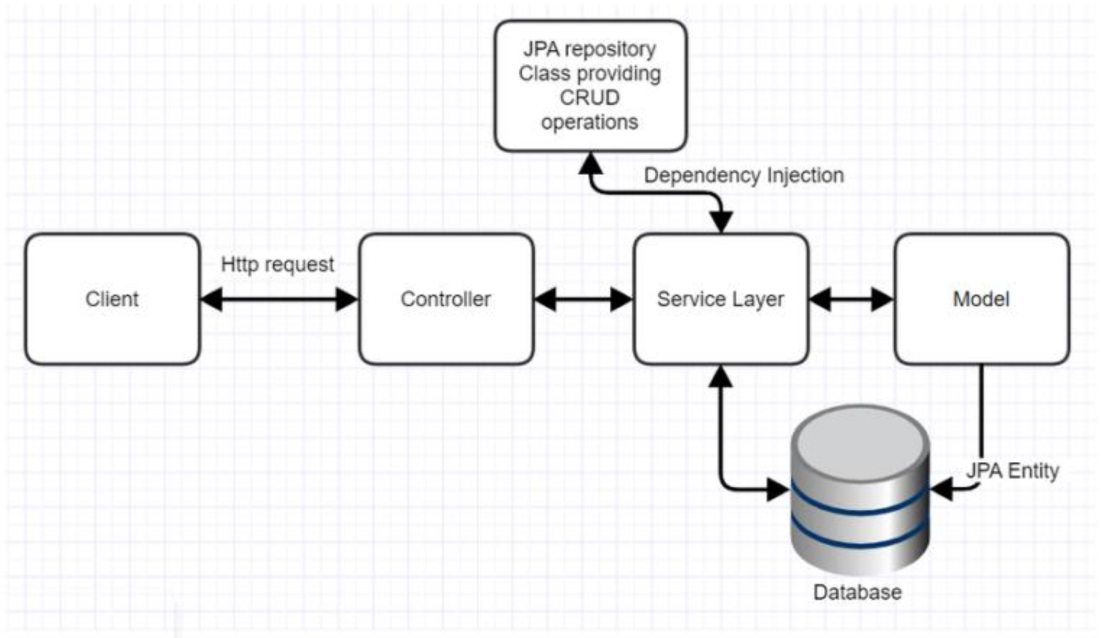


Fig A : Representation of Spring MVC pattern

5.2 Steps to create entities to perform Business logics:

1. create AppUser Entity:

1. In IntelliJ IDEA We need to create an Entity:(appuser).
2. By Creating a Appuser Repository 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 UserServiceImpl 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 URL’s

URL	Methods	Description	Format
localhost:8081/appuser/signup	POST	Give a single user description searched based on username	JSON
localhost:8081/appuser/login	POST	Give user and and password to login	JSON

2. Create restro Entity:

Creating **restro**Entity:

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

1. Create restro Entity:
2. By Creating a Appuser Repository interface and will make use of Spring Data JPA.
   a. addRestro method is use to adding restaurant.
   b. updateRestro method is used to update restaurant details.
3. By a creating restroServiceImpl class we can perform the required Business logics and exposes all services.
4. Finally, By creating a restroController class will handle all htpp requests and also will have fulling URL’s:

URL	Methods	Description	Format
localhost:8081/restro/addrestro	POST	Adding restaurant	JSON
localhost:8081/restro/getrestro	GET	Get all restaurants	Pretty
localhost:8081/restro/updaterestro	POST	Update restaurant details	JSON

3. **Create Category Entity:**

Creating **Category** Entity:

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

1. Create an Entity: category
2. Create a CategoryRepository interface and will make use of Spring Data JPA
   a. addcategory method used to add food item type.
   b. getCategory method is used to get category types.
   c. updateCategory method used toupdate food item type.

3. Create a categoryServiceImpl class and will expose all these services.
4. Finally, create a categeryController will have the following Uri's:

URL	Methods	Description	Format
localhost:8081/category/addcategory	POST	Adding category	JSON
localhost:8081/category/getcategory/1	GET	Showing no of restaurants present	Pretty
localhost:8081/category/updatecategory	POST	Adding category	JSON

4. **Create Product Entity:**

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:

2. Create an Entity: Product
3. Create a ProductRepository interface and will make use of Spring Data JPA
   b. UpdateProductStatus method used to update status.
   b. getProductsByCategory used to select the category
   c. getproductById method used to getting food item.
   d. getProductCountByName method is used to get name of food item
4. Create a ProductServiceImpl class and will expose all these services.
5. Finally, create a ProductController will have the following Uri's:

URL	Methods	Description	Format
localhost:8081/product/addproduct	POST	Adding food item	JSON
localhost:8081/product/getallproduct/1?status=true	GET	Getting food items	Pretty

5. **Create userpurchase Entity:**

Creating **userpurchase** Entity:

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

1.Create an Entity: Product

2. Create a userpurchaseRepository interface and will make use of Spring Data JPA .

a. Purchase method used to purchase the food .
b. getrecord used to get all details.
c. updatestatus method used toupdate ordered status.

3.Create a userpurcharseServiceImpl class and will expose all these services.

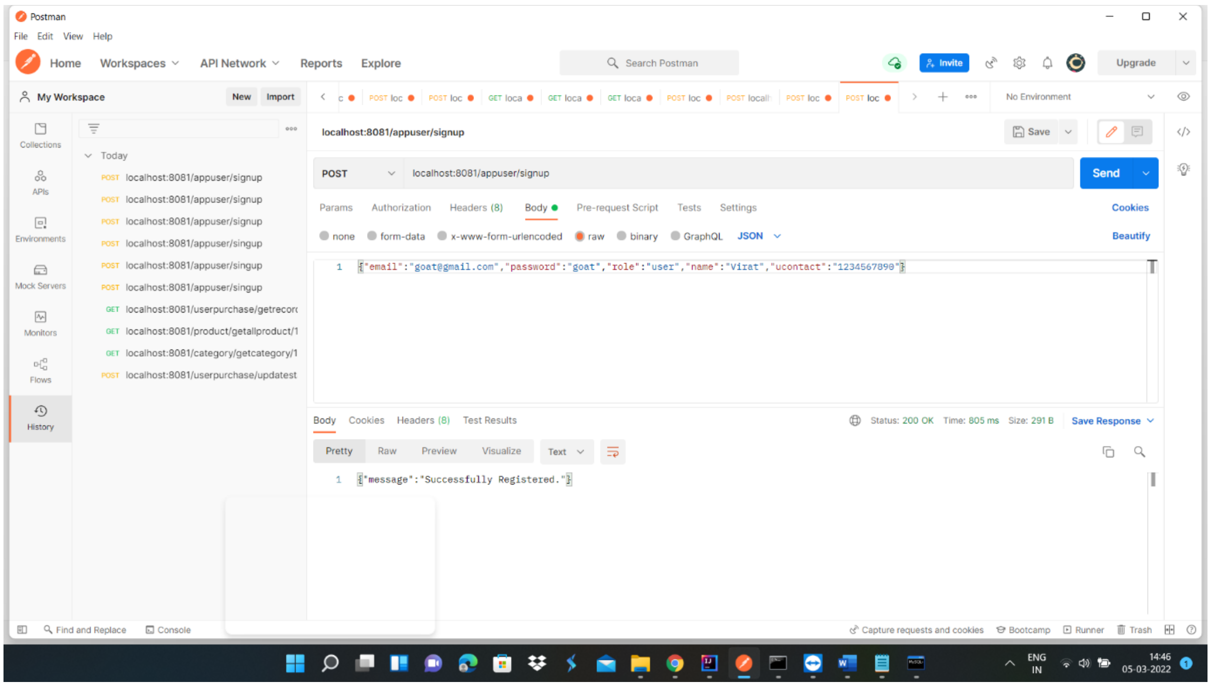
4.Finally, create a UserpurchaseController will have the following Uri's:

URL	Methods	Description	JSON
localhost:8081/userpurchase/purchase	POST	Ordering food items	Json
localhost:8081/userpurchase/getrecord?userId=1&status= Order Placed	GET	Getting order placed details	Pretty
localhost:8081/userpurchase/getrecord?userId=1	GET	Getting order details	Pretty
localhost:8081/userpurchase/updatestatus	POST	Updating order Status	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.

7.Functional Postman Testing:



POST localhost:8081/appuser/signup

```
{"email":"goat@gmail.com","password":"goat","role":"user","name":"Virat","ucontact":"1234567890"}
```

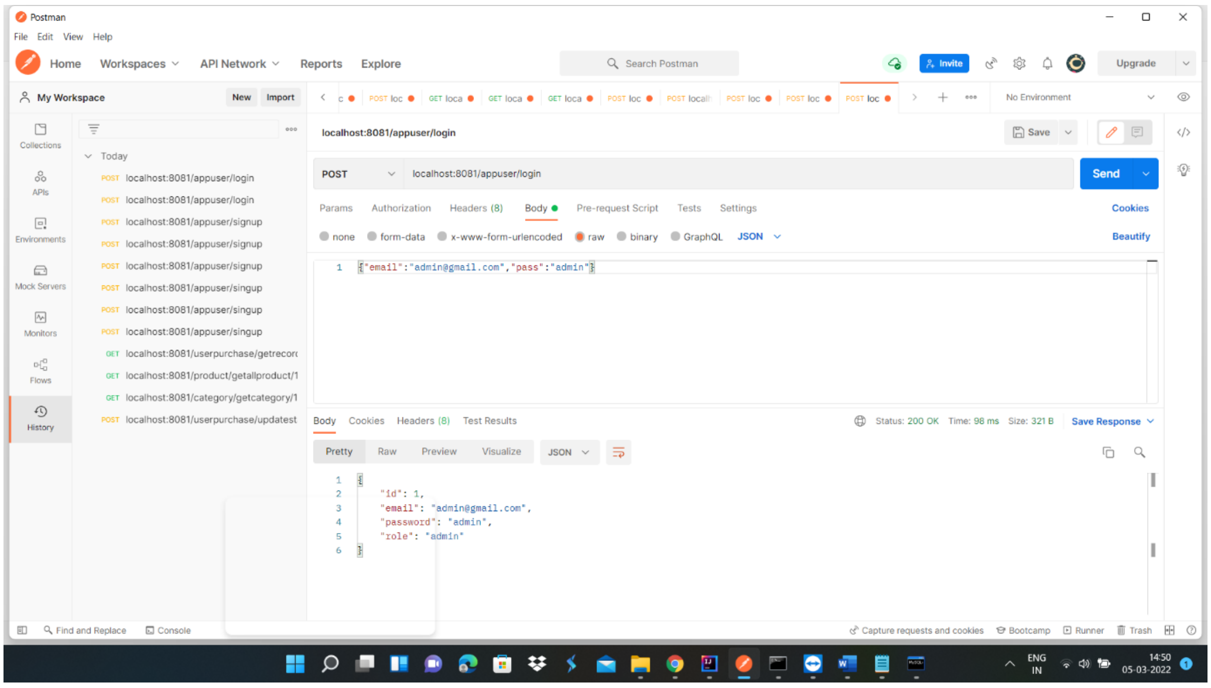
Status: 200 OK Time: 805 ms Size: 291 B

```
{"message":"Successfully Registered."}
```

History – Today:

```
POST localhost:8081/appuser/signup
POST localhost:8081/appuser/signup
POST localhost:8081/appuser/signup
POST localhost:8081/appuser/singup
POST localhost:8081/appuser/singup
POST localhost:8081/appuser/singup
GET  localhost:8081/userpurchase/getrecor
GET  localhost:8081/product/getallproduct/1
GET  localhost:8081/category/getcategory/1
POST localhost:8081/userpurchase/updatest
```

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).



My Workspace — History — Today

```
POST localhost:8081/appuser/login
POST localhost:8081/appuser/login
POST localhost:8081/appuser/signup
POST localhost:8081/appuser/signup
POST localhost:8081/appuser/signup
POST localhost:8081/appuser/singup
POST localhost:8081/appuser/singup
POST localhost:8081/appuser/singup
GET  localhost:8081/userpurchase/getrecord
GET  localhost:8081/product/getallproduct/1
GET  localhost:8081/category/getcategory/1
POST localhost:8081/userpurchase/updatest
```

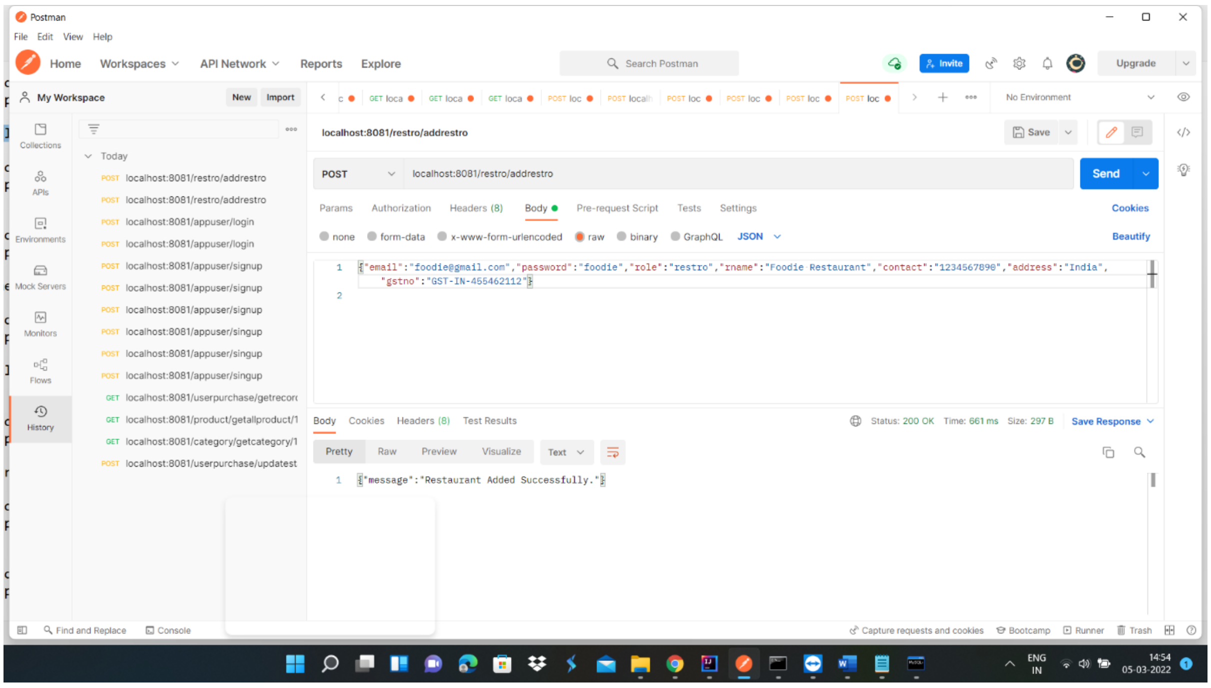
POST localhost:8081/appuser/login

Body (raw, JSON):

```
{"email":"admin@gmail.com","pass":"admin"}
```

Status: 200 OK Time: 98 ms Size: 321 B

```
{
    "id": 1,
    "email": "admin@gmail.com",
    "password": "admin",
    "role": "admin"
}
```



My Workspace — History — Today

```
POST localhost:8081/restro/addrestro
POST localhost:8081/restro/addrestro
POST localhost:8081/appuser/login
POST localhost:8081/appuser/login
POST localhost:8081/appuser/signup
POST localhost:8081/appuser/signup
POST localhost:8081/appuser/signup
POST localhost:8081/appuser/singup
POST localhost:8081/appuser/singup
POST localhost:8081/appuser/singup
GET  localhost:8081/userpurchase/getrecord
GET  localhost:8081/product/getallproduct/1
GET  localhost:8081/category/getcategory/1
POST localhost:8081/userpurchase/updatest
```

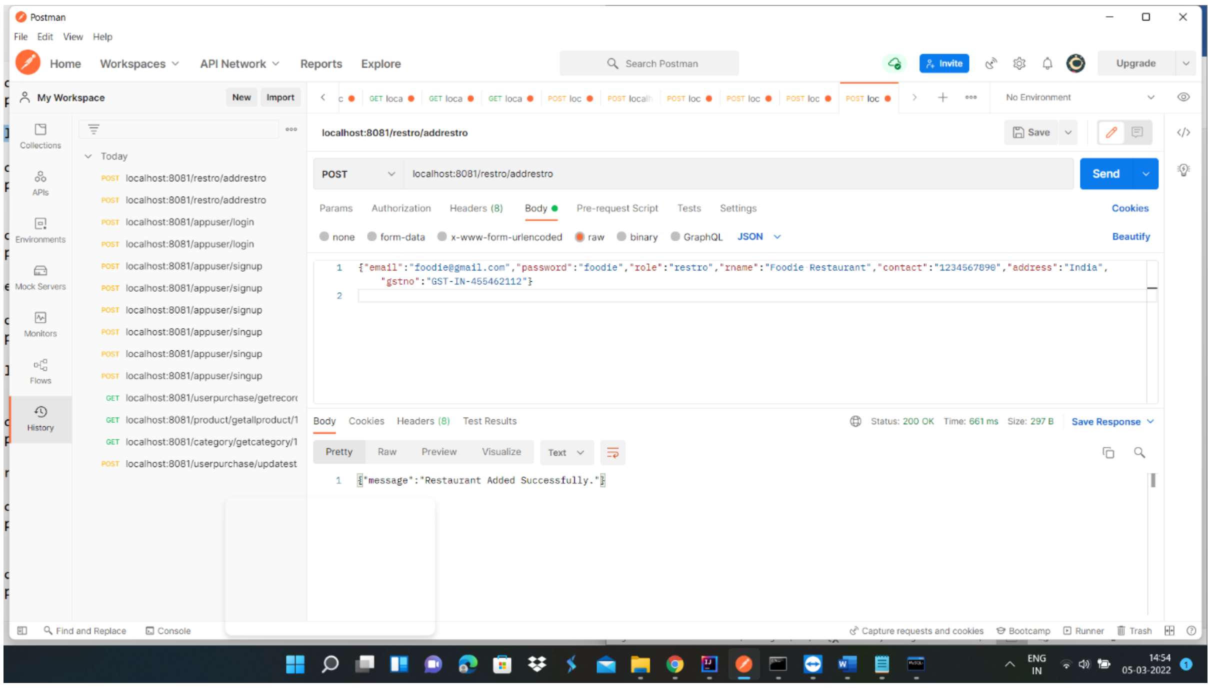
POST localhost:8081/restro/addrestro

Body (raw, JSON):

```
{"email":"foodie@gmail.com","password":"foodie","role":"restro","rname":"Foodie Restaurant","contact":"1234567890","address":"India",
    "gstno":"GST-IN-455462112"}
```

Status: 200 OK Time: 661 ms Size: 297 B

```
{"message":"Restaurant Added Successfully."}
```


localhost:8081/restro/addrestro

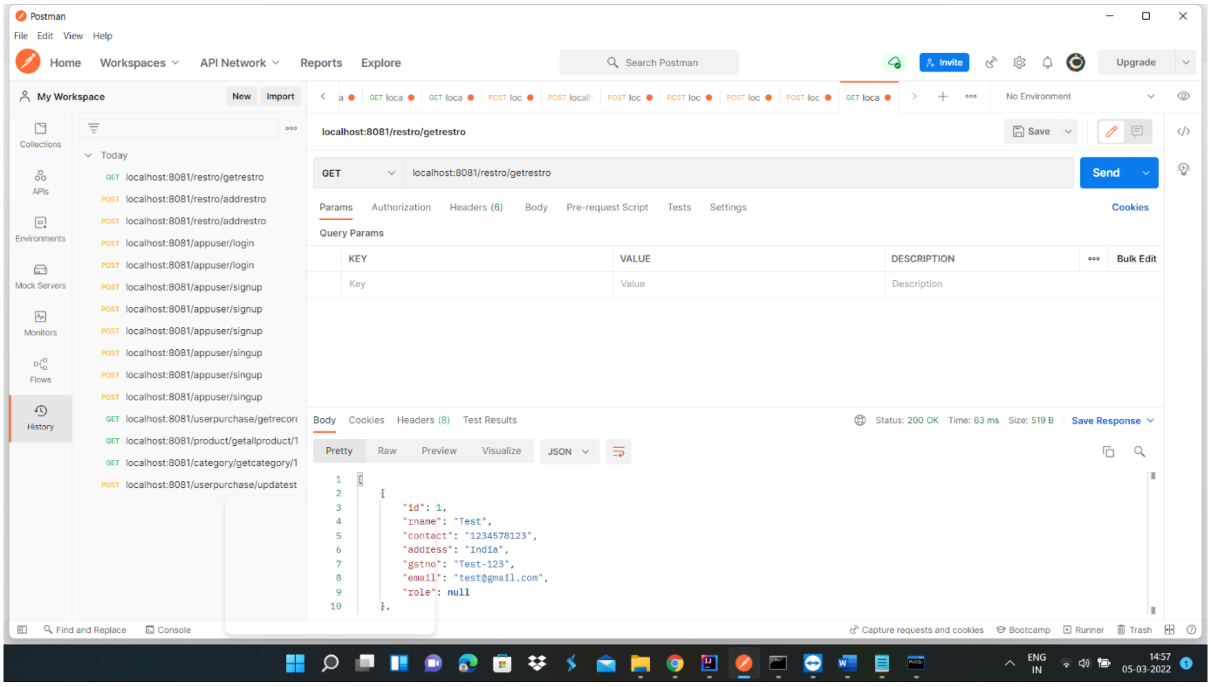
POST localhost:8081/restro/addrestro

```
{"email":"foodie@gmail.com","password":"foodie","role":"restro","rname":"Foodie Restaurant","contact":"1234567890","address":"India",
   "gstno":"GST-IN-455462112"}
```

Status: 200 OK Time: 661 ms Size: 297 B

```
{"message":"Restaurant Added Successfully."}
```

History – Today:
- POST localhost:8081/restro/addrestro
- POST localhost:8081/restro/addrestro
- POST localhost:8081/appuser/login
- POST localhost:8081/appuser/login
- POST localhost:8081/appuser/signup
- POST localhost:8081/appuser/signup
- POST localhost:8081/appuser/signup
- POST localhost:8081/appuser/singup
- POST localhost:8081/appuser/singup
- POST localhost:8081/appuser/singup
- GET localhost:8081/userpurchase/getrecorc
- GET localhost:8081/product/getallproduct/1
- GET localhost:8081/category/getcategory/1
- POST localhost:8081/userpurchase/updatest

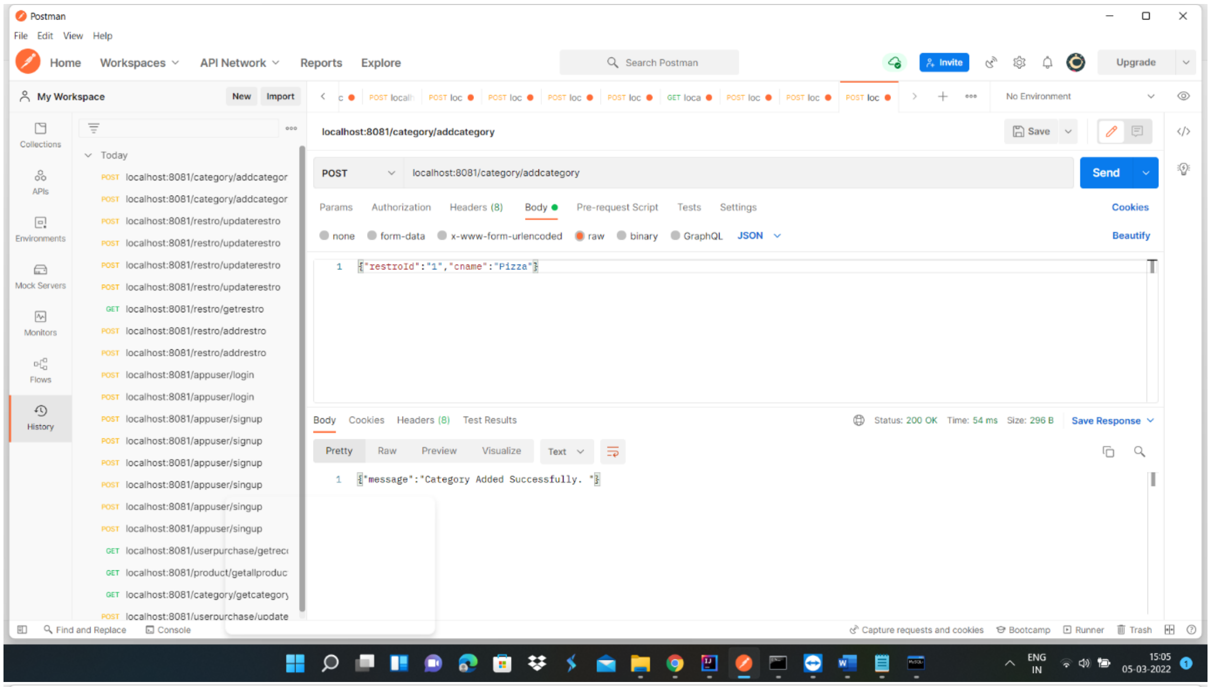

localhost:8081/restro/getrestro

GET localhost:8081/restro/getrestro

Status: 200 OK Time: 63 ms Size: 519 B

```
[
    {
        "id": 1,
        "rname": "Test",
        "contact": "1234578123",
        "address": "India",
        "gstno": "Test-123",
        "email": "test@gmail.com",
        "role": null
    },
```

History – Today:
- GET localhost:8081/restro/getrestro
- POST localhost:8081/restro/addrestro
- POST localhost:8081/restro/addrestro
- POST localhost:8081/appuser/login
- POST localhost:8081/appuser/login
- POST localhost:8081/appuser/signup
- POST localhost:8081/appuser/signup
- POST localhost:8081/appuser/signup
- POST localhost:8081/appuser/singup
- POST localhost:8081/appuser/singup
- POST localhost:8081/appuser/singup
- GET localhost:8081/userpurchase/getrecorc
- GET localhost:8081/product/getallproduct/1
- GET localhost:8081/category/getcategory/1
- POST localhost:8081/userpurchase/updatest



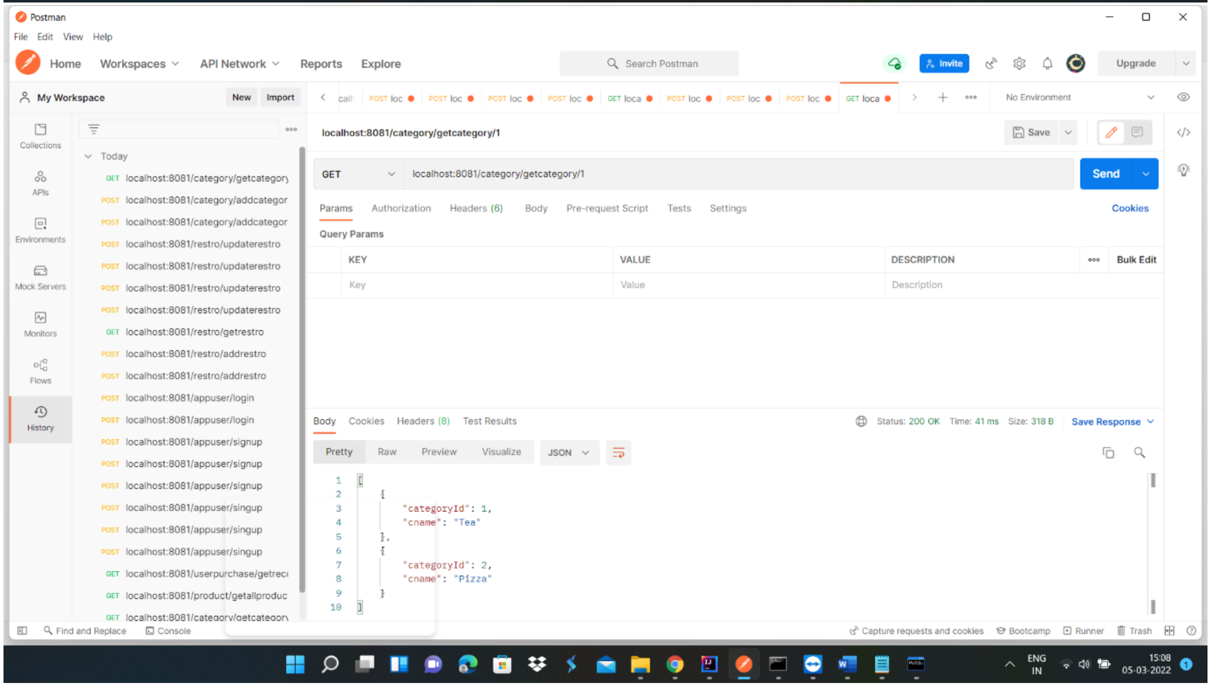
localhost:8081/category/addcategory

POST localhost:8081/category/addcategory

```
{"restroId":"1","cname":"Pizza"}
```

Status: 200 OK Time: 54 ms Size: 296 B

```
{"message":"Category Added Successfully. "}
```

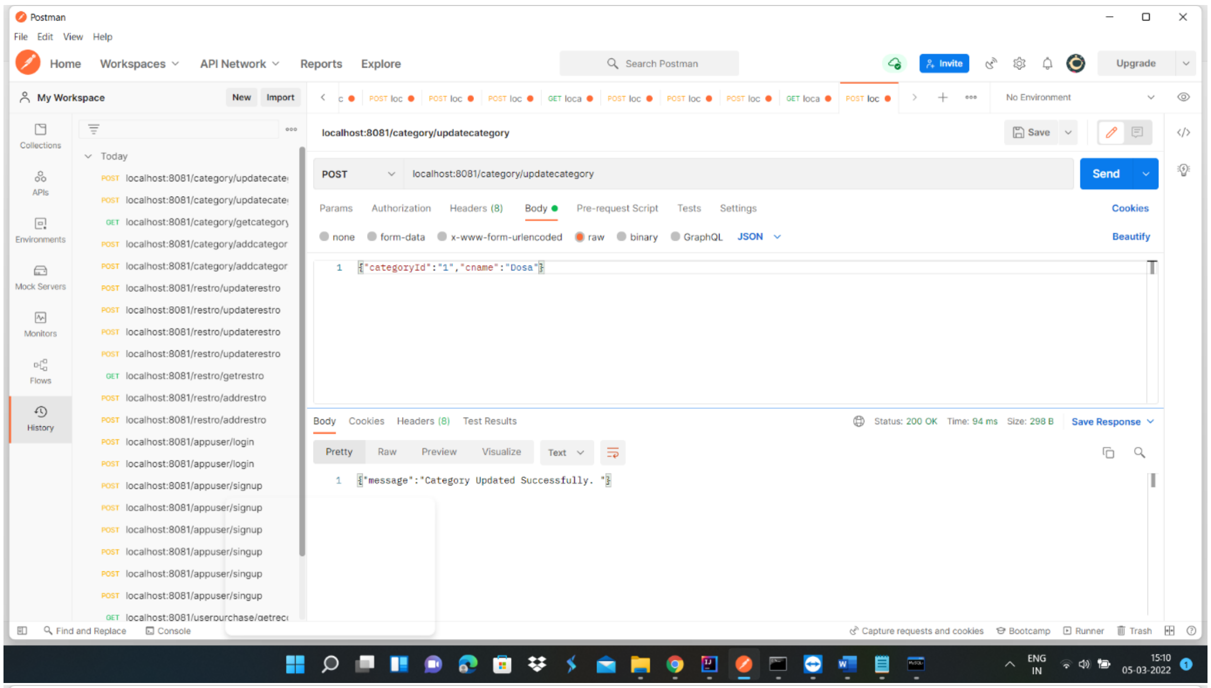


localhost:8081/category/getcategory/1

GET localhost:8081/category/getcategory/1

Status: 200 OK Time: 41 ms Size: 318 B

```
[
    {
        "categoryId": 1,
        "cname": "Tea"
    },
    {
        "categoryId": 2,
        "cname": "Pizza"
    }
]
```



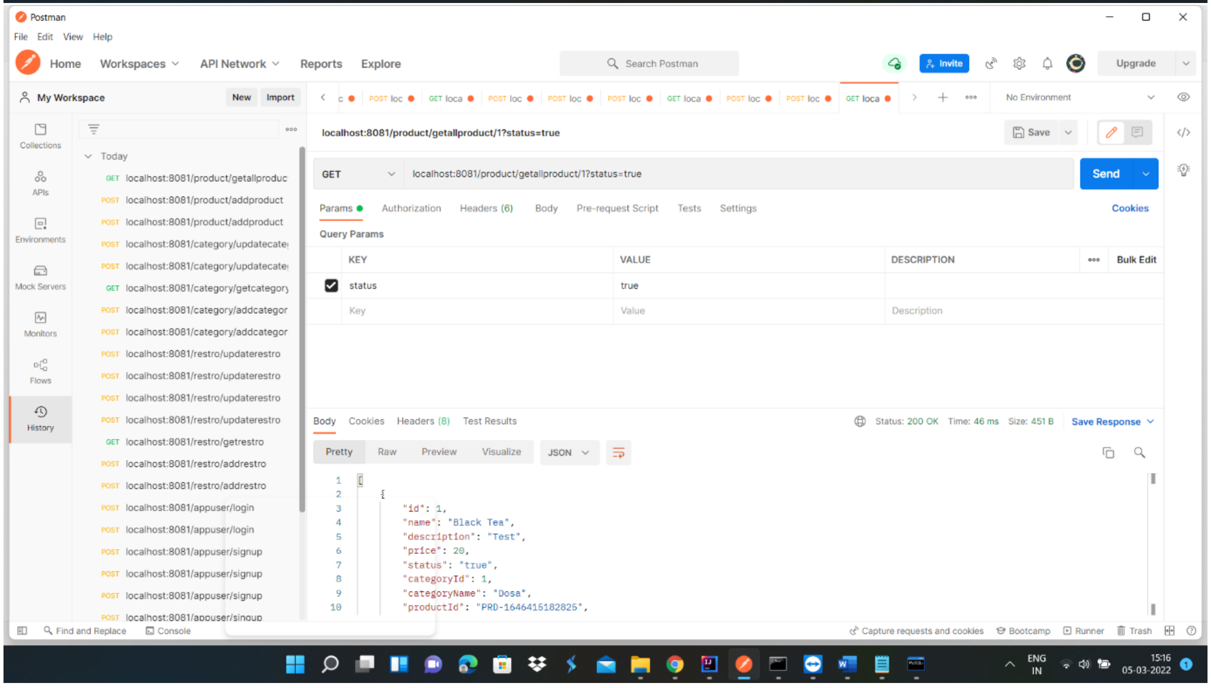
POST localhost:8081/category/updatecategory

Body (raw, JSON):

```
{"categoryId":"1","cname":"Dosa"}
```

Status: 200 OK Time: 94 ms Size: 298 B

```
{"message":"Category Updated Successfully. "}
```



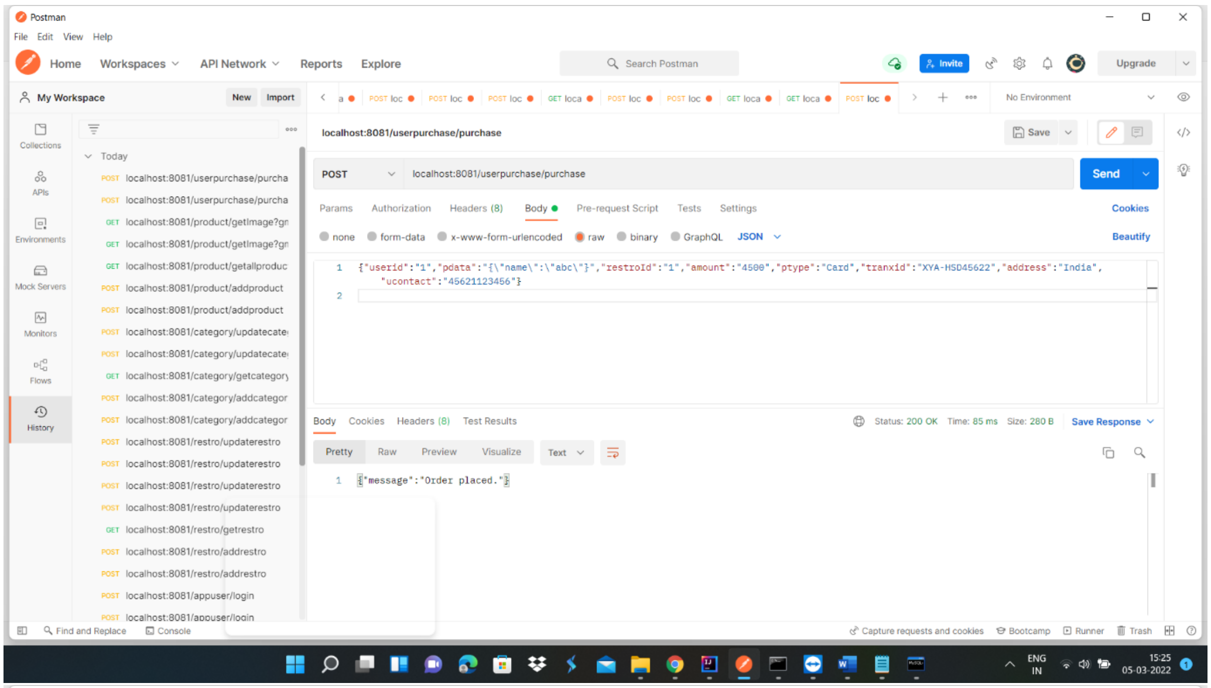
GET localhost:8081/product/getallproduct/1?status=true

Query Params

KEY	VALUE	DESCRIPTION
status	true	

Status: 200 OK Time: 46 ms Size: 451 B

```
[
    {
        "id": 1,
        "name": "Black Tea",
        "description": "Test",
        "price": 20,
        "status": "true",
        "categoryId": 1,
        "categoryName": "Dosa",
        "productId": "PRD-1646415182825",
```



POST localhost:8081/userpurchase/purchase

Body (raw, JSON):

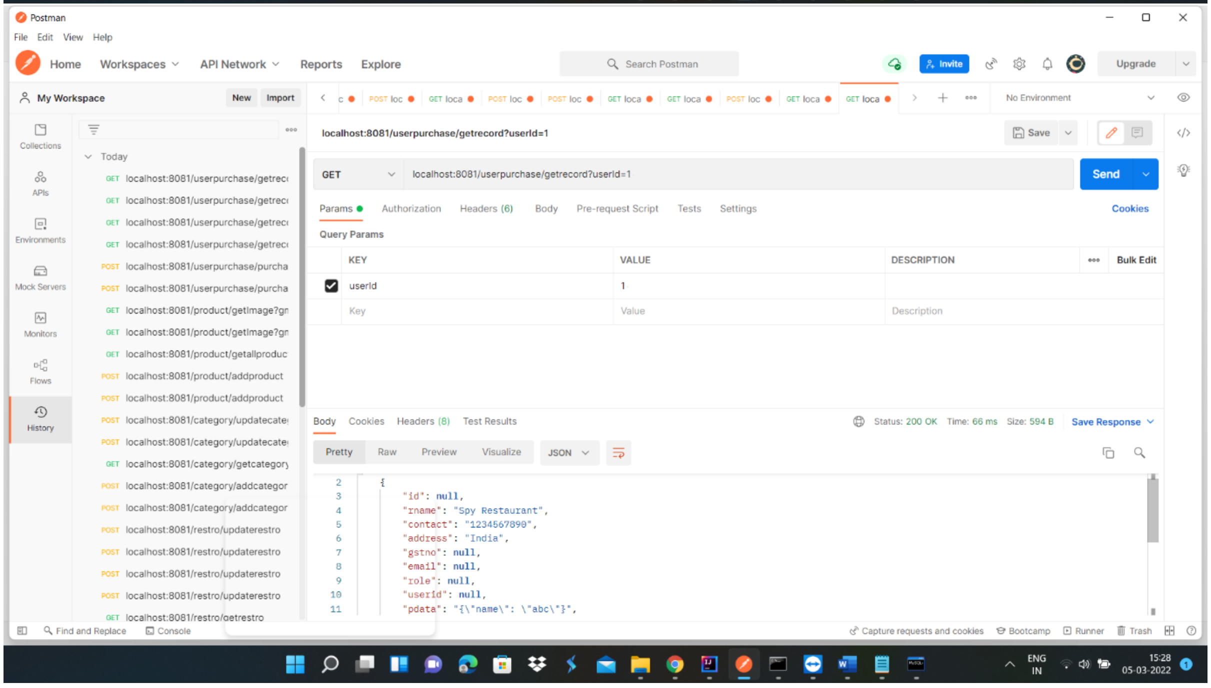
```
{"userid":"1","pdata":"{\"name\":\"abc\"}","restroId":"1","amount":"4500","ptype":"Card","tranxid":"XYA-HSD45622","address":"India",
   "ucontact":"45621123456"}
```

Status: 200 OK Time: 85 ms Size: 280 B

```
{"message":"Order placed."}
```

History (Today):

- POST localhost:8081/userpurchase/purcha
- POST localhost:8081/userpurchase/purcha
- GET localhost:8081/product/getImage?gn
- GET localhost:8081/product/getImage?gn
- GET localhost:8081/product/getallproduc
- POST localhost:8081/product/addproduct
- POST localhost:8081/product/addproduct
- POST localhost:8081/category/updatecate
- POST localhost:8081/category/updatecate
- GET localhost:8081/category/getcategory
- POST localhost:8081/category/addcategor
- POST localhost:8081/category/addcategor
- POST localhost:8081/restro/updaterestro
- POST localhost:8081/restro/updaterestro
- POST localhost:8081/restro/updaterestro
- POST localhost:8081/restro/updaterestro
- GET localhost:8081/restro/getrestro
- POST localhost:8081/restro/addrestro
- POST localhost:8081/restro/addrestro
- POST localhost:8081/appuser/login
- POST localhost:8081/appuser/login



GET localhost:8081/userpurchase/getrecord?userId=1

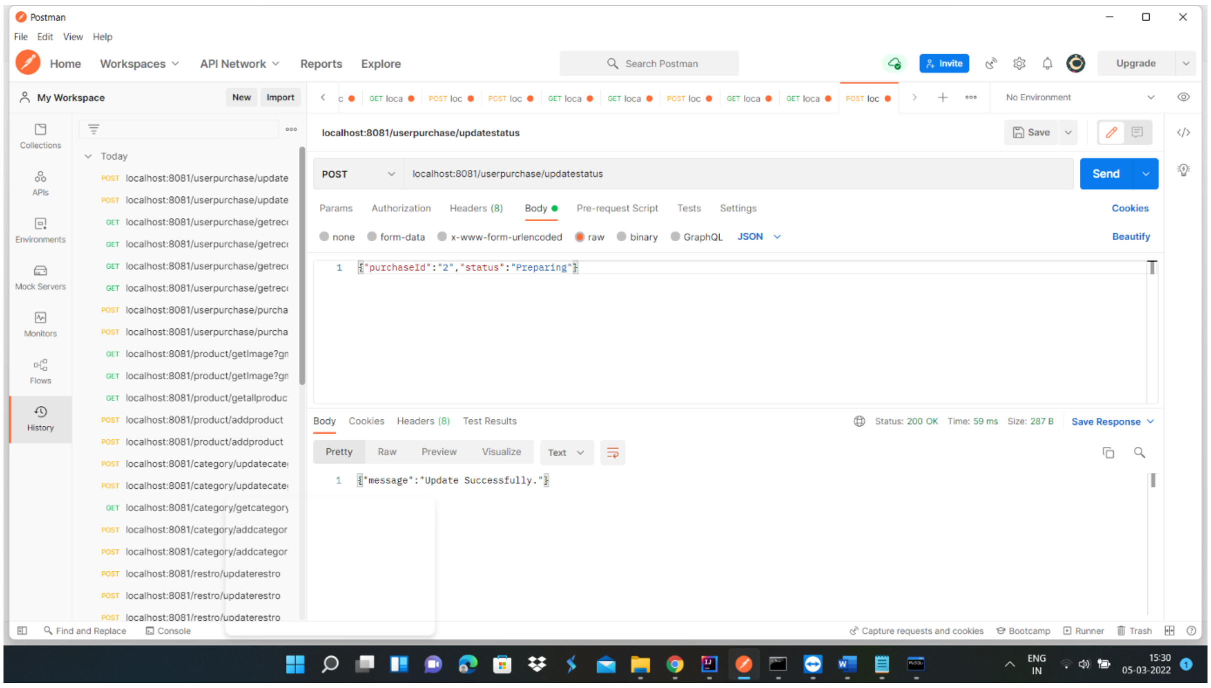
Query Params

KEY	VALUE	DESCRIPTION
userId	1	

Status: 200 OK Time: 66 ms Size: 594 B

```
    {
        "id": null,
        "rname": "Spy Restaurant",
        "contact": "1234567890",
        "address": "India",
        "gstno": null,
        "email": null,
        "role": null,
        "userId": null,
        "pdata": "{\"name\": \"abc\"}",
```

22



POST localhost:8081/userpurchase/updatestatus

Body (raw, JSON):

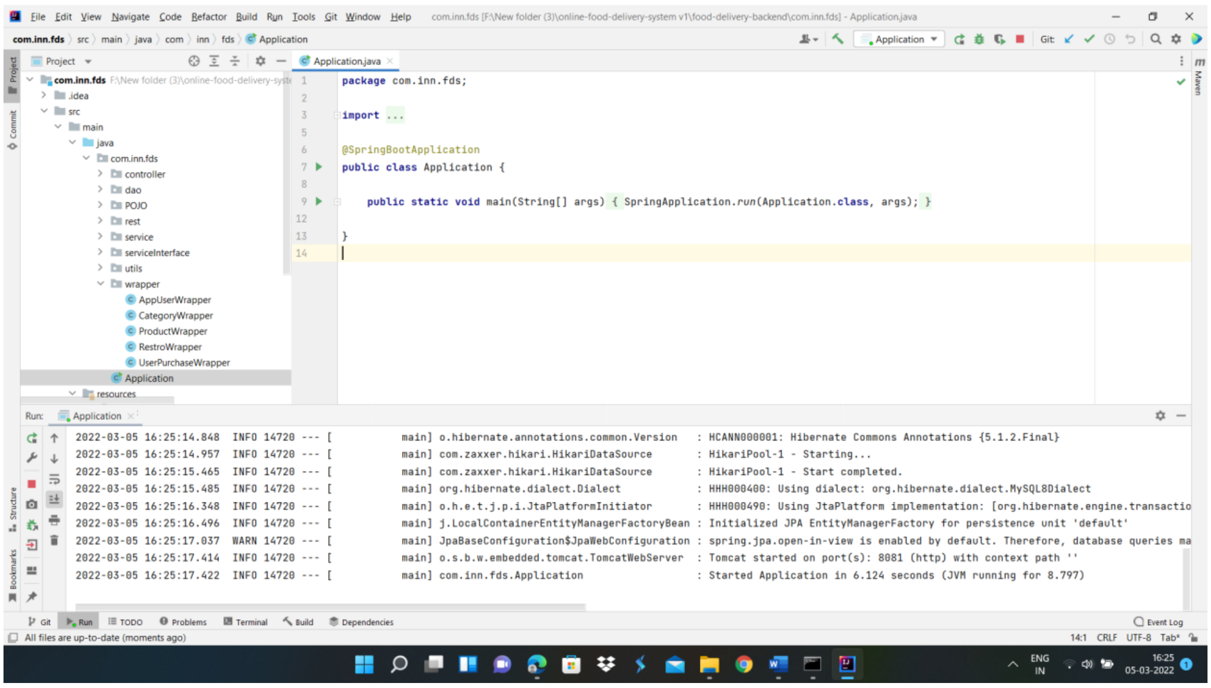
```
{"purchaseId":"2","status":"Preparing"}
```

Status: 200 OK Time: 59 ms Size: 287 B

```
{"message":"Update Successfully."}
```

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.



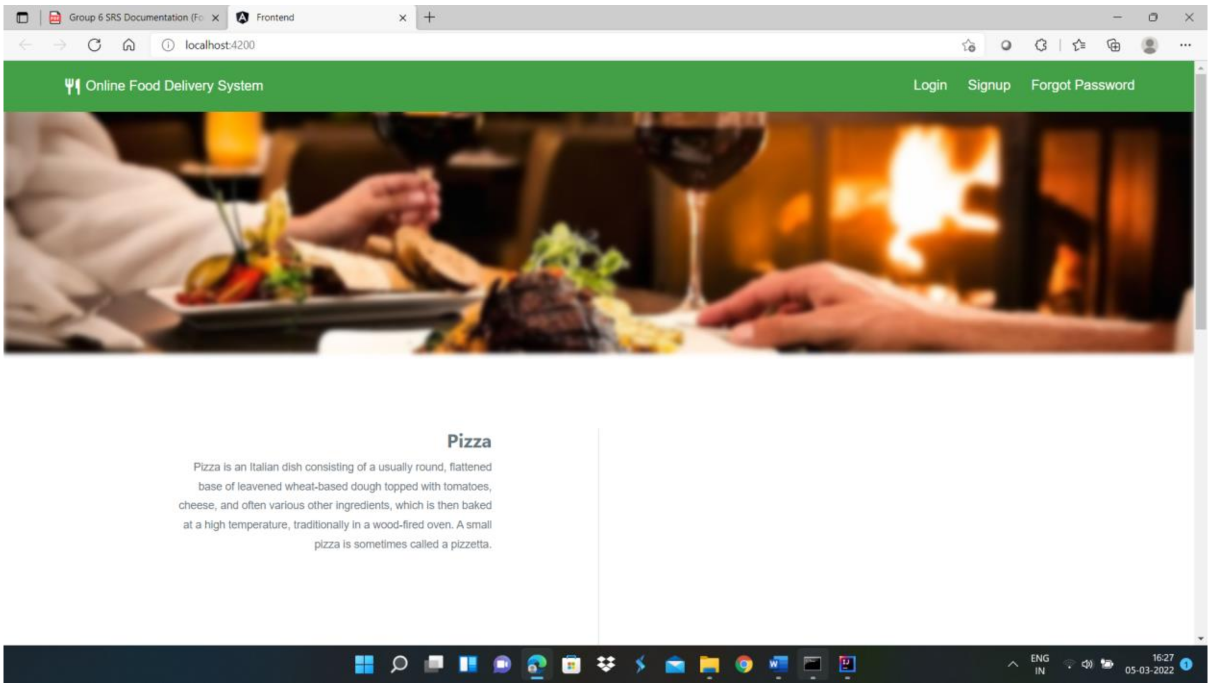
com.inn.fds [F:\New folder (3)\online-food-delivery-system v1\food-delivery-backend\com.inn.fds] - Application.java

```java
package com.inn.fds;

import ...

@SpringBootApplication
public class Application {

    public static void main(String[] args) { SpringApplication.run(Application.class, args); }

}
```

```
2022-03-05 16:25:14.848  INFO 14720 --- [           main] o.hibernate.annotations.common.Version   : HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
2022-03-05 16:25:14.957  INFO 14720 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2022-03-05 16:25:15.465  INFO 14720 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2022-03-05 16:25:15.485  INFO 14720 --- [           main] org.hibernate.dialect.Dialect            : HHH000400: Using dialect: org.hibernate.dialect.MySQL8Dialect
2022-03-05 16:25:16.348  INFO 14720 --- [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transactio
2022-03-05 16:25:16.496  INFO 14720 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2022-03-05 16:25:17.037  WARN 14720 --- [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries ma
2022-03-05 16:25:17.414  INFO 14720 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8081 (http) with context path ''
2022-03-05 16:25:17.422  INFO 14720 --- [           main] com.inn.fds.Application                  : Started Application in 6.124 seconds (JVM running for 8.797)
```

All files are up-to-date (moments ago)

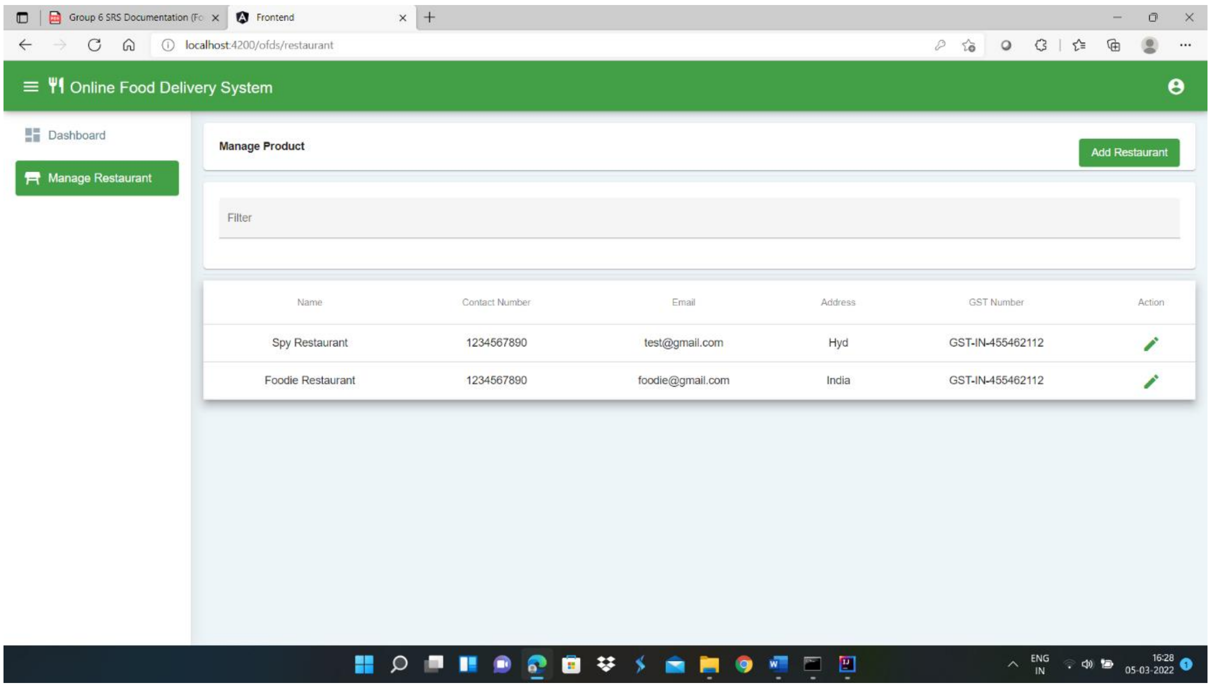
Home Page:



Pizza

Pizza is an Italian dish consisting of a usually round, flattened base of leavened wheat-based dough topped with tomatoes, cheese, and often various other ingredients, which is then baked at a high temperature, traditionally in a wood-fired oven. A small pizza is sometimes called a pizzetta.

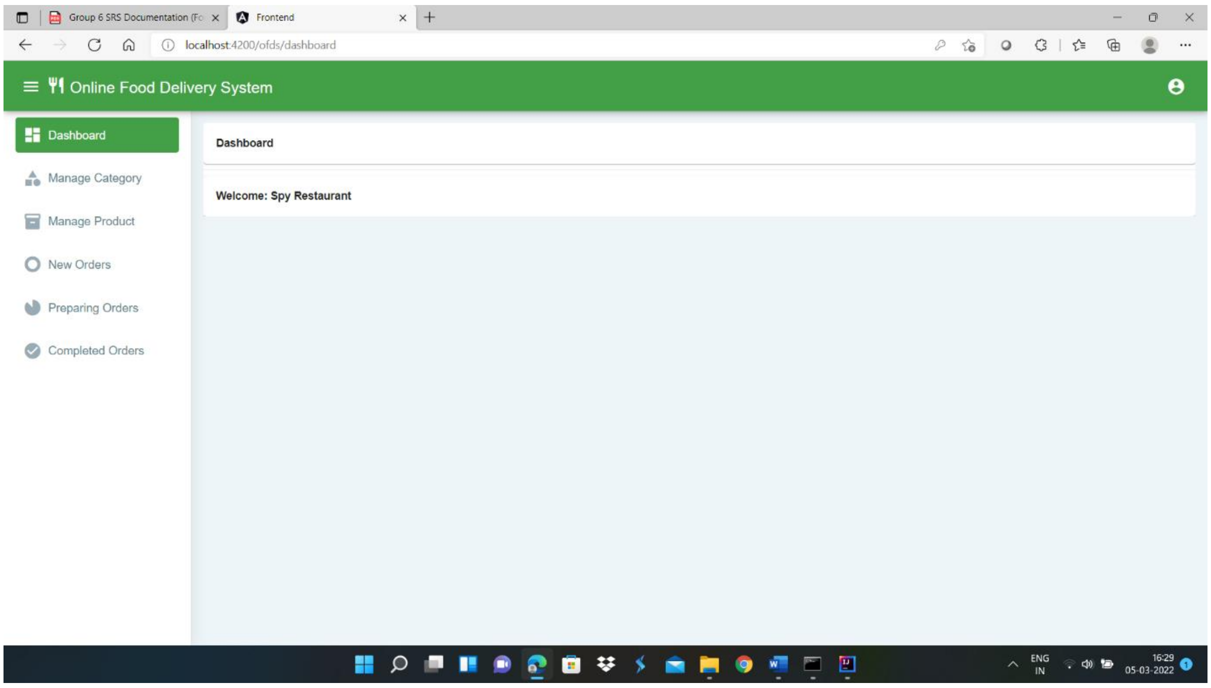
Admin Page:


Group 6 SRS Documentation (Fo
Frontend
localhost:4200/ofds/restaurant
Online Food Delivery System
Dashboard
Manage Restaurant
Add Restaurant
Filter

Manage Product

Name	Contact Number	Email	Address	GST Number	Action
Spy Restaurant	1234567890	test@gmail.com	Hyd	GST-IN-455462112	✎
Foodie Restaurant	1234567890	foodie@gmail.com	India	GST-IN-455462112	✎

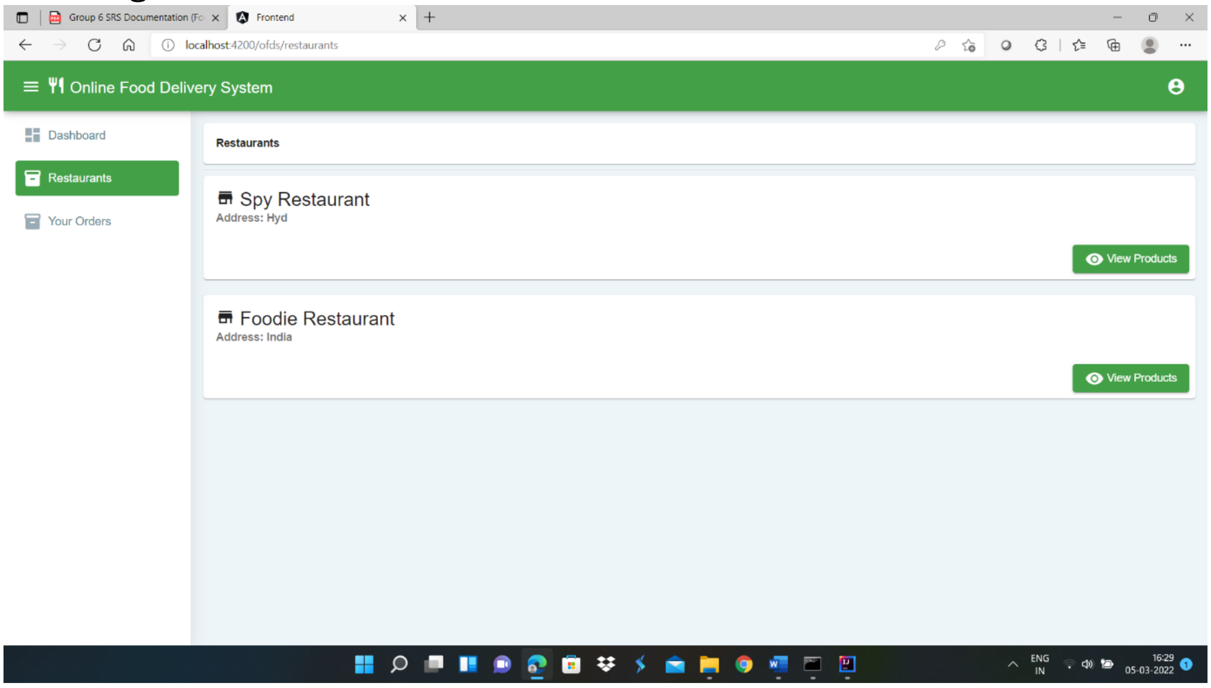
Restaurant Page:



Dashboard

Welcome: Spy Restaurant

User Page:


Group 6 SRS Documentation (Fo
Frontend
localhost:4200/ofds/restaurants
Online Food Delivery System
Dashboard
Restaurants
Your Orders

Restaurants

Spy Restaurant
Address: Hyd

View Products

Foodie Restaurant
Address: India

View Products

8.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’.