# ONLINE FOOD DELIVARY SYSTEM


**Team Leader:**

Pachala Deekshith

**Team Members:**

| | |
|---|---|
| Noora Sravani | 2576766 |
| P U V N S M Janakiram Sharma | 2577112 |
| Pachala Deekshith | 2576360 |
| Paramsetty Vineela Rathanajali | 2577156 |
| Peram Swapna | 2576321 |
| Pragati Pandey | 2576714 |
| Pramod X | 2577108 |
| Pramukha A U | 2577217 |


| | |
|---|---|
| **Group** | 5 |
| **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.

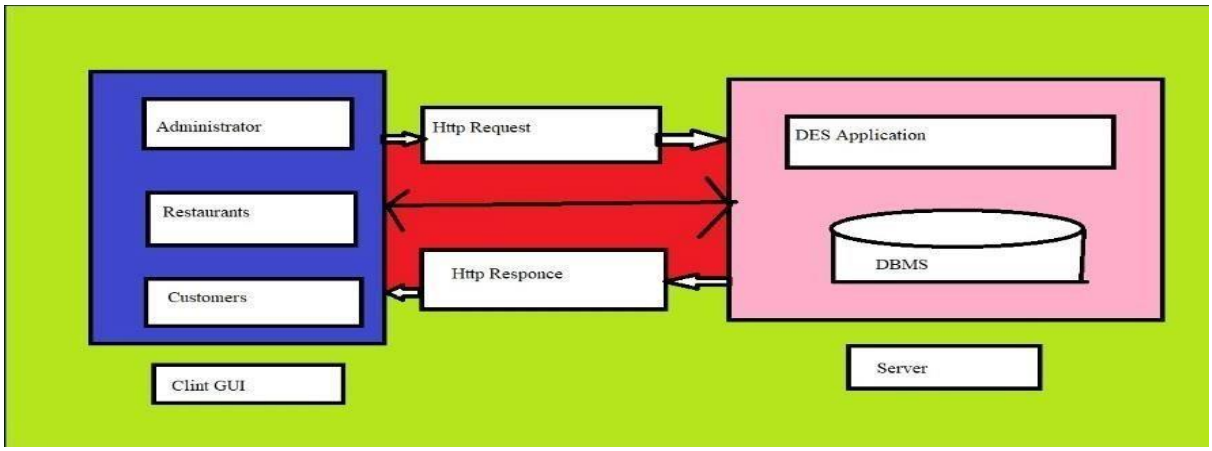
**Customers:**

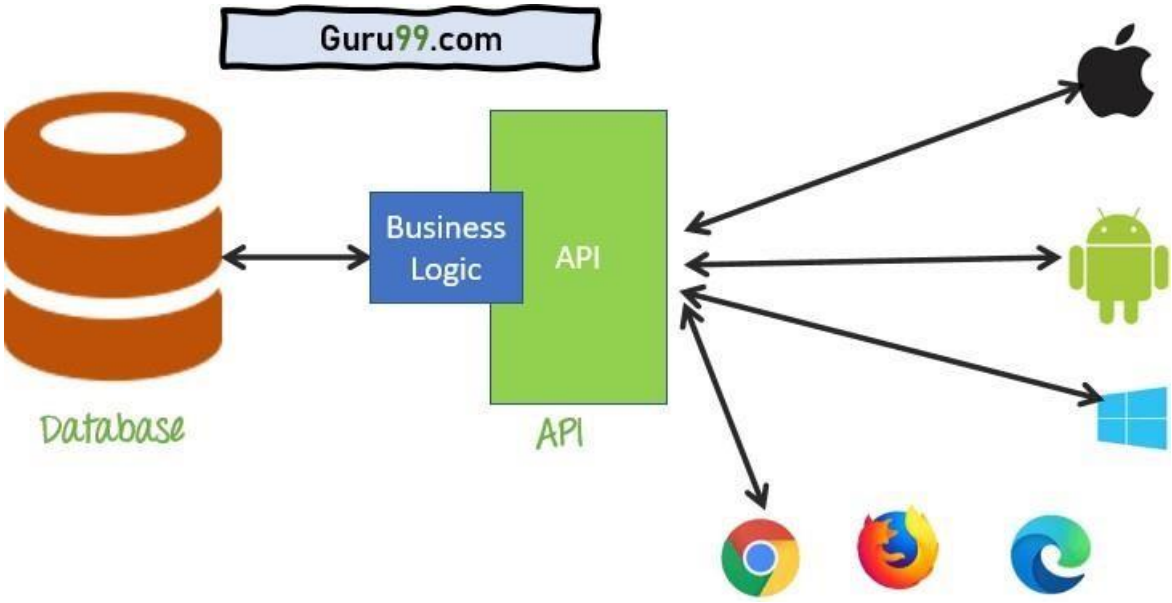
- Register their accounts in signup.
- After register successfully user can login.
- After login they can find numbers of Restaurants and select best one.
- Customers can see food items present in Restaurant.
- Add to cart or Order by proceeding amount.
- Customers can check their status once order a 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,<br>Contact Number,<br>Email | |
| 2 | User | Login | Email, Password | |
| 3 | Restaurants | Choose Restaurant | View Product | |
| 4 | Product | Add to cart,<br>Delete from cart,<br>Delete all the products from cart,<br>Procced to pay | Product id,<br>Product Name,<br>Price,<br>Quantity,<br>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: customer details ( database name: customer):

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 |
|---|---|
| Email(Primary Key) | Customer user name |
| Address | Customer address |
| Contact | Customer contact number |
| Name | Customer name |
| Password | Password given by user |

### 4.2 Table: Admin Details (Database name: admin):

In this table contains all restaurant details.

| Field Name | Description |
|---|---|

| | |
|---|---|
| Username(Primary Key) | Admin username |
| Password | Password for admin login |

### 4.3 Table: Product Details (Database name: product):

| Field Name | Description |
|---|---|
| Id(Primary Key) | Automatically generate by system |
| Actualprice | Enter price of product |
| avail | Say yes or no for availability of product |
| Category | Category of product adding |
| Des | Write description of product |
| Discount | Enter the discount amount |
| Imagepath | Enter the path of image adding |
| Name | Enter the name of the product |
| Price | Enter the price of product |

### 4.4 Table: Cart (database name: cart)

Here we can add food to cart

| Field Name | Description |
|---|---|
| Id(Primary key) | System automatically generate |
| Price | Item price |
| Quantity | Quantity of order |
| ProductID | ID of product |

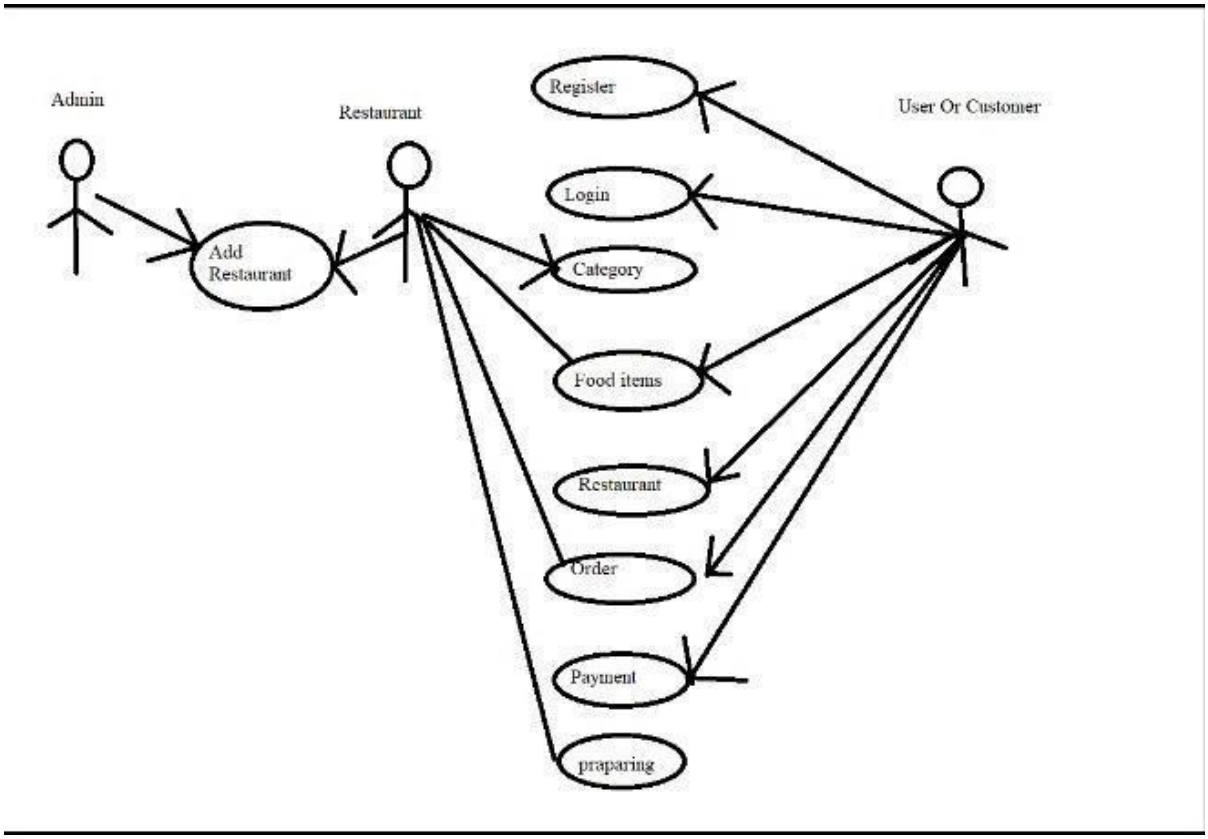
### 4.5 Table: User Purchase (Database name: purchase):

We can get in below table users transaction details.

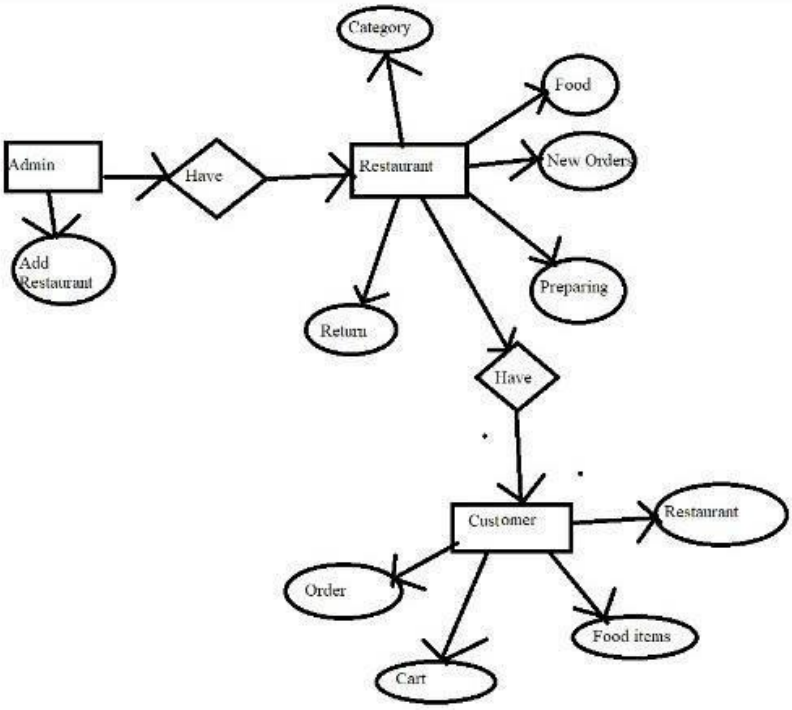
| Field Name | Description |
|---|---|
| Id(Primary Key) | System automatically genarate |
| Dop | Date of purchase |
| Productname | Name of the product |
| Quantity | Quantity of product |

| | |
|---|---|
| Totalcost | Total cost of order |
| Transactionid | Id generated for transaction |
| customeremail | Email id of customer placed an order |

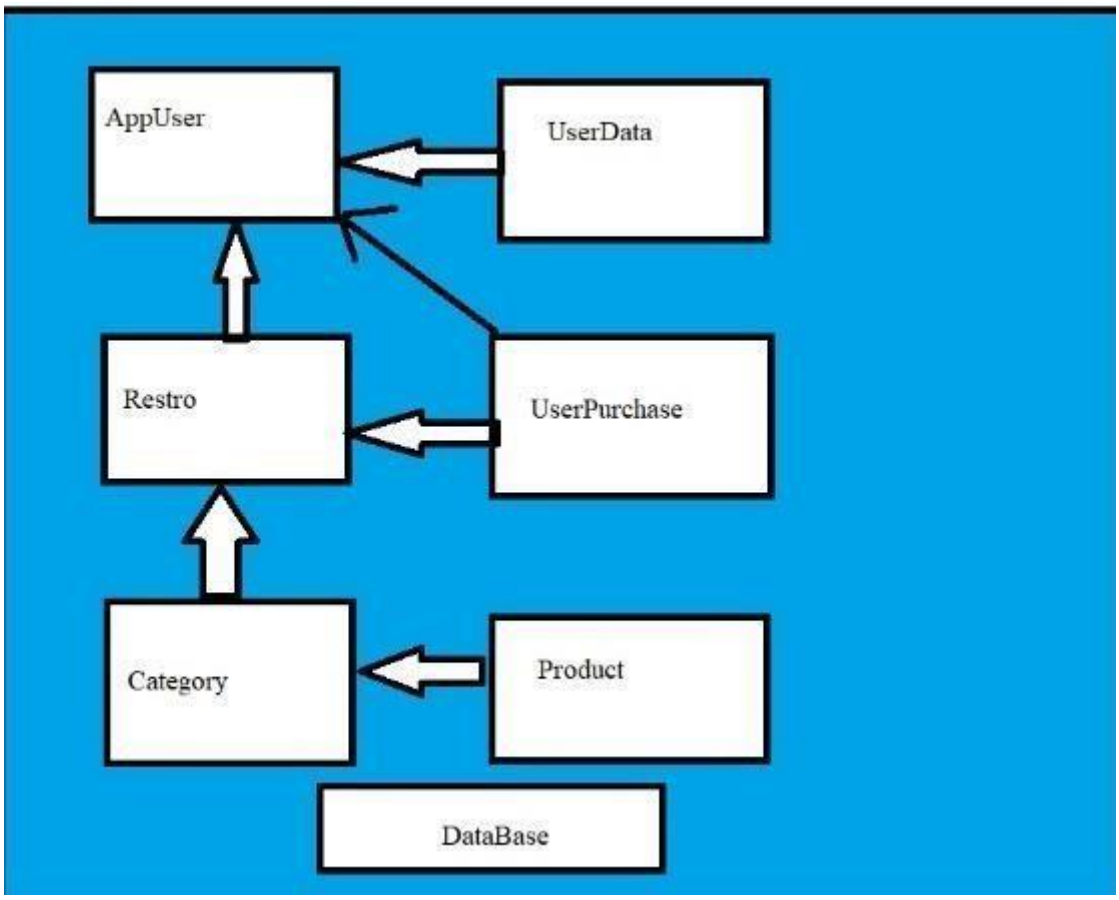
**4.6 UML Diagram:**




**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.

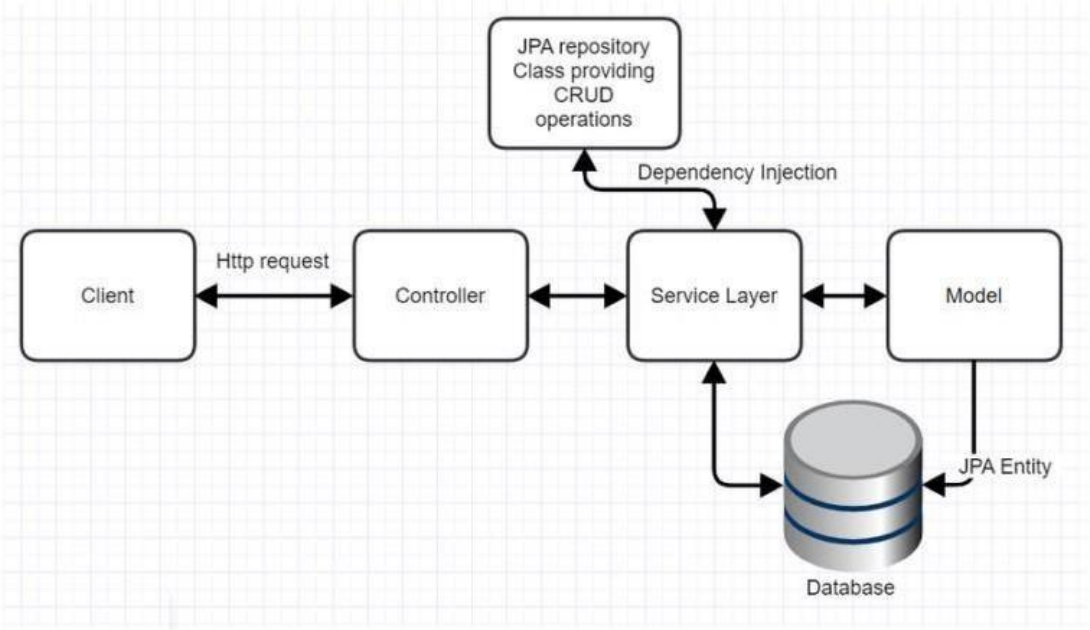



**Fig A : Representation of Spring MVC pattern**

## 5.2 Steps to create entities to perform Business logics:

### 1. create AppUser Entity:

- In IntelliJ IDEA We need to create an Entity:(customer).
- By Creating a customer 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.

- By Creating a UserServiceImpl class we can perform the required Business logics and exposes all Services.
- Finally, by creating a UserController class will handle all http requests and also will have following URL‟s

| URL | Methods | Description | Format |
|---|---|---|---|
| localhost:8084/customer | POST | Give a single user description searched based on username | JSON |
| localhost:8084/customers/ search/{keyword} | GET | Give a keyword | JSON |
| localhost:8084/customer/ {email} | DELETE | Give a email of customer | JSON |

2. **Create Cart Entity:**

Creating **Cart** Entity:

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

- Create an Entity: category
- Create a CategoryRepository interface and will make use of Spring

Data JPA

a. addcart method used to add food item type.
b. getCart method is used to get category types.
c. updateCart method used toupdate food item type.

3. Create a cartServiceImpl class and will expose all these services.
4. Finally, create a cartController will have the following Uri‟s:

| URL | Methods | Description | Format |
|---|---|---|---|
| localhost:8084/carts | GET | Getting the details of product in cart | JSON |
| localhost:8084/carts/add/{id} | PUT | Add product to cart by product id | JSON |
| localhost: 8084 /carts/minus/{id} | PUT | Remove product from cart | JSON |
| localhost: 8084 /carts/{id} | DELETE | | URL |

| | | | |
|---|---|---|---|
| | | Delete product present in cart by id | |

3. **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:8084/products/Admin | GET | Getting admin products | JSON |
| localhost:8084/products | POST | Product price | JSON |
| localhost:8084/products/{id} | DELETE | Delete product by id | URL |
| Localhost:8084/products/chinese | GET | Get chinese food | URL |

**5. Create purchase Entity:**

Creating **purchase** Entity:

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

i Create an Entity: Product

ii Create a purchaseRepository 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.

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

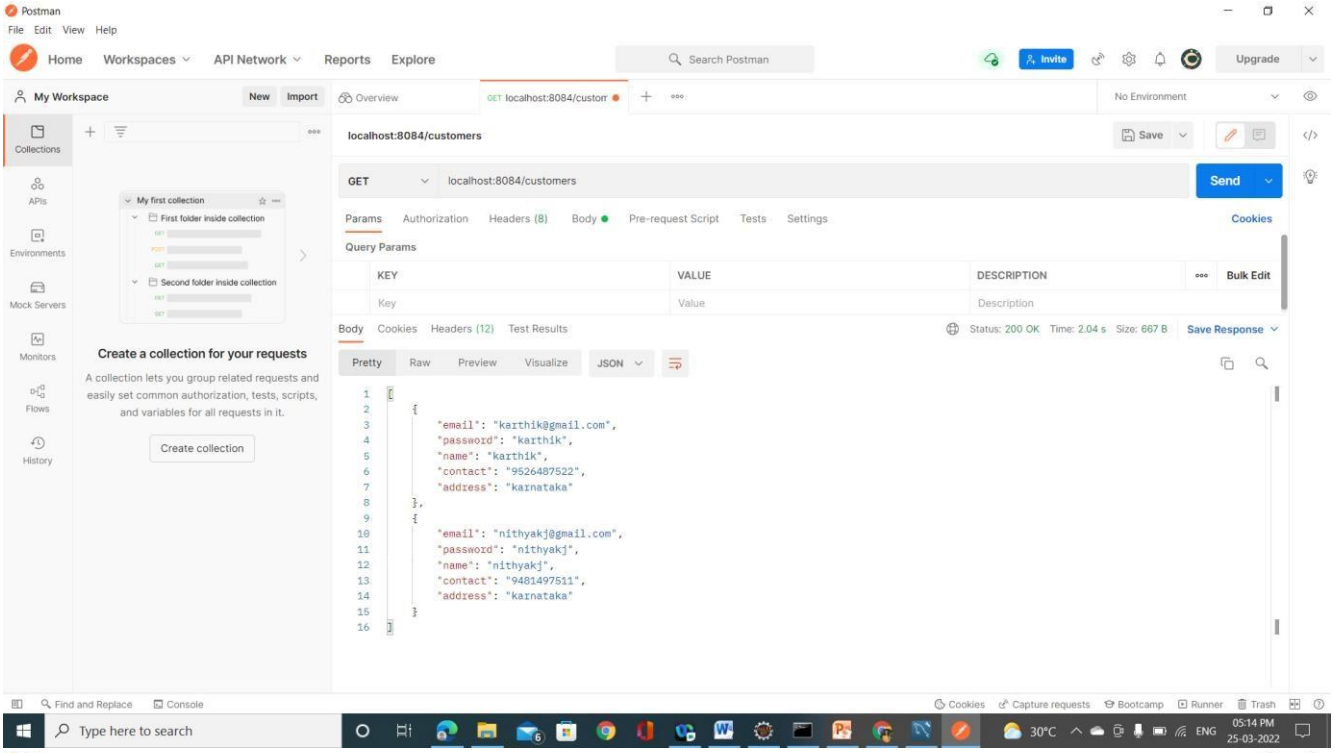
iv Finally, create a UserpurchaseController will have the following Uri‟s:

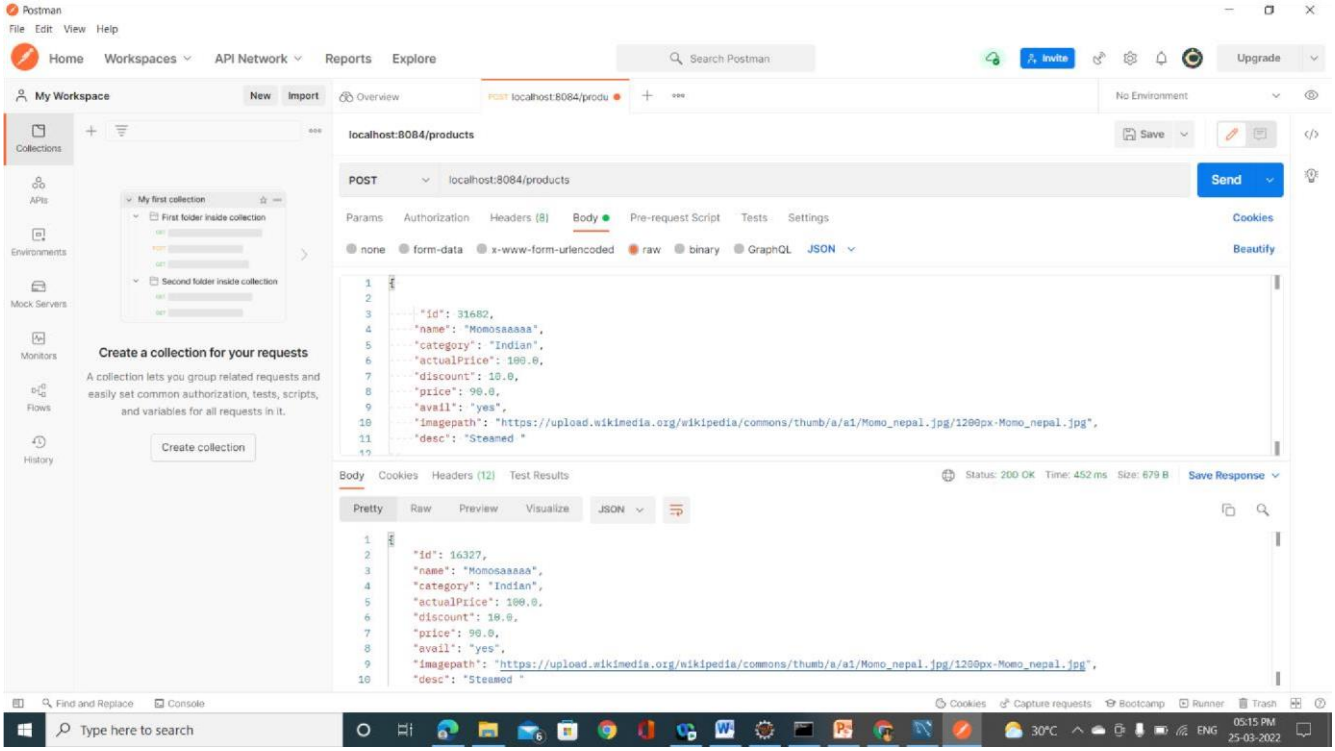
| **URL** | **Methods** | **Description** | **JSON** |
|---|---|---|---|
| localhost:8084/purchase/byEmail/{email} | GET | Get order purchase by email | URL |
| localhost:8084/ purchase | POST | Getting order placed details | URL |

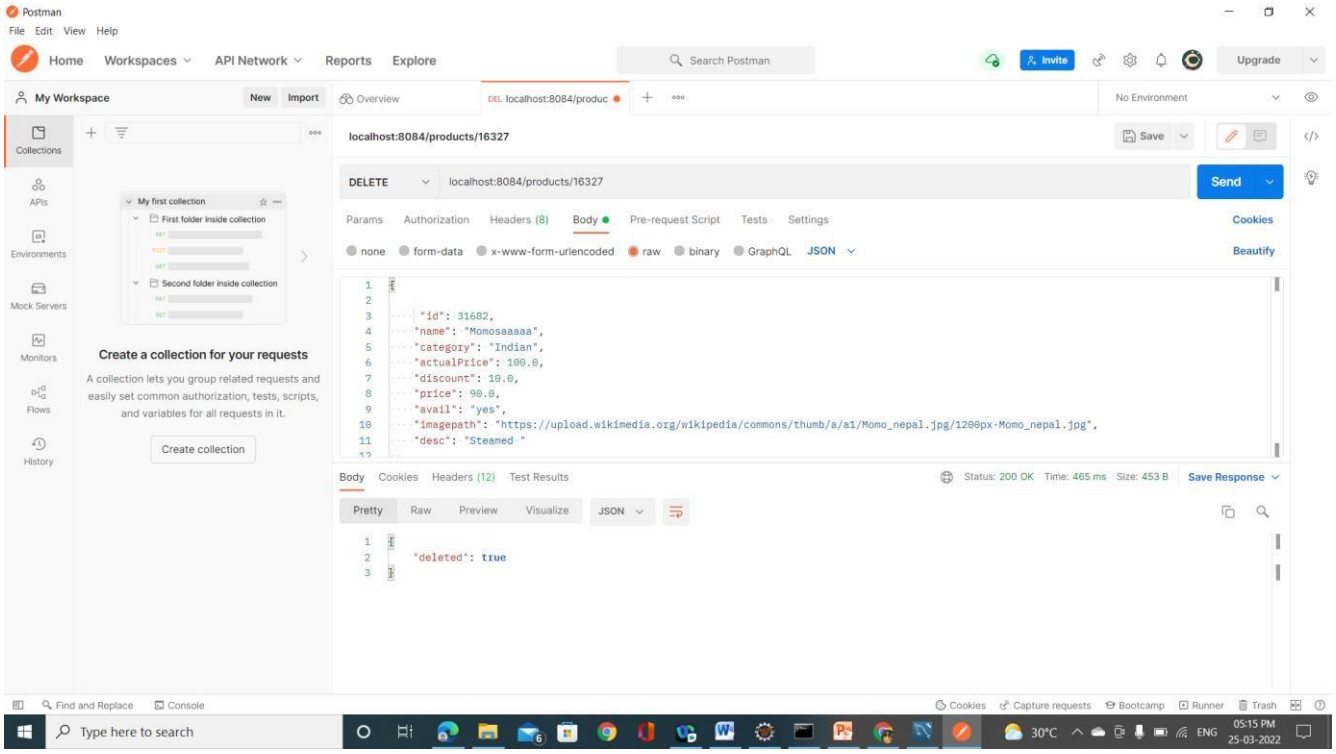
## 6. Implementation Details:

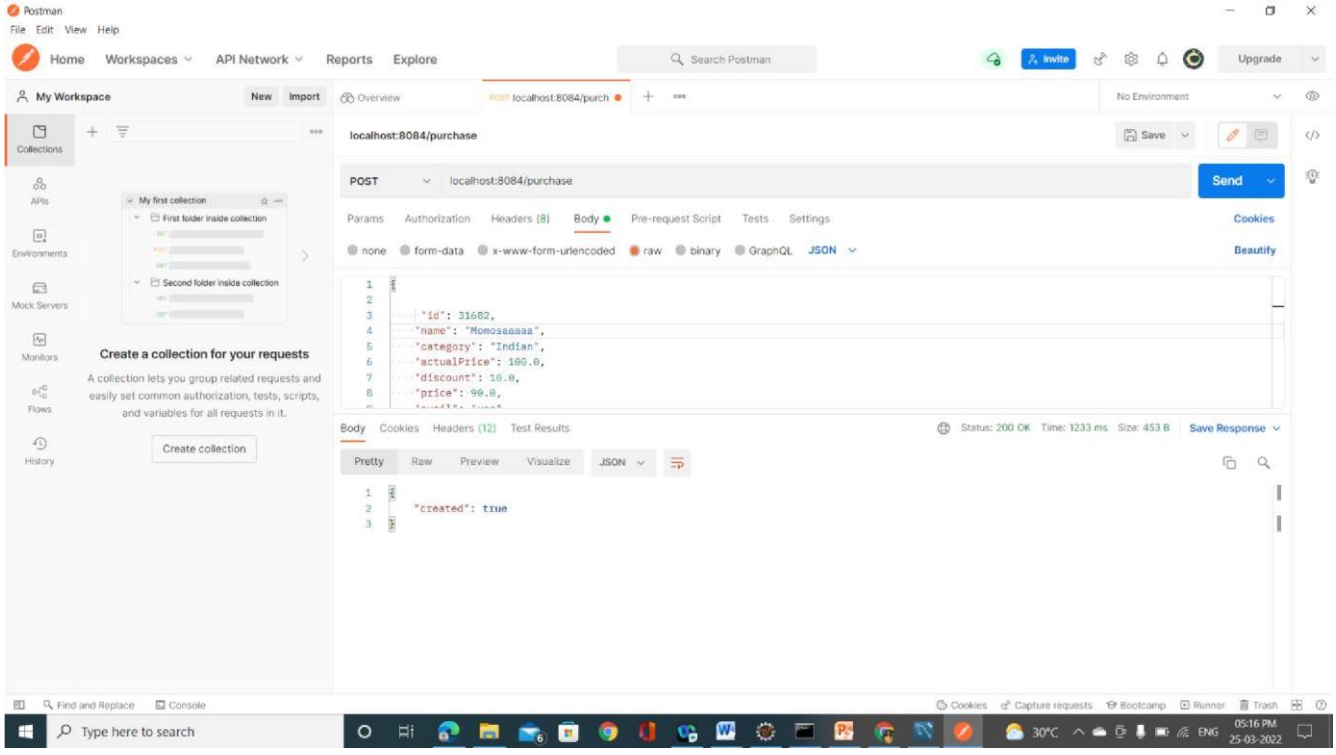
After deciding the technologies, We started to implement our system. First, we developed the backend for our system by creating a REST API for the server side application using java-based spring boot in IntelliJ IDEA. Then, we configured the backend API to connect to MySQL server @localhost:3306 to access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

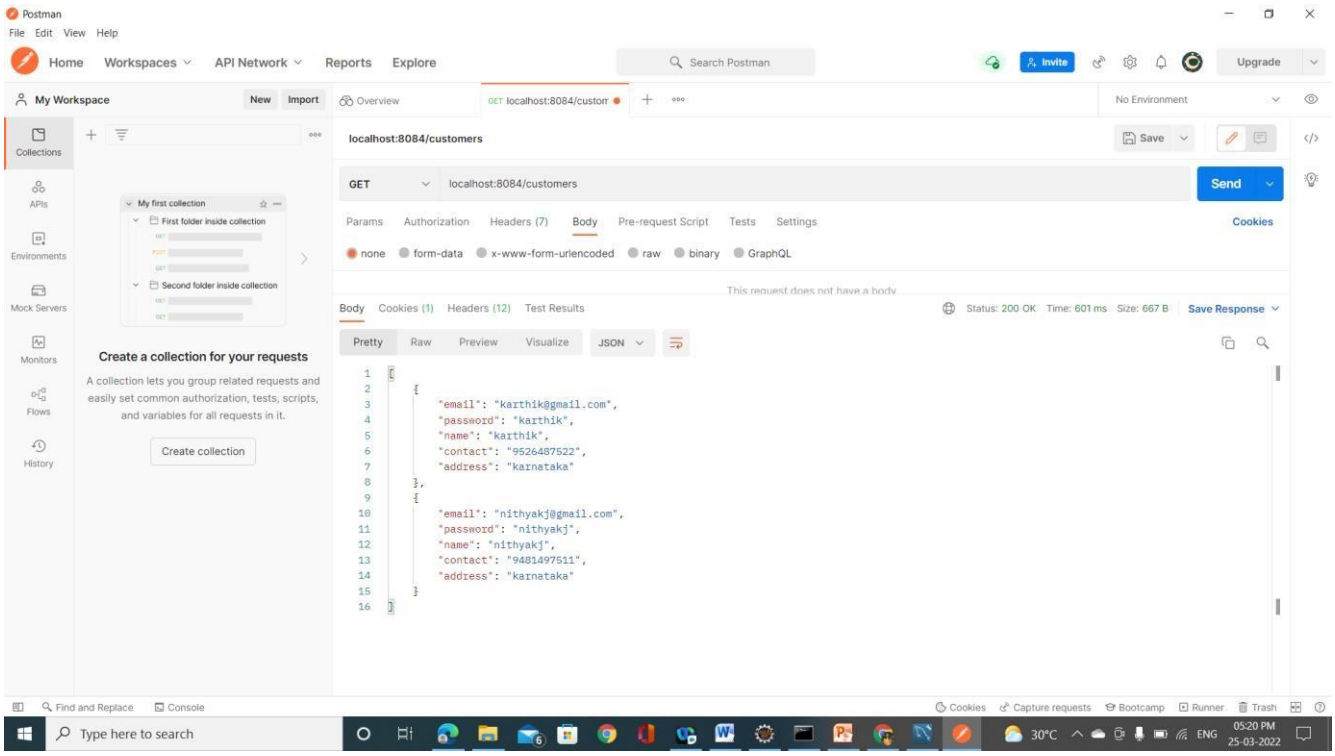
# 7. Functional Postman Testing:



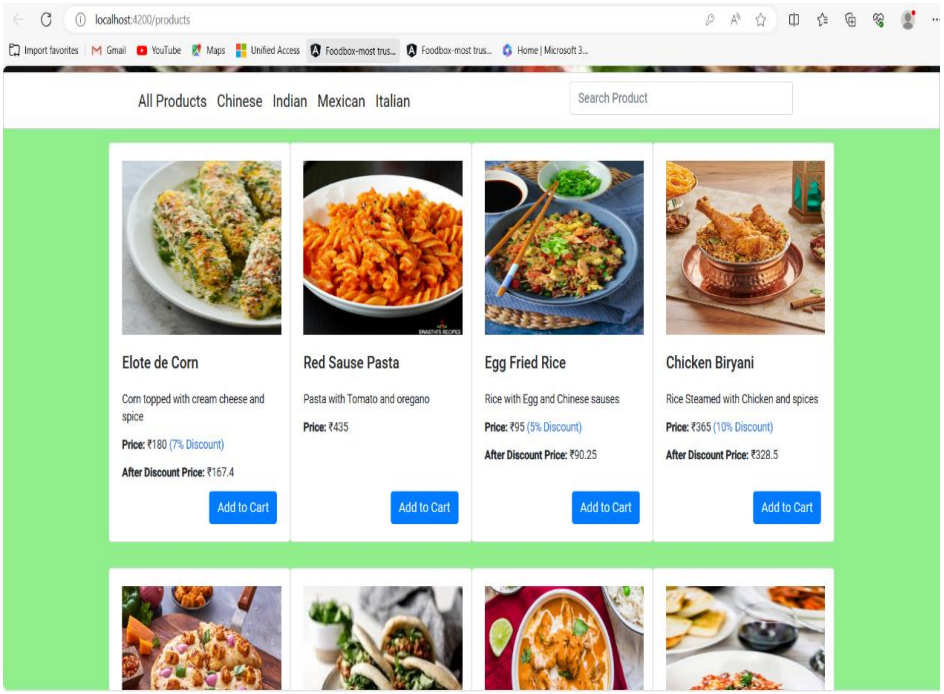

Postman
File Edit View Help
Home Workspaces API Network Reports Explore
Search Postman
Invite
Upgrade
My Workspace
New Import
Overview
DEL localhost:8084/produc
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
Flows
History
localhost:8084/products/16327
Save
DELETE
localhost:8084/products/16327
Send
Params Authorization Headers (8) Body Pre-request Script Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Beautify
{
"id": 31682,
"name": "Momosaaaaa",
"category": "Indian",
"actualPrice": 100.0,
"discount": 10.0,
"price": 90.0,
"avail": "yes",
"imagepath": "https://upload.wikimedia.org/wikipedia/commons/thumb/a/a1/Momo_nepal.jpg/1200px-Momo_nepal.jpg",
"desc": "Steamed "
Body Cookies Headers (12) Test Results
Status: 200 OK Time: 465 ms Size: 453 B
Save Response
Pretty Raw Preview Visualize JSON
{
"deleted": true
}
Create a collection for your requests
A collection lets you group related requests and easily set common authorization, tests, scripts, and variables for all requests in it.
Create collection
Find and Replace
Console
Cookies
Capture requests
Bootcamp
Runner
Trash
Type here to search
30°C
ENG
05:15 PM
25-03-2022


Postman
File Edit View Help
Home Workspaces API Network Reports Explore
Search Postman
Invite
Upgrade
My Workspace
New Import
Overview
POST localhost:8084/purch
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
Flows
History
localhost:8084/purchase
Save
POST
localhost:8084/purchase
Send
Params Authorization Headers (8) Body Pre-request Script Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Beautify
{
"id": 31682,
"name": "Momosaaaaa",
"category": "Indian",
"actualPrice": 100.0,
"discount": 10.0,
"price": 90.0,
Body Cookies Headers (12) Test Results
Status: 200 OK Time: 1233 ms Size: 453 B
Save Response
Pretty Raw Preview Visualize JSON
{
"created": true
}
Create a collection for your requests
A collection lets you group related requests and easily set common authorization, tests, scripts, and variables for all requests in it.
Create collection
Find and Replace
Console
Cookies
Capture requests
Bootcamp
Runner
Trash
Type here to search
30°C
ENG
05:16 PM
25-03-2022

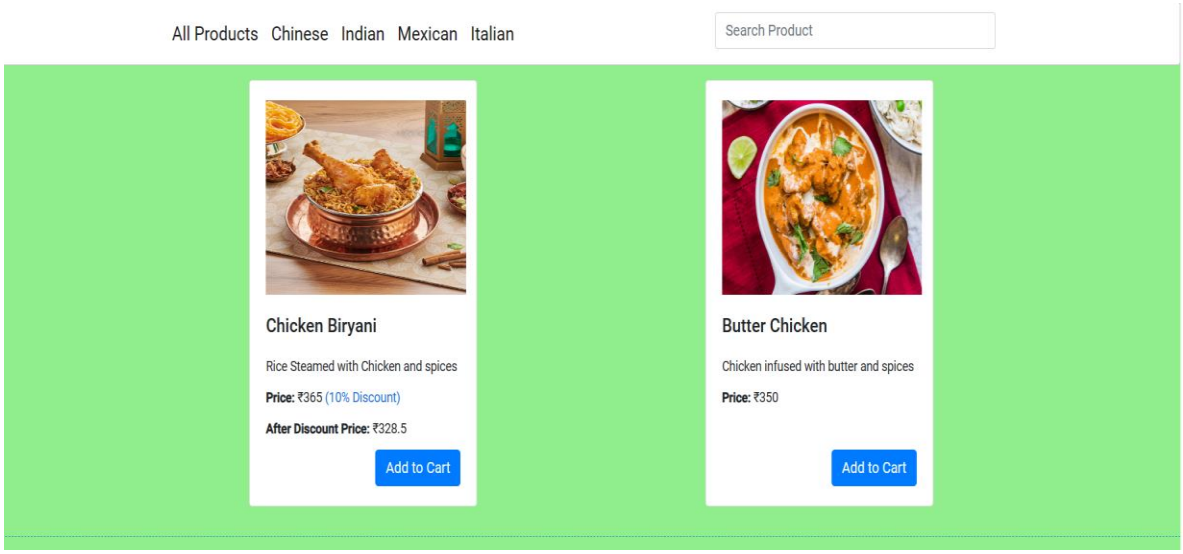



# HomePage:

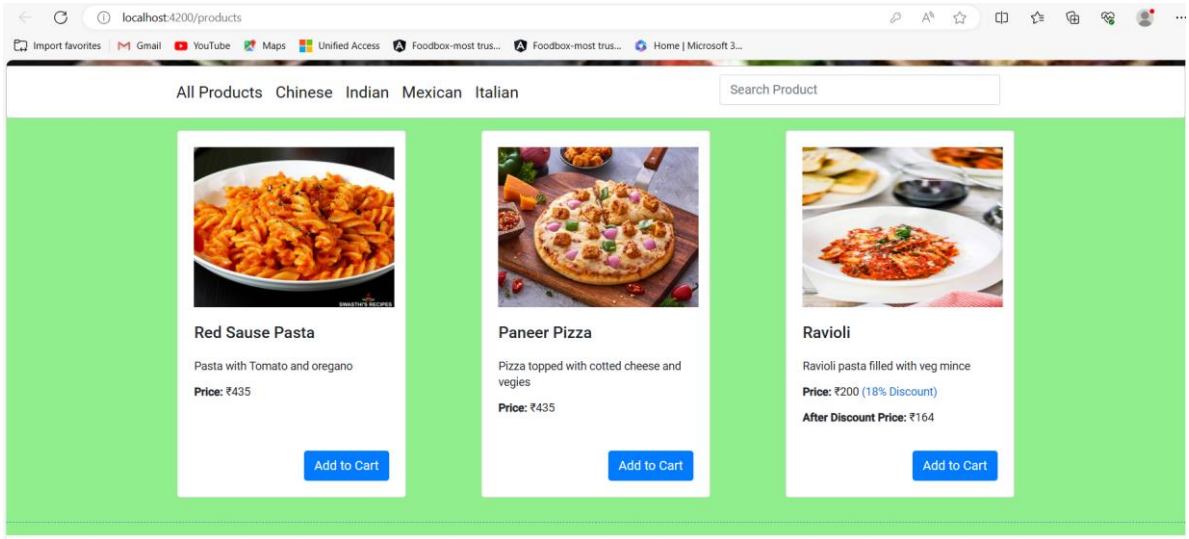
- ALLPRODUCT



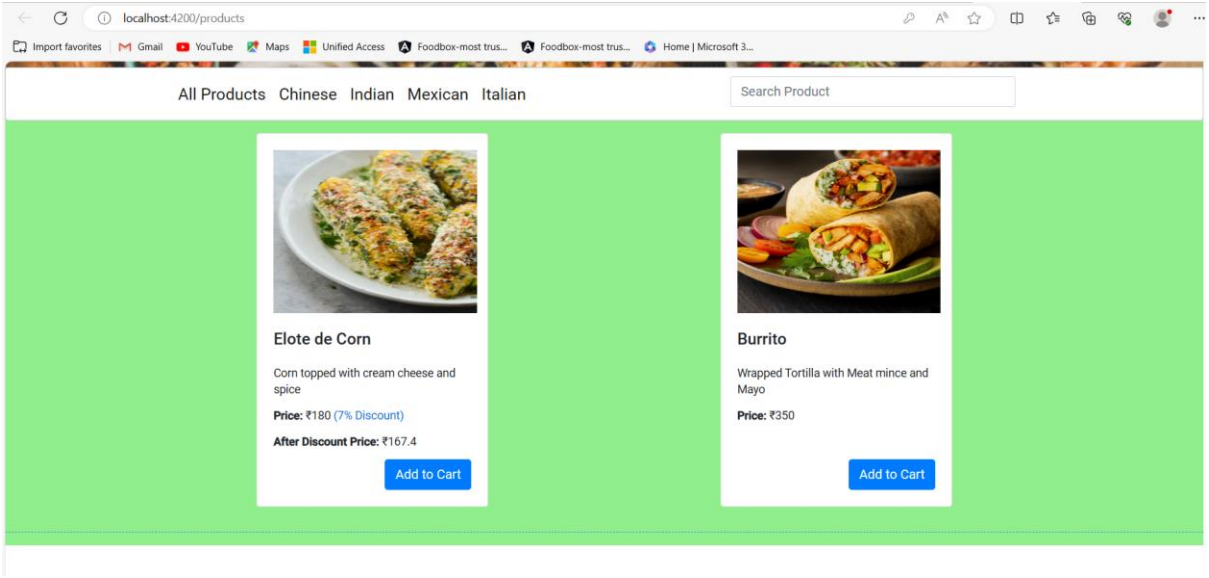
- Indian Product:



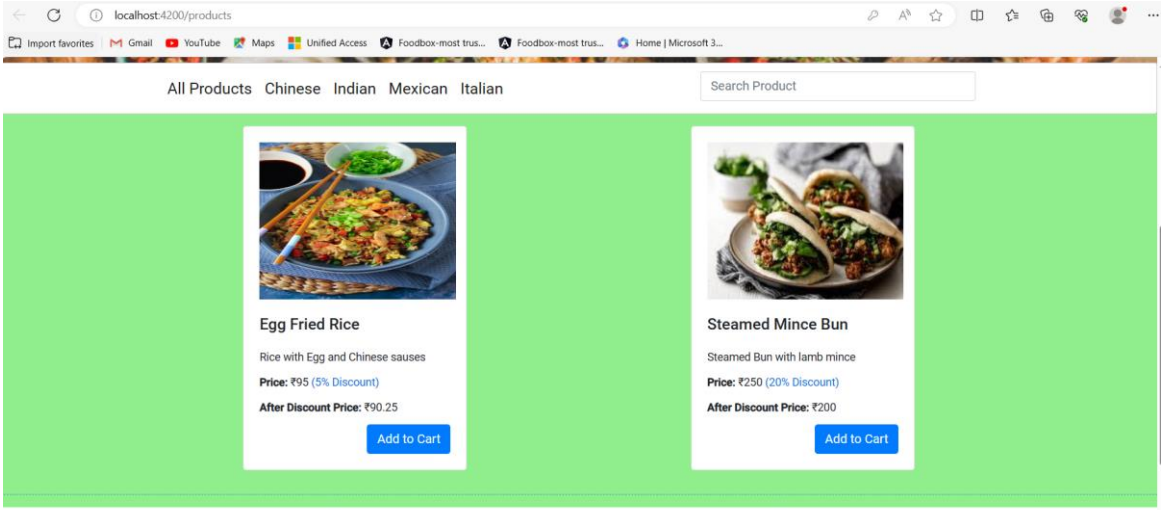

- Italian Product:



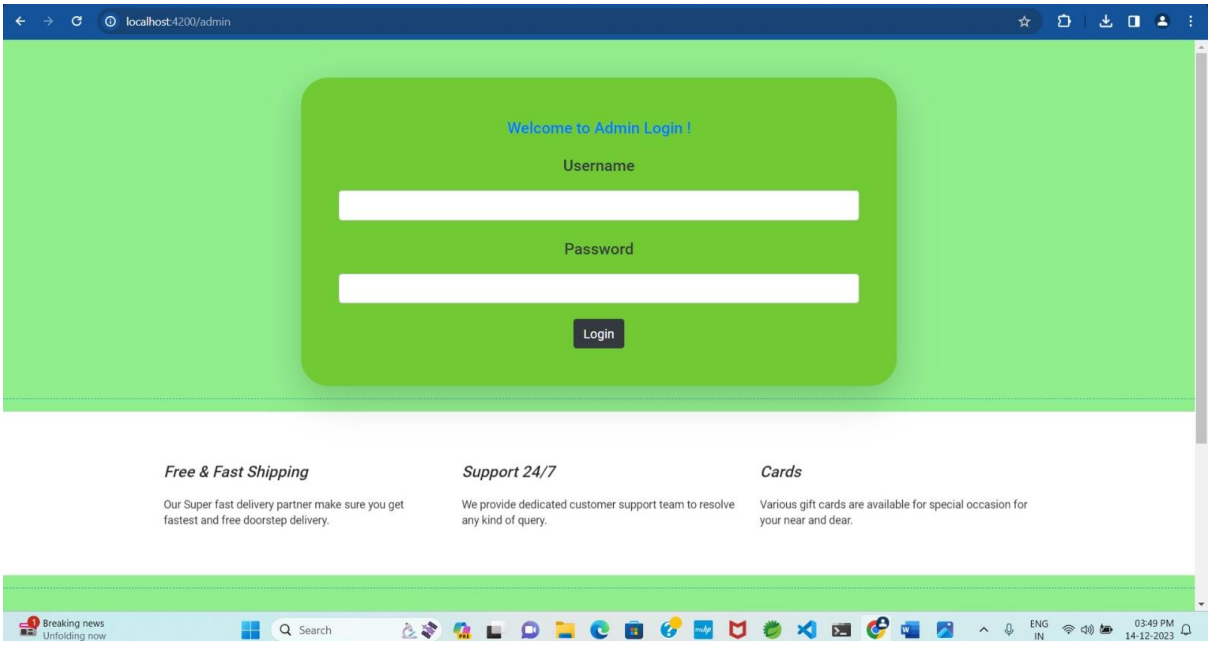
- Mexican Product:



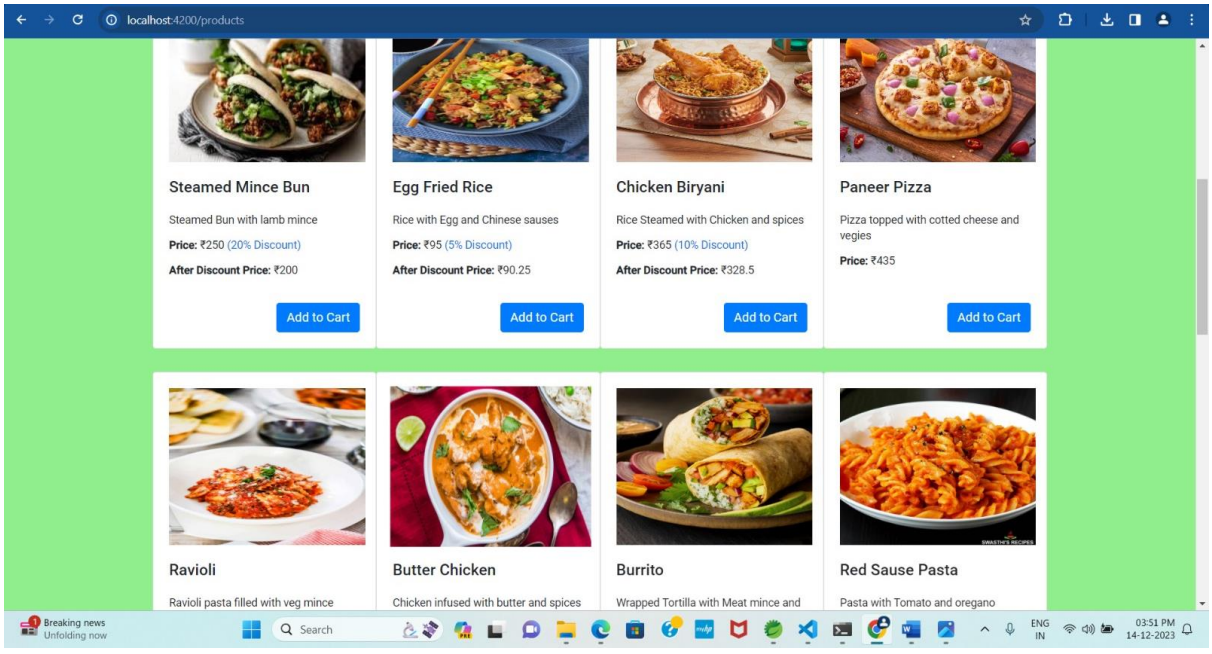

- Chinese Product



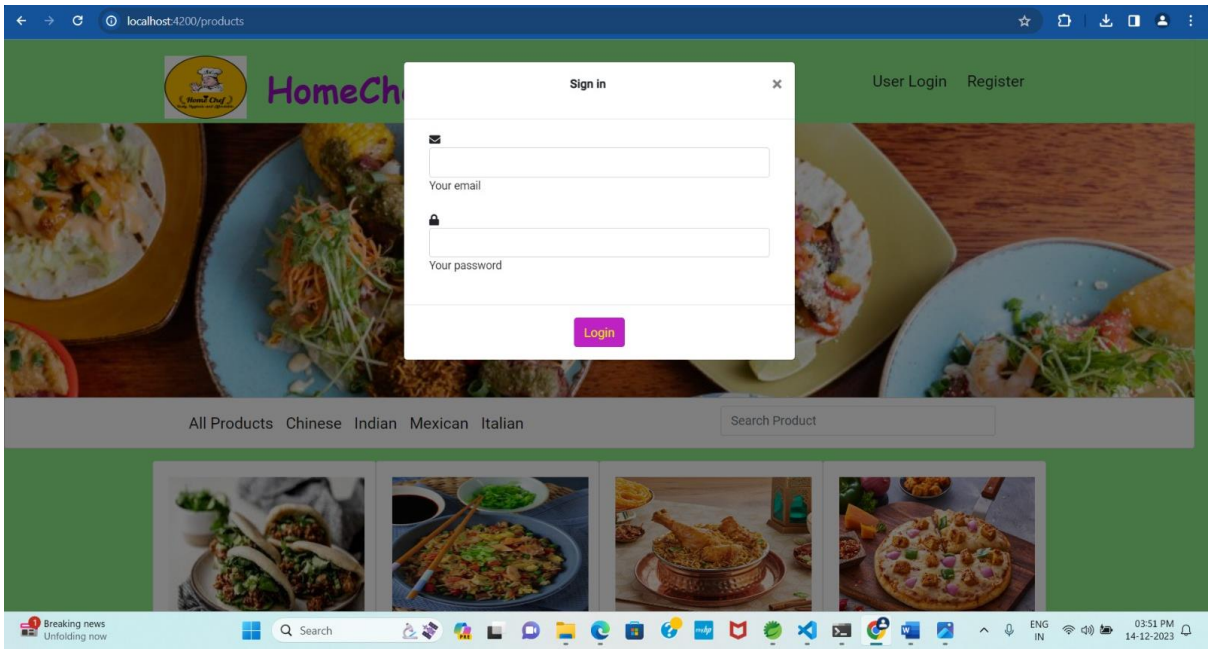
# Admin Page:



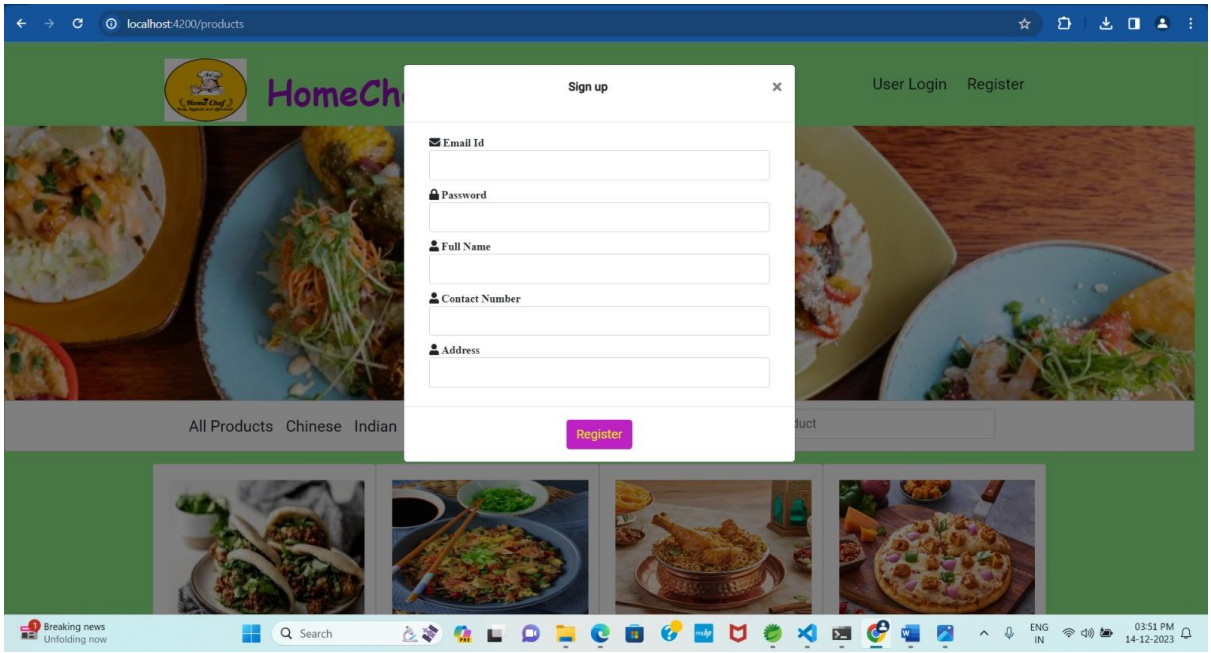

# Restaurant Page:



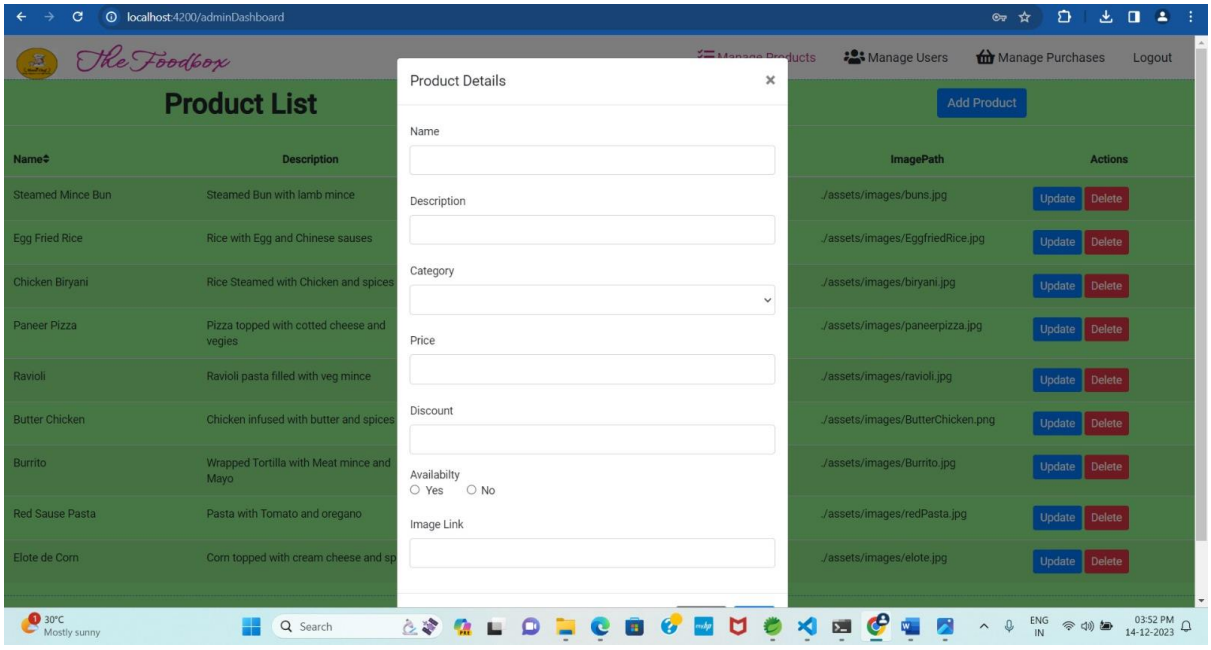

# User Page:



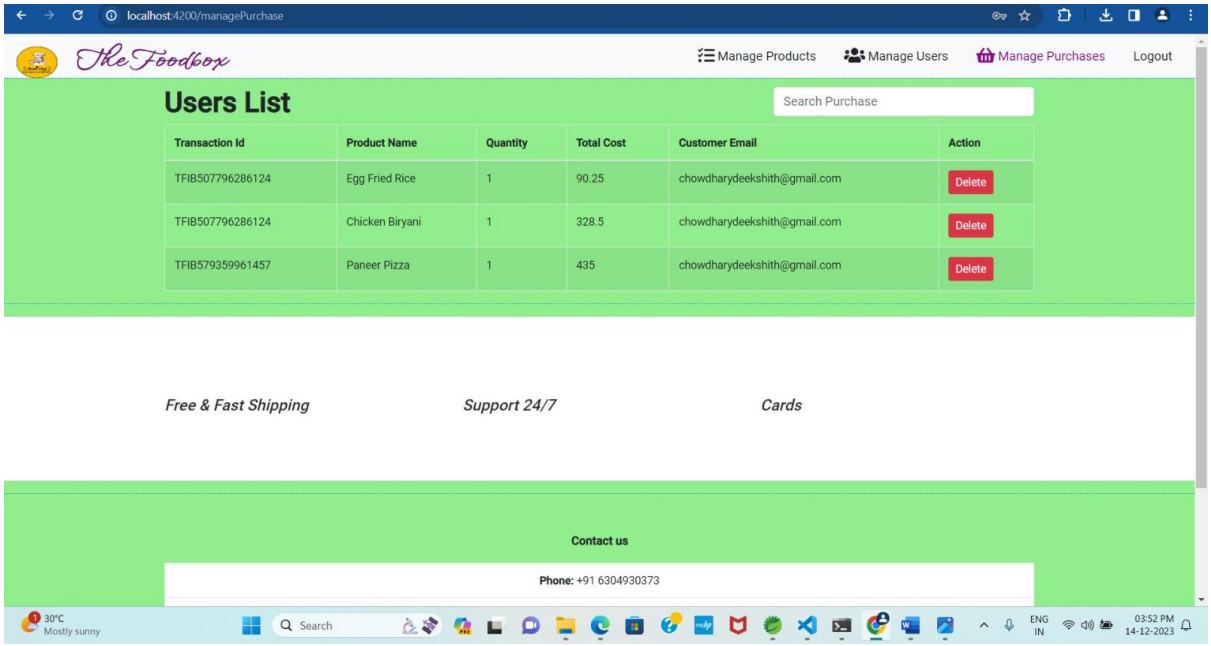

## Register Page:



- **Adding Product By admin Page:**




- **Admin Page after Login:**




## 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“.