RAMAIAH UNIVERSITY OF APPLIED SCIENCES

Faculty of Engineering & Technology			
Ramaiah University of Applied Sciences			
Department	Computer Science and Engineering	**Programme**	B. Tech. CSE
Semester/Batch	6th /2020		
Course Code	20CSC316A	**Course Title**	Introduction to Design Patterns
Course Leader(s)	Prabhakar A		

Assignment			
Register No.	20ETCS002062 20ETCS002117 20ETCS002151	Name of Student	KAKARLA SAI RAKESH SANIYA K R V.SRI VENKAT

Sections	**Marking Scheme**	**Max Marks**	**First Examiner Marks**	**Second Examiner Marks**
	Abstract statement	2		
	Detail Walkthrough	8		
	Part-A Max Marks	**10**		
	Draw the Class diagram.	3		
	Discuss the architectural style and design pattern used and justify.	10		
	Discuss salient quality parameters that is addressed. Justify your design.	2		
	Part-B2 Max Marks	**15**		
	Total Marks	**25**		

Course Marks Tabulation				
Component-1(B)Assignment	**First Examiner**	**Remarks**	**Second Examiner**	**Remarks**
A				
B				
Marks (out of 25)				

Signature of First Examiner Signature of Second Examiner

Please note:

1. Documental evidence for all the components/parts of the assessment such as the reports, photographs, laboratory exam / tool tests are required to be attached to the assignment report in a proper order.
2. The First Examiner is required to mark the comments in RED ink and the Second Examiner's comments should be in GREEN ink.

3. The marks for all the questions of the assignment have to be written only in the **Component – CET B: Assignment** table.
4. If the variation between the marks awarded by the first examiner and the second examiner lies within +/- 3 marks, then the marks allotted by the first examiner is considered to be final. If the variation is more than +/- 3 marks then both the examiners should resolve the issue in consultation with the Chairman BoE.

Assignment

Instructions to students:

1. The assignment consists of **2** questions: Part A –**1** Question, Part B- **1** Questions.
2. Maximum marks is **25**.
3. The assignment has to be neatly word processed as per the prescribed format.
4. **Submission Date:**
5. **Submission after the due date is not permitted.**
6. **IMPORTANT**: It is essential that all the sources used in preparation of the assignment must be suitably referenced in the text.
7. Marks will be awarded only to the sections and subsections clearly indicated as per the problem statement/exercise/question

RAMAIAH UNIVERSITY OF APPLIED SCIENCES

Scenario

Discuss with the Course leader and identify the project scenario. In this regard students are required to develop DIK model.

PART A **10 Marks**

A.1 Write an abstract statement.

A.2 Do a detailed walk through identifying all the transactions and process that happen in each of the transactions?

PART B **15 Marks**

B1.1 Draw a Class Diagram depicting the system.

B1.2 Discuss the architectural style and design pattern used to implement the design. Justify

B1.3 Discuss salient quality parameters that is addressed. Justify your design.

Note: Make appropriate assumptions to make the specification complete.

****

RAMAIAH UNIVERSITY OF APPLIED SCIENCES

PART-A:

Food delivery application (DIKW – DATA INFORMATION KNOWLEDGE WISDOM MODEL)

Data:

Food Delivery application collects the data of the customer orders, including the date and time of the day, location, and food preferences, favourite food of customers etc.,

Information:

The application will analyze the order data patterns and designs. The company finds that customers in certain areas prefer specific types of cuisine, and the orders peak during certain times of the day. And also finds the food which is rated by many customers from that particular restaurant.

Knowledge:

Based on the information and analysis, the company understand the trends of the generation of the food and the expansion in restaurant partnerships in identified areas and ensuring timely deliveries during peak hours can improve customer satisfaction and increase revenue. The company decides to invest in recruiting new restaurant partners in the identified areas and to optimize its delivery routes during peak hours.

Wisdom:

As a result of the changes, the company sees an increase in customer satisfaction, as well as an improvement in revenue and market share. The company realizes that by using data to inform decision-making, they can better understand their customers and improve their business outcomes. The decision to invest in new restaurant partnerships and optimized delivery routes is informed by a combination of data, information, and knowledge, and is based on a deeper understanding of the company's customers and business goals.

A1. Abstract statement:

According to the scenario selected by us (food delivery system), which is an application/system used for food delivery from restaurants to the customers. By this Application/system any user(customer) can order food from the restaurants which are already a part of the application/system. Here mainly the system depends on the rating of the customers.

[CUSTOMER SHARES LOCATION → CUSTOMER SELECTS RESTAURANT → CUSTOMER ORDERS FOOD →CUSTOMER PAYS THE BILL→ RESTAURANT ACCEPTS THE ORDER→DELIVERY GUY DELIVERS FOOD TO CUSTOMER →CUSTOMERS GIVE THE FEEDBACK]

A2. Walk-through:

1. User sign-up for the food delivery application by providing his details (e-mail id/phone number etc.,) and by selecting the agent and customer. The Application stores the details of the user.

   [process: here the details for sign up will be stored in an restricted database table, the details for the type of the person is also stored here.]

2. user logins to the application by using their credentials. The application verifies the credentials provided by the customer while logging-in.

   [for login the credentials are taken from the database and then verifies for login else gives invalid credentials. After verification the session of that user/delivery agent will be started.]

3. Customer shares his location, admin validates the location and displays the restaurants based on it, and also it displays all the categories and offers on restaurant in section-wise.

[by the location attribute the restaurants are displayed, based on the already existed restaurant data in the database.]

4. Customer selects the restaurant according to his/her needs from which he/she wants to order the food. Customer can also select the option for the veg/non-veg food.

   [By the selection of the restaurant the food will be displayed, the food will be displayed to the customer on the basis of the data of the food and also the attributes for present in the database.]

5. Application gives a option for filtering of the type of food such as specials of the day, most-rated, deal of the day etc.., based on the season and day. Then, application displays the available food based on their selection.

   [By using the filtering option that is used in the back-end by comparing the attribute of the data given in the filter and sorting of the data based on it.]

6. Customer selects the food and chooses the quantity of the food he needs and add them to the cart. Once all the items are added to cart, application asks for the details of the address of the customer. customer provides the details of the order.

   [from here the billing will be done based on the price attribute of the food data.]

7. The application displays the total amount of the order and details about the order, section for applying any coupon code/gift card. Then the customer can apply the coupons during checkout, and the application then displays the amount that needed to be paid after applying coupon.

   [The billing calculated will based on the previous step and then displayed and also each food price and quantity is displayed by using the data stored from previous steps.]

8. Customer proceeds for the checkout, application displays all the mode of payments (credit/debit cards, UPI, wallets, pay later, cash on delivery).

   [Here the data of the mode of payments data from the database is displayed and details will be redirected to the the bank server]

9. Customer selects the mode of payment and proceeds to pay. Bank verifies and validates the transaction if the payment is done by the credit, debit card. The application of pay later will verified the transaction if the payment is done by the pay later option. If the payment is successful, the application generates the message of order placement and also send order details to customer's email/phone number. Else, the process will be repeated from 7^{th} step.

   [The data received from the previous step will be verified by the bank and transaction will be validated and if at all there is an issue in validating by the bank using that data. The payment will redirect to the user as failed payment.] and [The data of the placement of the order will be saved in the database of the orders.]

10. If the Restaurant rejects the order the amount of the order will be refunded to the customer's bank in the 7 working days provided by the details by the customer. Else the application assigns that order to delivery agent. Once the delivery agent is assigned by the application, information of the delivery agent such as his name, his live location, estimated time for delivery is shown to the customer by the application. Application also gives an option for calling to delivery agent. Application also displays the order details and the address details of the customer to the delivery agent.

    [The transaction will be handled by the bank which receives the payment validation request if the transaction fails in an order and the money is debited. Then the bank may take upto 7 days of time for verification and validation of the refund]

11. Delivery agent visits the restaurant and picks up the order. the delivery agent call to the customer for verifying address of customer and goes to location. Application notifies the customer once the delivery guy comes to the location in the estimated time provided by the application. delivery

12. agent delivers the order to the customer.

    [The information delivery agent and restaurant data will be already present in the database and that will be updating in the application and in the database, This manages the user display for the delivery guy and restaurant details. The delivery guy will also get the location data from the database from where the user given has been stored before.]

13. Customer takes the order and pays if he has selects cash on delivery else just takes order. application provides an option for rating and reviewing the restaurant and order. Customer can rate about the order and restaurant after having the food.

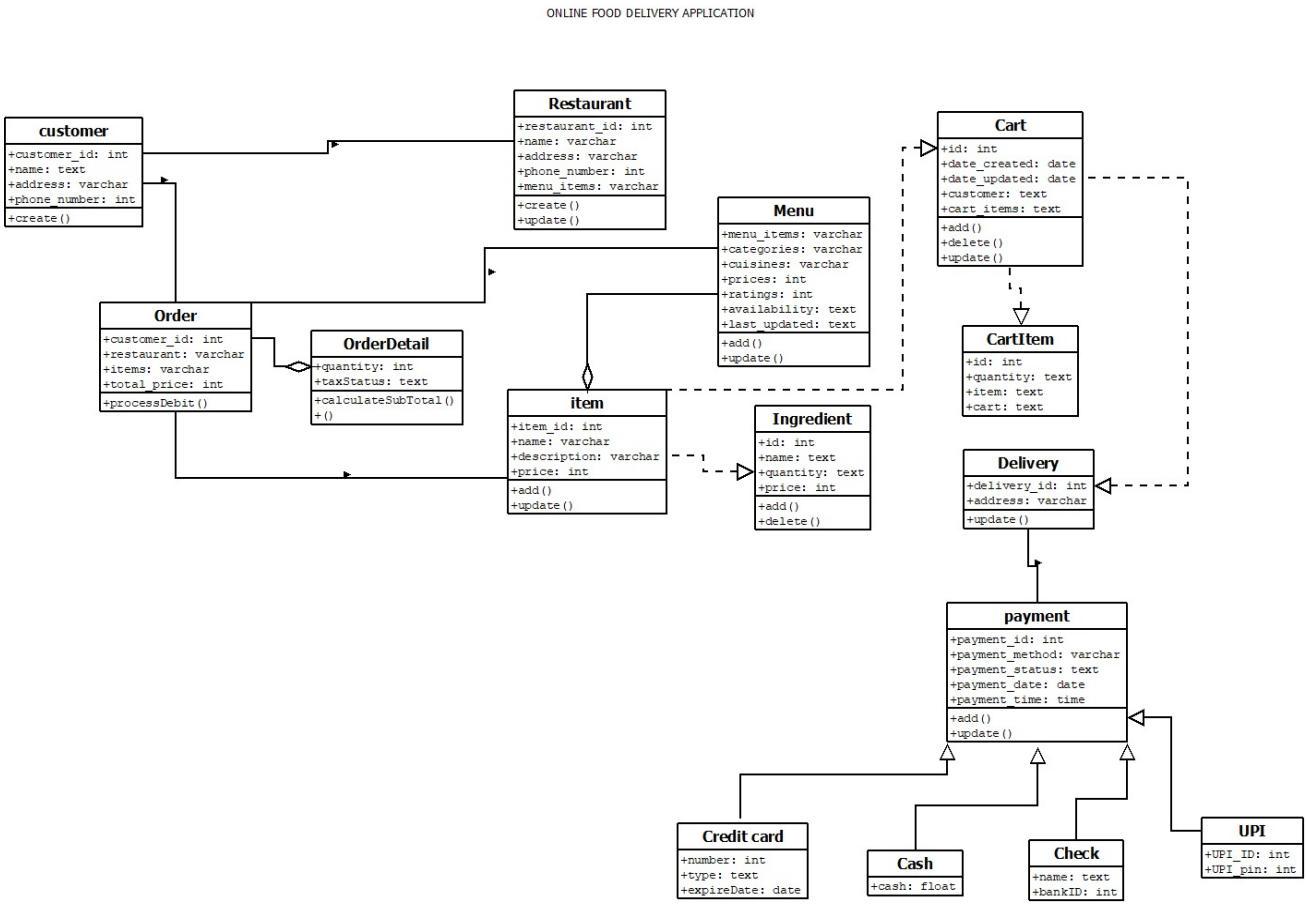
    [The reviews and the ratings will be taken from the user and then the data will be stored for the purpose of judging the total ratings of the delivery agent and the restaurant.]

14. Then, customer signs out from the application.

    [Logout is used for ending the session of that user from the application.

PART B:

B1.1 Draw a Class Diagram depicting the system.



B1.2 Discuss the architectural style and design pattern used to implement the design. Justify

The online food delivery system (Application) can be implemented using the Model-View-Controller (MVC) Architectural pattern.

Model-View-Controller (MVC) Architectural Pattern:

Architectural Style: Model-View-Controller (MVC)

The Model-View-Controller (MVC) architectural style is a widely used and well-established pattern for designing user interfaces and web applications. It provides a clear separation of concerns, where the model represents the data and business logic, the view displays the user interface, and the controller. Handles user input and updates the model and view accordingly.

The MVC design pattern is a widely used pattern in software development. It separates the system into three main components: the model, the view, and the controller.

The model is responsible for managing the data and business logic of the application, the view is responsible for presenting the data to the user, and the controller is responsible for handling user input and updating the model and view accordingly. In the case of an online food delivery system, the model would handle tasks such as managing orders and customer information, while the view would handle presenting menus and order status to the user. The controller would handle user input, such as placing orders or updating account information, and would update the model and view accordingly.

The benefits of using the MVC pattern include improved modularity and separation of concerns, which can lead to a more maintainable and scalable application. Additionally, the MVC pattern can help ensure that changes to one component do not affect the others, making it easier to update and modify the application over time. In the context of the online food delivery system, the model can represent data storage and business logic, such as managing orders, inventory, and user accounts. The view can be the user interface that displays the menu, order status, and delivery information. The controller can handle user input, such as placing an order, updating the order status, and processing payments.

In conclusion, the client-server architecture and MVC design pattern are suitable for the online food delivery system because they provide clear separation of concerns, modularity, and ease of maintenance. The online food delivery system can be implemented using the client-server architectural style and the Model-View-Controller (MVC) design pattern. The architectural style and design pattern used to implement a design can greatly impact the overall structure, organization, and maintainability of a software system. The choice of architectural style and design pattern should be based on various factors, such as the system's requirements, scalability, performance, and maintainability goals. In this context, let's discuss the architectural style and design. Pattern used to implement the design and justify the choice.

Implementation of the application by using MVC architecture:

Front-end (UI) Interface:

Model:

- Data models for menu items, orders, user accounts, and other relevant entities.
- Methods for fetching data from the server, such as retrieving menu items or user account details.
- Methods for updating data, such as placing an order or updating user account information.

View:

- UI components for displaying menu items, order details, and user account information.
- UI components for capturing user input, such as selecting menu items, entering delivery address, and making payments.
- Methods for updating the UI based on data received from the server or user input.

Controller:

- Methods for handling user actions, such as clicking on menu items, placing orders, and updating user account information.
- Methods for updating the model with new data, such as updating the order status or user account details.
- Methods for updating the view with new data or instructions, such as displaying order confirmation or error messages.

Back-end (Server-side) Interface:

Model:

- Database operations for storing and retrieving data, such as menu items, orders, and user accounts.
- Business logic for processing orders, calculating prices, and managing user accounts.
- APIs or interfaces for communication with the front-end (UI) components.

View:

- Not applicable, as the view is a client-side (front-end) component.

Controller:

- APIs or interfaces for handling incoming requests from the front-end (UI) components.
- Methods for processing user input, validating data, and updating the model accordingly.
- Methods for sending responses or instructions to the front-end (UI) components, such as returning order. confirmation or error messages.

The design patterns that can be used for the implementation of this food delivery system:

Design patterns are reusable solutions to common problems in software design. For a food delivery system, there are several design patterns that can be applied to make it efficient, scalable, and easy to maintain. Some of the design patterns that could be applied to a food delivery system are:

Command pattern:

- This is a design where there exists some buttons and user can control them accordingly to his usability. This is used like taking a command from the user and action will be done is the back-end. It turns requests into stand-alone objects that contain all information about the request.

Model-View-Controller (MVC) Pattern:

- MVC is a design pattern that separates the application into three main components: Model, View, and Controller. The Model represents the data and the business logic, the View represents the user interface, and the Controller acts as an intermediary between the Model and the View. Implementing the MVC pattern would help in separating concerns, making it easier to maintain and modify the system.

Observer Pattern:

- The Observer pattern is a design pattern that allows an object to notify other objects of any changes to its state. whenever the user cancels the order the change of the database state will occur. (in the tables of the database)

Singleton Pattern:

- The Singleton pattern is a design pattern that restricts the instantiation of a class to a single instance and provides a global point of access to it. In the context of a food delivery system, this pattern could be used to ensure that there is only one instance of the order processing system, to prevent duplicate orders and inconsistencies.

Design Pattern: Dependency Injection (DI) is a design pattern that allows objects to be loosely coupled, by injecting their dependencies rather than creating them internally. This promotes modularity, flexibility, and testability, as dependencies can be easily substituted or mocked during testing or runtime.

Implementation of these patterns would depend on the specific requirements of the food delivery system. However, here is a brief example of how these patterns could be implemented:

Command Pattern implementation:

- The command pattern can be implemented by using the registration function, where the user will give the input via the text and the details will be stored in the database and as soon as the user wants to login into the application the credentials will be validated by using the stored data present in the backend database which refers to the details of the registration.

MVC Pattern Implementation:

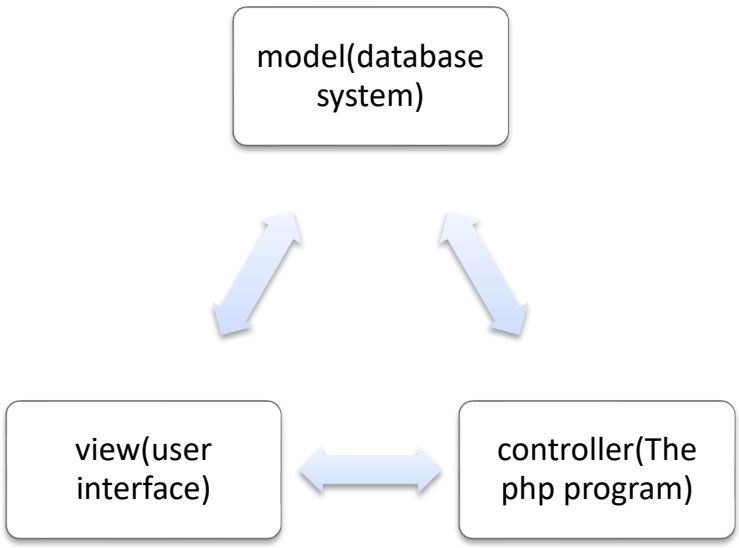
- The Model could represent the data and the business logic for the food delivery system, such as customer and order information. The View could represent the user interface, such as the website or mobile application used by customers to place orders. The Controller could act as an intermediary between the Model and the View, handling requests and updating the Model accordingly.

Observer Pattern Implementation:

- The order class could implement the Observer pattern, notifying the order class whenever the user cancels the order to change it state.

Singleton Pattern Implementation:

- The Order Processing System class could be implemented as a Singleton, ensuring that there is only one instance of the order processing system. This would prevent duplicate orders and inconsistencies in the system.



- The above figure represents the flow of the data from one component to other component.

B1.3 Discuss salient quality parameters that is addressed. Justify your design.

The design of an online food delivery system needs to address various quality parameters to ensure that the system is efficient, reliable, and user-friendly. Some of the key quality parameters that the design should address are:

Performance**:** The Application should be designed to handle a large number of users and orders without any performance degradation. The system should have an efficient database design and use caching and load-balancing techniques to ensure that it can handle the traffic.

Availability: The Application should be designed to be highly available and should have measures in place to handle failures gracefully. This could include using redundancy, failover mechanisms, and disaster recovery procedures.

Security: The Application should be designed to be secure and should have measures in place to prevent unauthorized access, data breaches, and other security threats. This could include using encryption, firewalls, intrusion detection and prevention systems, and secure coding practices.

Scalability**:** The Application should be designed to be scalable, so that it can easily accommodate growth in the number of users and orders. This could include using a microservices architecture, which allows the system to be broken down into smaller, more manageable components that can be scaled independently.

Usability**:** The Application should be designed to be user-friendly, so that customers can easily navigate the system and place orders. This could include using a simple and intuitive user interface, clear and concise instructions, and easy-to-use payment and checkout processes.

Maintainability**:** The Application should be designed to be easily maintainable, so that it can be updated and enhanced without causing disruption to the system. This could include using modular design and clean coding practices, as well as providing documentation and training for system administrators and developers.

In order to address these quality parameters, the design of the online food delivery system should incorporate various architectural styles and design patterns. For example, using a microservices architecture can help to improve scalability and maintainability, while using a layered architecture can help to improve security and performance. The use of design patterns such as the Model-View-Controller (MVC) pattern can help to improve usability by separating the user interface from the underlying data and business logic. Overall, a well-designed online food delivery system should consider these quality parameters and use appropriate architectural styles and design patterns to ensure that the system is efficient, reliable, and user-friendly.

- **Justification:**

The Dependency Injection (DI) pattern is chosen for the following reasons:

Loose Coupling: DI promotes loose coupling between components, as dependencies are injected rather than hard-coded. This reduces dependencies between components and makes the system more maintainable and extensible.

Testability: DI allows for easy substitution of dependencies during testing, as dependencies can be easily mocked or replaced with test doubles. This makes unit testing and integration testing more feasible and effective.

Modularity: DI promotes modularity, as components can be developed independently and combined using dependency injection. This makes the system more modular and promotes code reusability.

Flexibility: DI allows for flexibility in choosing and changing dependencies at runtime, making it suitable for dynamic and changing environments. This enables the system to adapt to changing requirements and configurations without impacting the overall system architecture.

- Here we have defined many design patterns according to the food delivery system, from them we have implemented some of the design patterns which makes the design precise:
- We have implemented the singleton design pattern, while user is ordering the food the time and date of the order time will be taken into consideration which prevents to form the duplicate orders.
- We have implemented the command design pattern, the user is free to use the application to operate however he likes, which represents the request of the user will be satisfied by the operation made by the application.
- We have implemented the observer design pattern, whenever the user cancels the order the application notifies the database to change the state, such that the changes will occur in the order booking table of the database accordingly.

Implementation of the application:

Registration:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>ZWIGGY DELIVERY- Register</title>
    <style>
   html{
     background-image: url('CW1.jpg');
     height: 100%;
     background-repeat: no-repeat;
     background-position: center;
     background-size: cover;
   }
form {
    border-radius: 5px;
    width: 300px;
}

h1 {

    font-family: 'Trebuchet MS', 'Lucida Sans Unicode', 'Lucida Grande', 'Lucida Sans',
Arial, sans-serif;
    color: white;
    text-align: center;
    margin-top: 0;
}
input[type=text], input[type=password], input[type=email] {
    padding: 10px;
    width: 100%;
    border-radius: 5px;
    border: 1px solid #ccc;
    margin-bottom: 20px;
    box-sizing: border-box;
}
input[type=submit] {
    background-color: #4CAF50;
```

```
    color: white;
    padding: 12px 20px;
    border: none;
    border-radius: 4px;
    cursor: pointer;
    width: 100%;
}
input[type=submit]:hover {
    background-color: #45a049;
}
label {
    color:white;
    font-weight: bold;
}
    </style>
</head>

<body>
<div>
    <form action="" method="POST">
        <h1>Registration</h1>
        <label for="Username">UserName</label>
        <input type="text" id="Username" name="Username" required>
        <label for="email">Email</label>
        <input type="email" id="email" name="email" required>
        <label for="Phone_number">Phone Number</label>
        <input type="text" id="P_N" name="P_N" required>
        <label for="password">Password</label>
        <input type="password" id="password" name="password" required>
        <label for="confirm-password">Confirm Password</label>
        <input type="password" id="confirm-password" name="confirm-password" required>
        <input type="submit" value="Register">
        Click here to <a href="li.php" title="Login"> Login

    </form>
</div>
<?php

 $Name = $id = $Contact = $Password = "" ;

 if($_SERVER["REQUEST_METHOD"]=="POST"){
     $Name = $_POST['Username'];
     $Id = $_POST['email'];
     $Contact = $_POST['P_N'];
     $Password = $_POST['password'];

     $con = mysqli_connect("localhost","root","","T");

     $query = "INSERT INTO `registration` (username,email,phone_number,password) VALUES
('$Name','$Id', '$Contact', '$Password')";
                 $result = mysqli_query($con,$query);
                 if($result){
      echo ("Registration Succesful");
                 }
                             eise
```

```
                            {

                                echo ("Fail");
                            }
                }

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

Fig 1.1: The above figure represents the php program code for the registration of the user into application.

Login:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>TEESKPO</title>
  <style>
    html{
    background-image: url('CW.jpeg');
    width:100%;
    height: 100%;
     background-repeat: no-repeat;
     background-position: center;
     background-size: cover;
   }

  .login-box {
    width: 350px;
    margin: auto;
    padding: 30px;
    margin-top: 130px;
  }

  h1 {

    font-family: 'Trebuchet MS', 'Lucida Sans Unicode', 'Lucida Grande', 'Lucida Sans',
Arial, sans-serif;
    text-align: center;
    font-size: 36px;
    margin-bottom: 30px;
  }

  form {
    display: flex;
    flex-direction: column;
  }

  label {
    font-weight: bolder;
    font-size: 18px;
    margin-bottom: 10px;
  }
```

```
  input[type="text"],
  input[type="password"] {
    border-radius: 5px;
    border: none;
    padding: 10px;
    margin-bottom: 20px;
  }

  button[type="submit"] {
    background-color: hsl(122, 39%, 49%);

    color: rgb(255, 255, 255);
    border: none;
    border-radius: 5px;
    padding: 10px;
    font-size: 18px;
    cursor: pointer;
  }

  button[type="submit"]:hover {
    background-color: #3e8e41;
  }

  </style>
</head>
<body>
  <div class="login-box">
    <h1>Login</h1>
    <form role="form" name="login" action="" method="post">
      <label for="email">Email id:</label>
      <input type="text" id="email" name="email" required>
      <label for="password">Password:</label>
      <input type="password" id="password" name="password" required>
      <button type="submit"  name="submit" value="Login" >Login</button>
      <br>
        Click here to <a href="reg.php" title="Registration"> Register here
    </form>
  </div>

  <?php

        if($_SERVER['REQUEST_METHOD']=='POST'){
        $email = $_POST['email'];

        $password = $_POST['password'];


        $con = mysqli_connect("localhost","root","","T");

        if (mysqli_connect_errno())
                  {
                  echo "Failed to connect to MySQL: " .mysqli_connect_error();
```

```
                    }



      $query = "SELECT * FROM registration WHERE email='$email' and password='$password'";
                        $result = mysqli_query($con,$query);
                        if($result)
                        {
                            if(mysqli_num_rows($result)>0)
                            {
                                session_start();
                                $_SESSION['email'] = $email;

                                header("Location: hm.php");
                            }
                            else{
                                echo("Invalid credenatials");
                            }

                        }

        }

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

Fig 1.2: The above figure represents the php program code for the log-in of the user into application.

Home page:

```
<!DOCTYPE html>
<html>
<head>
<title>SSS Food Court</title>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/5.15.4/css/all.min.css">
<style>
body {font-family: "Times New Roman", Georgia, Serif;}
h1, h2, h3, h4, h5, h6 {
  font-family: "Playfair Display";
  letter-spacing: 5px;
}
.btn:hover{
    color: blue;
}
.header .a .logo i{
    color:azure;
    font-size:2rem;
```

```
}
.header .a .logo:hover{
    color:black;
}
@import
url('https://fonts.googleapis.com/css2?family=Roboto:wght@100;300;400;500;700&display=swap'
);

</style>
</head>
<body>

<!-- Navbar (sit on top) -->
<div class="header">
  <div class="a" style="letter-spacing:4px;">
    <a href="#home" class="logo"><i class="fa-solid fa-fork-knife"></i>Tiskpoo</a>
    <!-- Right-sided navbar links. Hide them on small screens -->
    <div class="w3-right w3-hide-small">
      <a href="#home" class="w3-bar-item w3-button">Home</a>
      <a href="#about" class="w3-bar-item w3-button">About</a>
      <a href="#menu" class="w3-bar-item w3-button">Menu</a>
      <a href="menu.php" class="w3-bar-item w3-button">Order</a>
      <a href="#contact" class="w3-bar-item w3-button">Contact</a>
      <a href="orders.php" class="w3-bar-item w3-button">View Orders</a>
      <a href="lo.php" class="w3-bar-item w3-button">Logout</a>
    </div>
  </div>
</div>

<!-- Header -->
<header class="w3-display-container w3-content w3-wide" style="max-width:1600px;min-
width:500px" id="home">
  <img class="w3-image" src="food4.jpg" alt="Hamburger Catering" width="1600" height="800">
  <div class="w3-display-bottomleft w3-padding-large w3-opacity">
    <h1 class="w3-xxlarge">SSS Food Court</h1>
  </div>
</header>

<!-- Page content -->
<div class="w3-content" style="max-width:1100px">

  <!-- About Section -->
  <div class="w3-row w3-padding-64" id="about">
    <div class="w3-col m6 w3-padding-large w3-hide-small">
     <img src="food5.jpg" class="w3-round w3-image w3-opacity-min" alt="Table Setting"
width="700" height="850">
    </div>

    <div class="w3-col m6 w3-padding-large">
      <h1 class="w3-center">About</h1><br>
      <h5 class="w3-center">Tradition since 1995</h5>
      <p class="w3-large">SSS Food Court is a restaurent with cozy look and spacious.Also 5
star restaurent with delcious food and we provides you the best services.Enjoy your
food!!!</p>
```

```
    </div>
  </div>

  <hr>


  <!-- Menu Section -->
  <div class="w3-row w3-padding-64" id="menu">
    <div class="w3-col l6 w3-padding-large">

      <h1 class="w3-center">Our Menu</h1><br>
      <h4>Bread Basket</h4>
      <p class="w3-text-grey">Assortment of fresh baked fruit breads and muffins</p>
      <p>Price:RS.100</p>
      <label for="Quantity">Quantity:</label>
      <input type="number" id="Quantity" name="Quantity" placeholder="Enter Quantity:"><br>
      <a href="#" class="btn">Add to Cart</a><br>

      <h4>Honey Almond Granola with Fruits</h4>
      <p class="w3-text-grey">Natural cereal of honey toasted oats, raisins, almonds and
dates</p>
      <p>Price:RS.150</p>

      <label for="Quantity">Quantity:</label>
      <input type="number" id="Quantity" name="Quantity" placeholder="Enter Quantity:"><br>
      <a href="#" class="btn">Add to Cart</a><br>

      <h4>Belgian Waffle</h4>
      <p class="w3-text-grey">Vanilla flavored batter with malted flour</p>
      <p>Price:Rs.120</p>
      <label for="Quantity">Quantity:</label>
      <input type="number" id="Quantity" name="Quantity" placeholder="Enter Quantity:"><br>
      <a href="#" class="btn">Add to Cart</a><br>

      <h4>Scrambled eggs</h4>
      <p class="w3-text-grey">Scrambled eggs, roasted red pepper and garlic, with green
onions</p>
      <p>Price:Rs.110</p>
      <label for="Quantity">Quantity:</label>
      <input type="number" id="Quantity" name="Quantity" placeholder="Enter Quantity:"><br>
      <a href="#" class="btn">Add to Cart</a><br>

      <h4>Blueberry Pancakes</h4>
      <p class="w3-text-grey">With syrup, butter and lots of berries</p>
      <p>Price:Rs.75</p>
      <label for="Quantity">Quantity:</label>
      <input type="number" id="Quantity" name="Quantity" placeholder="Enter Quantity:"><br>
      <a href="#" class="btn">Add to Cart</a>
    </div>
  </div>

  <hr>
```

```
  <!-- Contact Section -->
  <div class="w3-container w3-padding-64" id="contact">
    <h1>Contact</h1><br>
    <p>We offer full-service for any event, large or small. We understand your needs and we
will deliver the food to satisfy the biggerst criteria of them all, both look and taste. Do
not hesitate to contact us.</p>
    <p class="w3-text-blue-grey w3-large"><b>Marathalli,Bangalore Karnataka 560090</b></p>
    <p>You can also contact us by phone 00553123-2323 or email sss@gmail.com, or you can
send us a message here:</p>
    <form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]); ?>">
    <label for="email">Email:   </label>
    <input type="email" id="email" name="email" required><br><br>

    <label for="name">Name:</label>
    <input type="text" id="name" name="name"><br><br>
    <label for="text">Contact number:</label>
    <input type="text" id="c_n" name="c_n"><br><br>
    <label for="nop">No of persons:</label>
    <input type="nop" id="nop" name="nop"><br><br>
    <label for="date">Date:</label>
    <input type="date" id="date" name="date"><br><br>
    <label for="ds">Description:</label>
    <input type="ds" id="ds" name="ds"><br><br>
    <input type="submit" name='sm' value="send message">

  </div>

<!-- End page content -->
</div>

<!-- Footer -->
<footer class="w3-center w3-light-grey w3-padding-32">
  <p>Powered by <a href="https://www.w3schools.com/w3css/default.asp" title="W3.CSS"
target="_blank" class="w3-hover-text-green">Tiskpoo</a></p>
</footer>

<?php


    $servername = "localhost";
    $username = "root";
    $password = "";
    $dbname = "T";


    $conn = new mysqli($servername, $username, $password, $dbname);


    if ($conn->connect_error) {
      die("Connection failed: " . $conn->connect_error);
    }

    if (isset($_POST['sm'])) {
      $Id = $_POST['email'];
```

```
      $name = $_POST["name"];
      $cn=$_POST["c_n"];
      $nop = $_POST["nop"];
      $date = $_POST["date"];
      $ds = $_POST["ds"];


      $sql = "INSERT INTO contact(email,name,phone_number,people,date,description)
          VALUES ('$Id','$name','$cn','$nop', '$date','$ds')";


      if ($conn->query($sql) === TRUE) {
        echo "message sent   successfully!";
      } else {
        echo "Error: " . $sql . "<br>" . $conn->error;
      }
    }


    $conn->close();
  ?>

</body>
</html>
```

Fig 1.3: The above figure represents the php program code for the home page of the application.

Place Order:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Sofia">
    <title>TEESKPO</title>
    <style>
        html{
            background-color: azure;
            background-image: url("cw4.jpeg");
            color: azure;
            text-align: center;
            border: 10px;
            padding-top:80px;
            background-repeat: no-repeat;
            font-family: 'Trebuchet MS', 'Lucida Sans Unicode', 'Lucida Grande', 'Lucida
Sans', Arial, sans-serif;
            height: 100%;
            background-repeat: no-repeat;
            background-position: center;
            background-size: cover;
        }
```

```
        h1{
            color: black;
            text-align: center;
            font-family: 'Trebuchet MS', 'Lucida Sans Unicode', 'Lucida Grande', 'Lucida
Sans', Arial, sans-serif;
            text-decoration: underline;
        }
        input[type="text"]{
            border-radius: 5px;
            border-style:ridge;
            padding: 2px;
            margin-bottom: 20px;
        }
        form:label{
            background-color: azure;
        }
        ul {
        list-style-type: none;
        margin: 0;
        padding: 0;
        overflow: hidden;
        background-color: #333;
        }

        li {
            float:left;
            }

        li a {
            display: block;
            color: white;
            text-align: center;
            padding: 12px 16px;
            text-decoration: none;
    }

li a:hover {
  background-color: #111;
}
        .cap{
            text-align:center;
            font-family: "Sofia", sans-serif;
        }
        .cap span{
            background-color:aliceblue;

        }
    </style>
</head>
    <body>
        <h1 style=color:aliceblue;>ORDER NOW!!!</h1>
        <form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]); ?>">
        <label for="email">Email:   </label>
        <input type="email" id="email" name="email" required><br><br>
```

```
        <label for="name">Name:</label>
        <input type="text" id="name" name="name"><br>

        <label for="food">Select the Food:</label><br>
            <input type="checkbox" id="bb" name="food[]" value="Bread Basket">
            <label for="vehicle1"> Bread Basket</label><br>
            <input type="checkbox" id="hag" name="food[]" value="Honey Almond Granola with
Fruits">
            <label for="hag"> Honey Almond Granola with Fruits</label><br>
            <input type="checkbox" id="bw" name="food[]" value="Belgian Waffle">
            <label for="bw"> Belgian Waffle</label><br>
            <input type="checkbox" id="se" name="food[]" value="Scrambled eggs">
            <label for="se"> Scrambled eggs</label><br>
            <input type="checkbox" id="bp" name="food[]" value="Blueberry Pancakes">
            <label for="bp"> Blueberry Pancakes</label> <br><br>


        <label for="ad">Address:</label>
        <input type="text" id="ad" name="ad"><br>
        <?php
        date_default_timezone_set("Asia/Kolkata");
        ?>
        <input type="text" name="date" value="<?php echo date('Y-m-d H:i:s'); ?>">
        <input type="submit" value="place order">
    </form>
    <p>Click here to view bookings<a href="orders.php" title="bookings"> My bookings</p>


    </section>
    <br>

    <?php


        $servername = "localhost";
        $username = "root";
        $password = "";
        $dbname = "T";


        $conn = new mysqli($servername, $username, $password, $dbname);


        if ($conn->connect_error) {
            die("Connection failed: " . $conn->connect_error);
          }


          if ($_SERVER["REQUEST_METHOD"] == "POST") {
            $email = $_POST["email"];
            $name = $_POST["name"];
            $address = $_POST["ad"];
            $date = $_POST["date"];
            $food = $_POST["food"];
```

```
            foreach ($food as $selected_food) {
              $sql = "INSERT INTO ord (email, name, address, date, food) VALUES ('$email',
'$name', '$address', '$date', '$selected_food')";

              if ($conn->query($sql) === FALSE) {
                echo "ORDER FAILED TO PLACE";
              }
            }

            $conn->close();
          }

          ?>

</body>

</html>
```

Fig 1.4: The above figure represents the php program code for the ordering of food by the user from the application.

View Orders:

```
<?php
if(isset($_POST['submit'])){
    $email = $_POST['email'];
$conn = mysqli_connect("localhost", "root", "", "T");
$sql = "SELECT id,name,address,date,food FROM ord WHERE email = '$email'";


$result = mysqli_query($conn, $sql);


if (mysqli_num_rows($result) > 0) {
    echo "<center><h2>Your Bookings</h2></center>";
    echo "<center><table>";
    echo "<tr><th>ORDER ID</th><th>ORDERED
BY</th><th>ADDRESS</th><th>ORDERED_AT</th><th>ORDERED_FOOD</th></tr>";
    while ($row = mysqli_fetch_assoc($result)) {
        echo "<tr><td>" . $row["id"] . "</td><td>" . $row["name"] . "</td><td>" .
$row["address"] . "</td><td>" .$row["date"] ."</td><td>" .$row["food"] ."</td></tr>";
    }
    echo "</center></table>";
} else {
    echo "No bookings found on this email";
}

mysqli_close($conn);
}
?>

<!DOCTYPE html>
```

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>

</head>
<style>
        html{
            background-color: azure;
            color: rgb(0, 0, 0);
            text-decoration: underline;
            font-family: 'Trebuchet MS', 'Lucida Sans Unicode', 'Lucida Grande', 'Lucida
Sans', Arial, sans-serif;
        }
        form {
            display: flex;
            flex-direction: column;
            align-items: center;
            margin-top: 50px;
        }

        label {
            margin-bottom: 10px;
            font-weight: bold;
        }

        input[type="text"] {
            padding: 10px;
            border: 1px solid #ccc;
            border-radius: 4px;
            font-size: 16px;
            margin-bottom: 20px;
            width: 300px;
            max-width: 100%;
        }

        input[type="submit"] {
            background-color: #4CAF50;
            color: white;
            padding: 10px 20px;
            border: none;
            border-radius: 4px;
            cursor: pointer;
            font-size: 11px;
            transition: background-color 0.3s ease;
        }

        input[type="submit"]:hover {
            background-color: #3e8e41;
        }
        </style>
<body>
```

```
    <div class="container">
    <form method="post" action="">
    <label for="email">Enter Your email-id:</label>
    <input type="text" name="email" placeholder="Your email address">
    <input type="submit" name="submit" value="view_bookings">
    <br>
        Click here to cancel your booking<a href="cancel.php" title="cancel"> Cancellation
</form>
</div>
</body>
</html>
```

Fig 1.5: The above figure represents the php program code for the viewing of the orders ordered by the user.

Cancel order:

```
<?php

    $servername = "localhost";
    $username = "root";
    $password = "";
    $dbname = "T";


    $conn = new mysqli($servername, $username, $password, $dbname);


    if ($conn->connect_error) {
        die("Connection failed: " . $conn->connect_error);
    }

    if ($_SERVER["REQUEST_METHOD"] == "POST") {
        $Id = $_POST['order_id'];
        $email=$_POST['email'];


        $sql = "DELETE FROM ord where id='$Id' AND email='$email'";


        if ($conn->query($sql) === TRUE) {
            echo "order cancelled successfully!";
        } else {
            echo "Error: " . $sql . "<br>" . $conn->error;
        }
    }


    $conn->close();
?>

<!DOCTYPE html>
<html lang="en">
```

```
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <!-- <link rel="stylesheet" href="css/rf.css"> -->
</head>
<style>
        html{
            background-color: azure;
            color: rgb(0, 0, 0);
            text-decoration: underline;
            font-family: 'Trebuchet MS', 'Lucida Sans Unicode', 'Lucida Grande', 'Lucida
Sans', Arial, sans-serif;
        }
        form {
            display: flex;
            flex-direction: column;
            align-items: center;
            margin-top: 50px;
        }

        label {
            margin-bottom: 10px;
            font-weight: bold;
        }

        input[type="int"] {
            padding: 10px;
            border: 1px solid #ccc;
            border-radius: 4px;
            font-size: 16px;
            margin-bottom: 20px;
            width: 300px;
            max-width: 100%;
        }

        input[type="text"] {
            padding: 10px;
            border: 1px solid #ccc;
            border-radius: 4px;
            font-size: 16px;
            margin-bottom: 20px;
            width: 300px;
            max-width: 100%;
        }

        input[type="submit"] {
            background-color: #4CAF50;
            color: white;
            padding: 10px 20px;
            border: none;
            border-radius: 4px;
            cursor: pointer;
```

```
            font-size: 11px;
            transition: background-color 0.3s ease;
        }

        input[type="submit"]:hover {
            background-color: #3e8e41;
        }
        </style>
<body>
    <div class="container">
    <form method="post" action="">
    <label for="email">Enter Your email-id:</label>
    <input type="text" name="email" placeholder="Your email address">
    <label for="book_id">Enter Your order-id:</label>
    <input type="int" name="order_id" placeholder="order id">
    <input type="submit" name="submit" value="cancel">
    <br>
        Click here to view bookings<a href="orders.php" title="bookings"> My bookings
</form>
</div>
</body>
</html>
```

Fig 1.6: The above figure represents the php program code for the cancellation of the orders by the user.

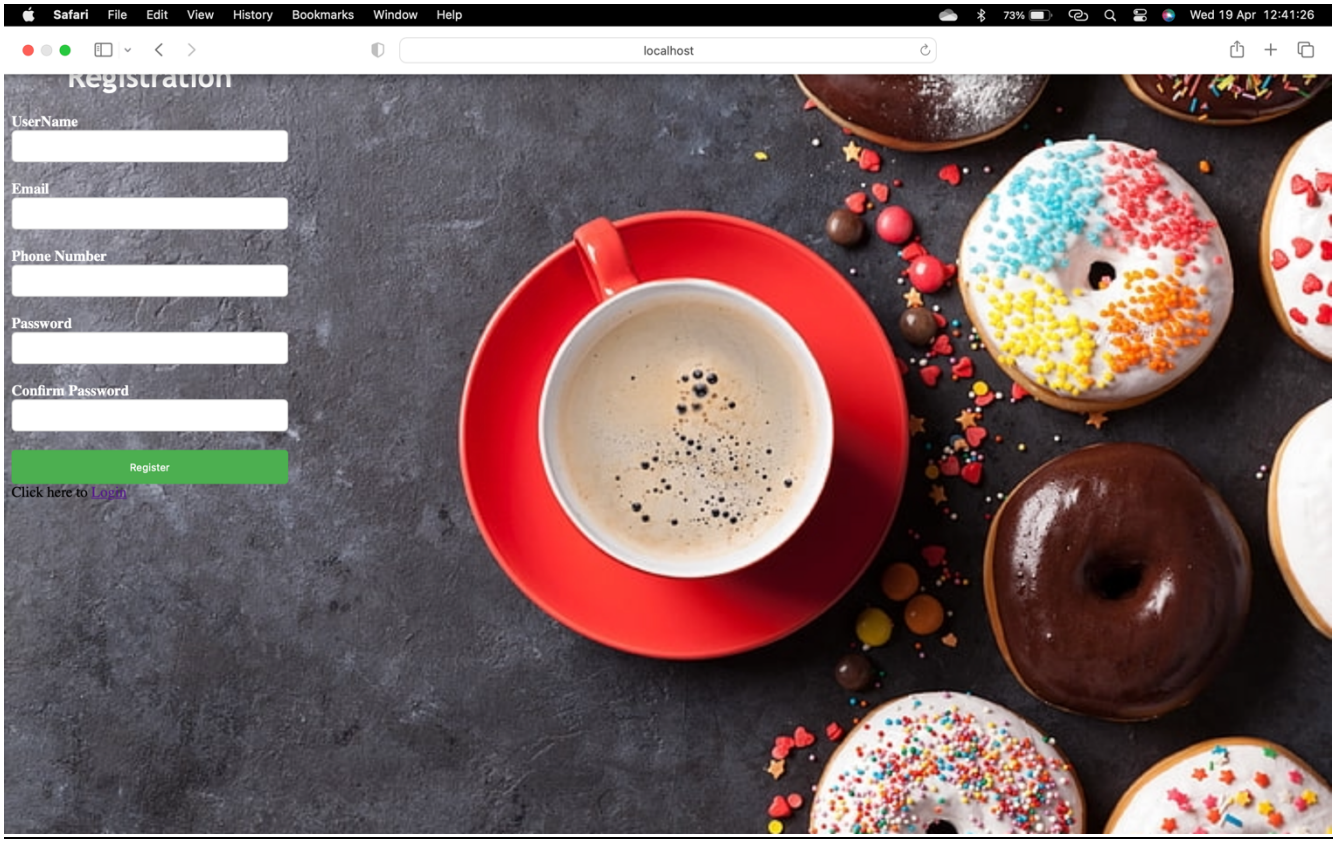
Logout:

```
<?php
session_start();
$_SESSION = array();
session_destroy();
header("Location: li.php");
exit;
?>
```

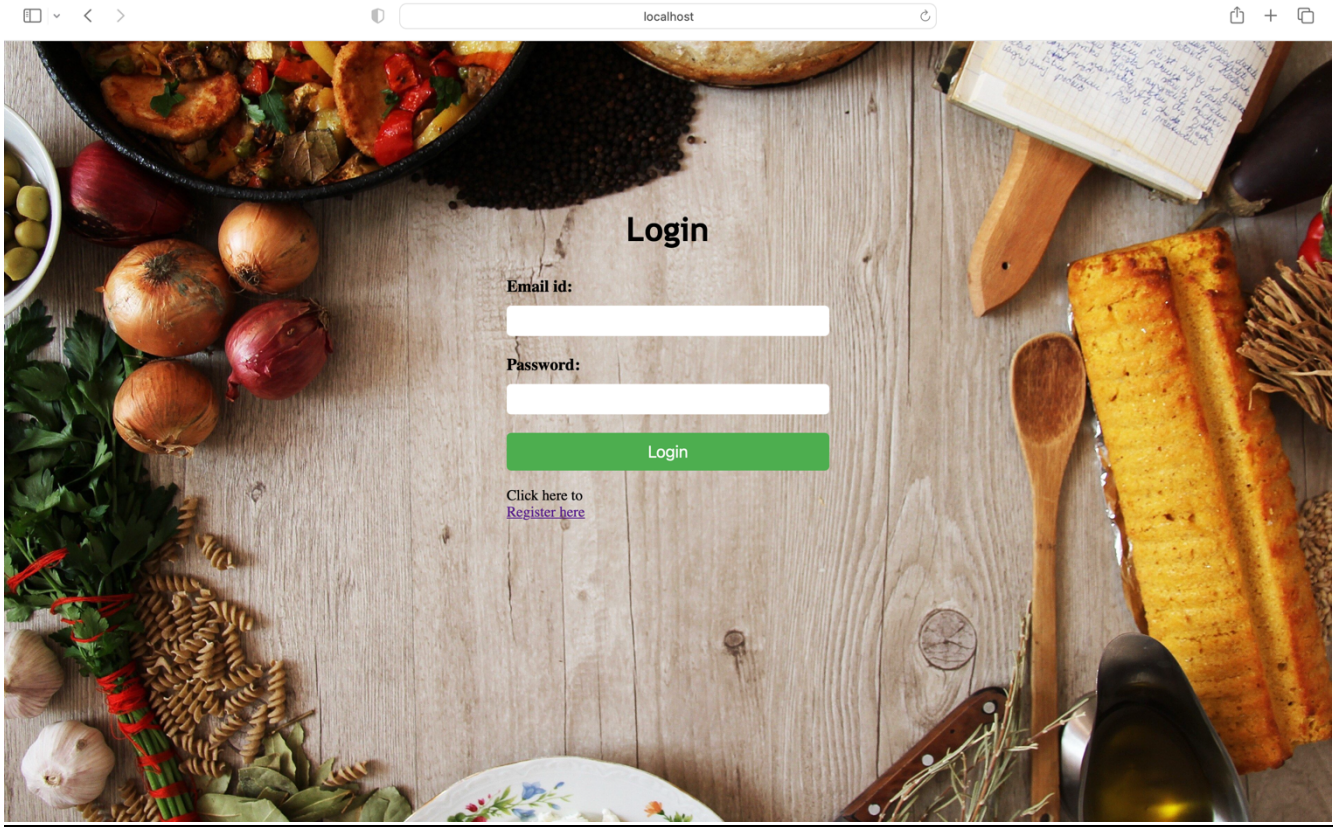
Fig 1.7: The above figure represents the php program code for the logout of the user from the application.

Presentation of the results:

Registration:

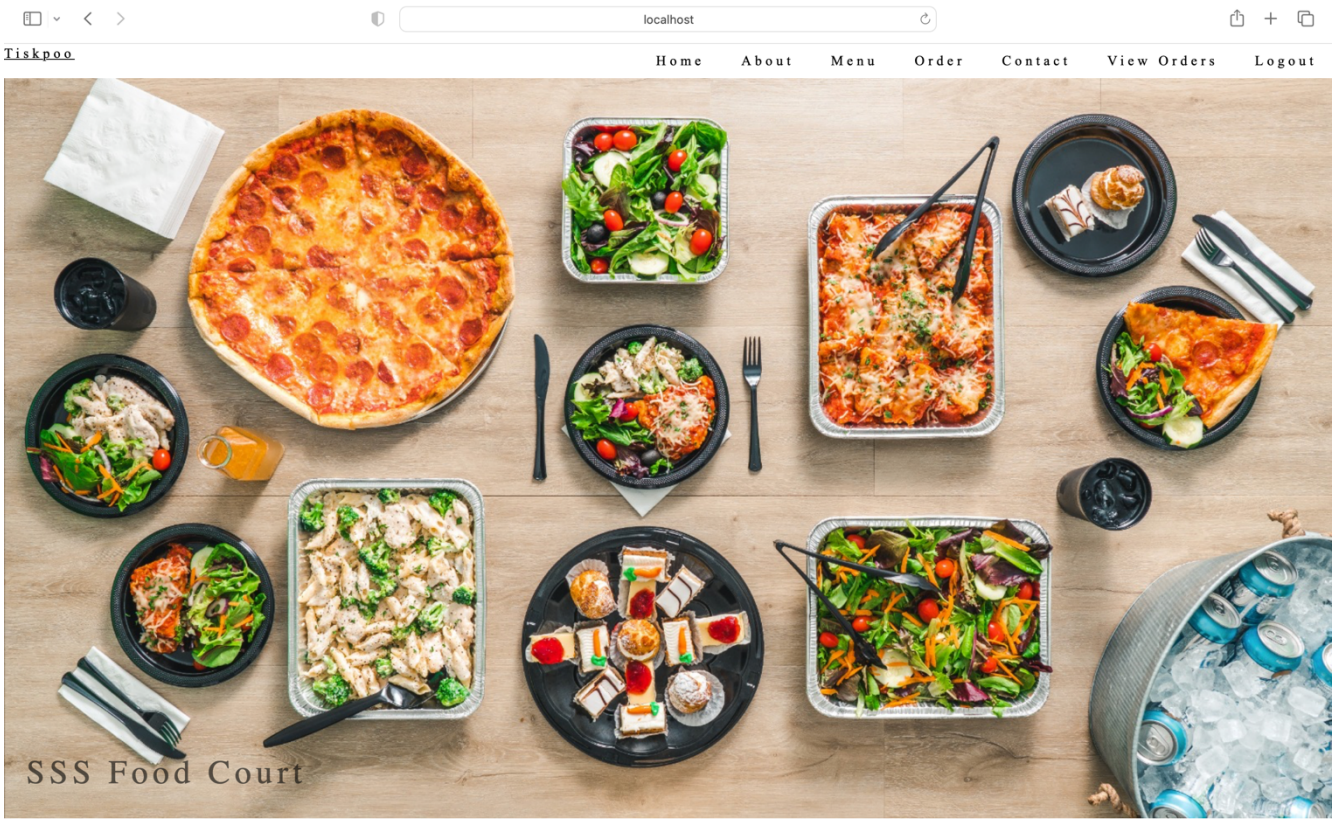


Login:



RAMAIAH UNIVERSITY OF APPLIED SCIENCES

Home page:



About us:

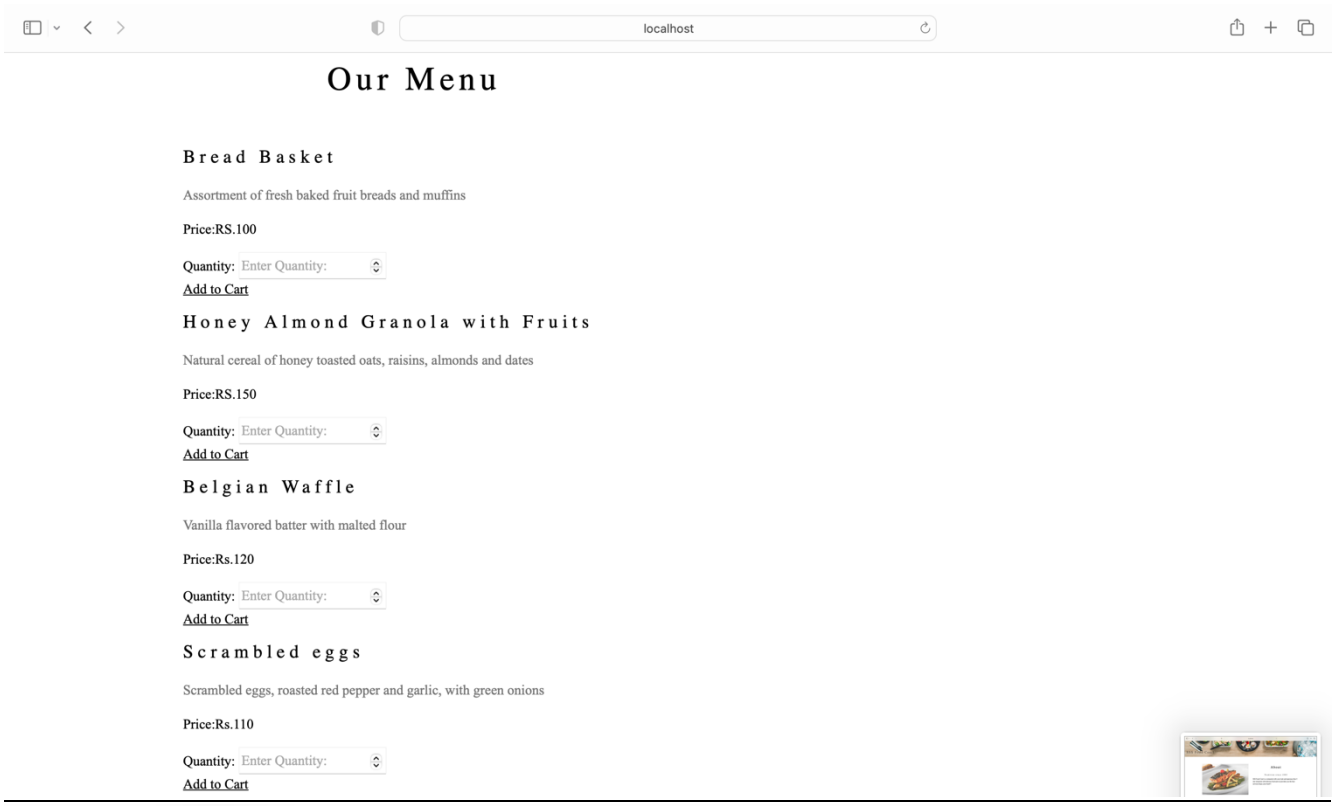
About

Tradition since 1995

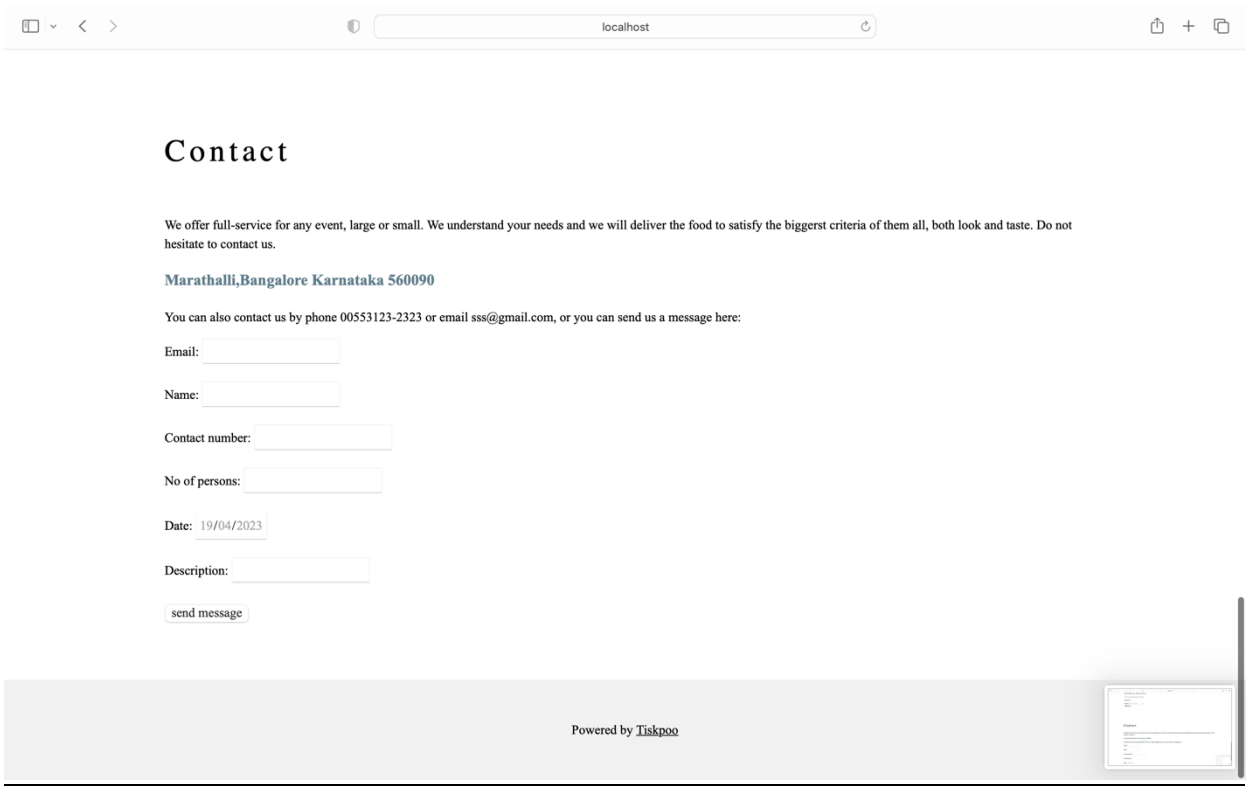
SSS Food Court is a restaurent with cozy look and spacious.Also 5 star restaurent with delcious food and we provides you the best services.Enjoy your food!!!

RAMAIAH UNIVERSITY OF APPLIED SCIENCES

Menu:

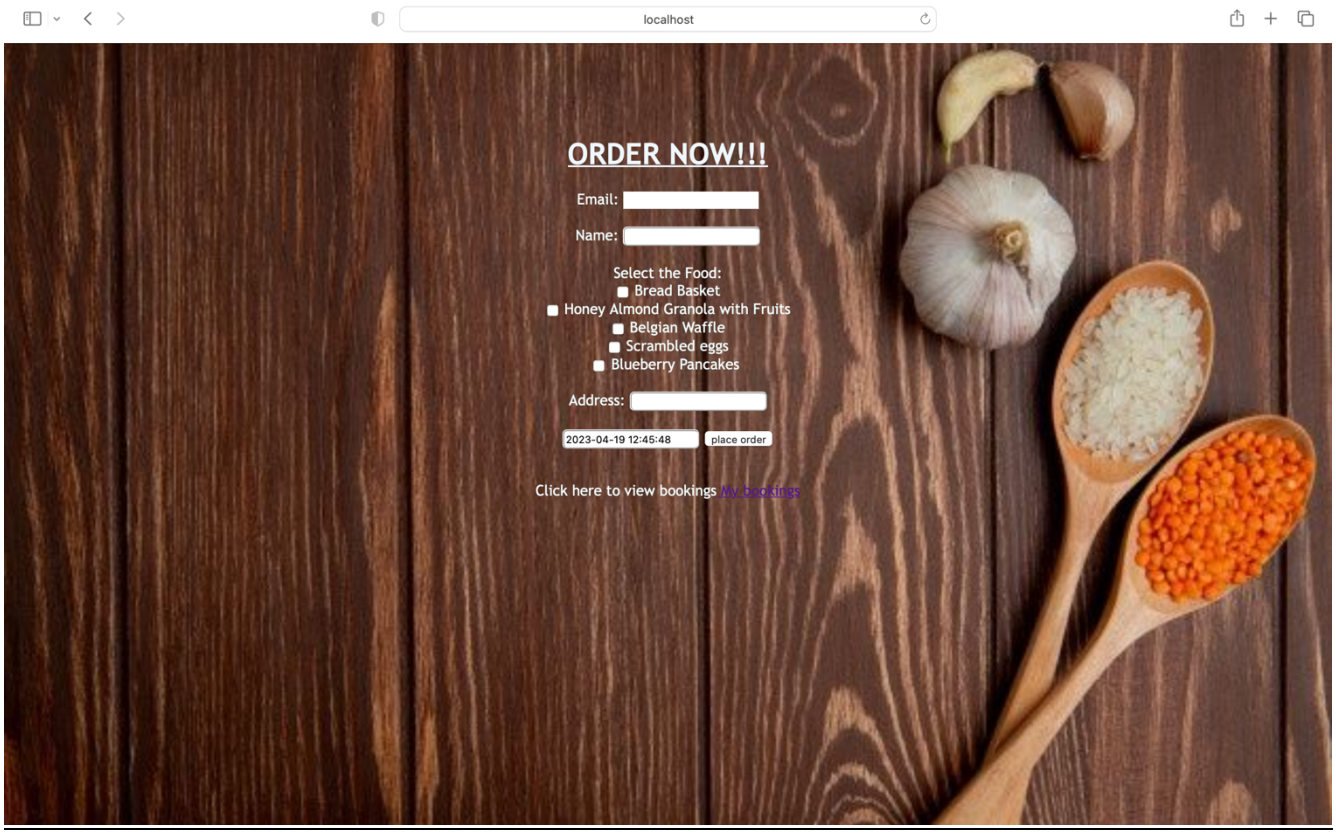


Contact us:

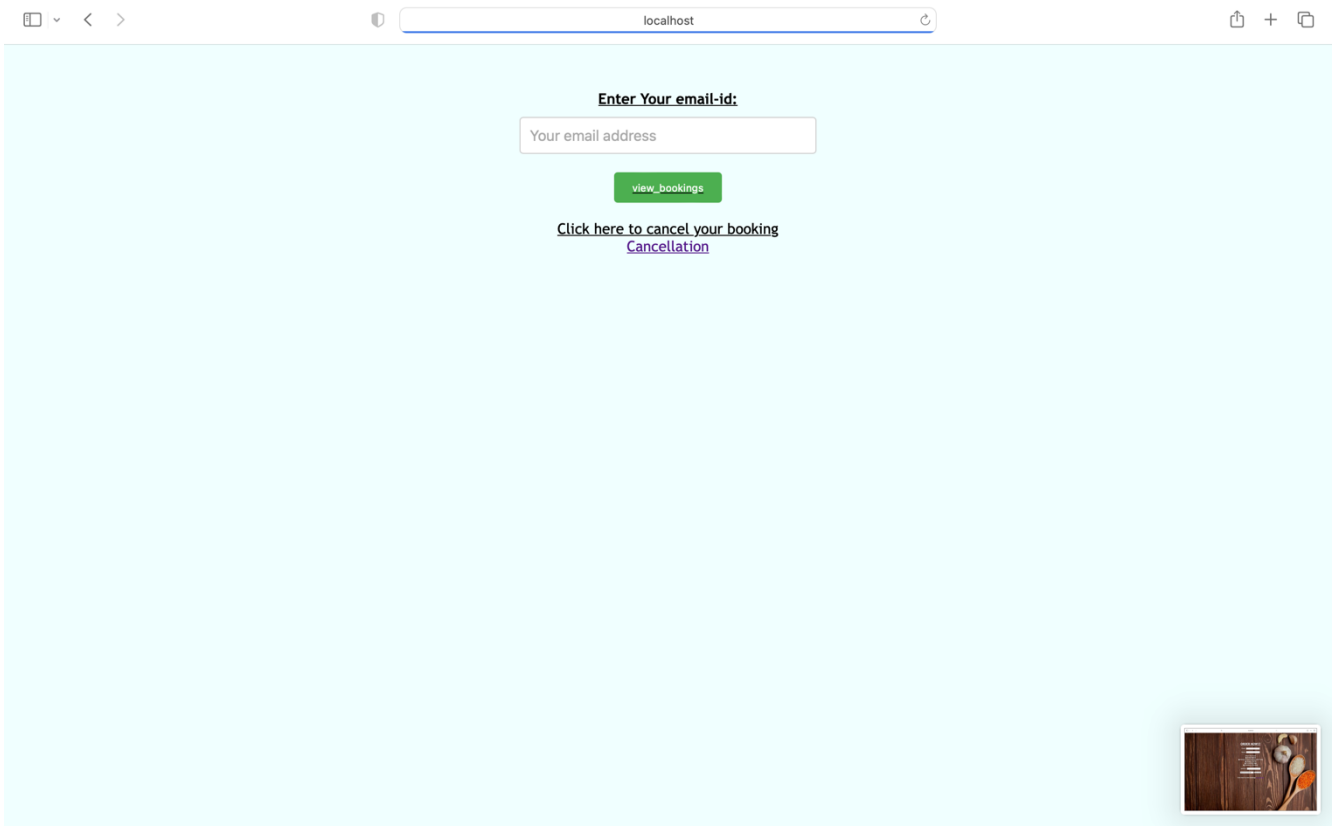


RAMAIAH UNIVERSITY OF APPLIED SCIENCES

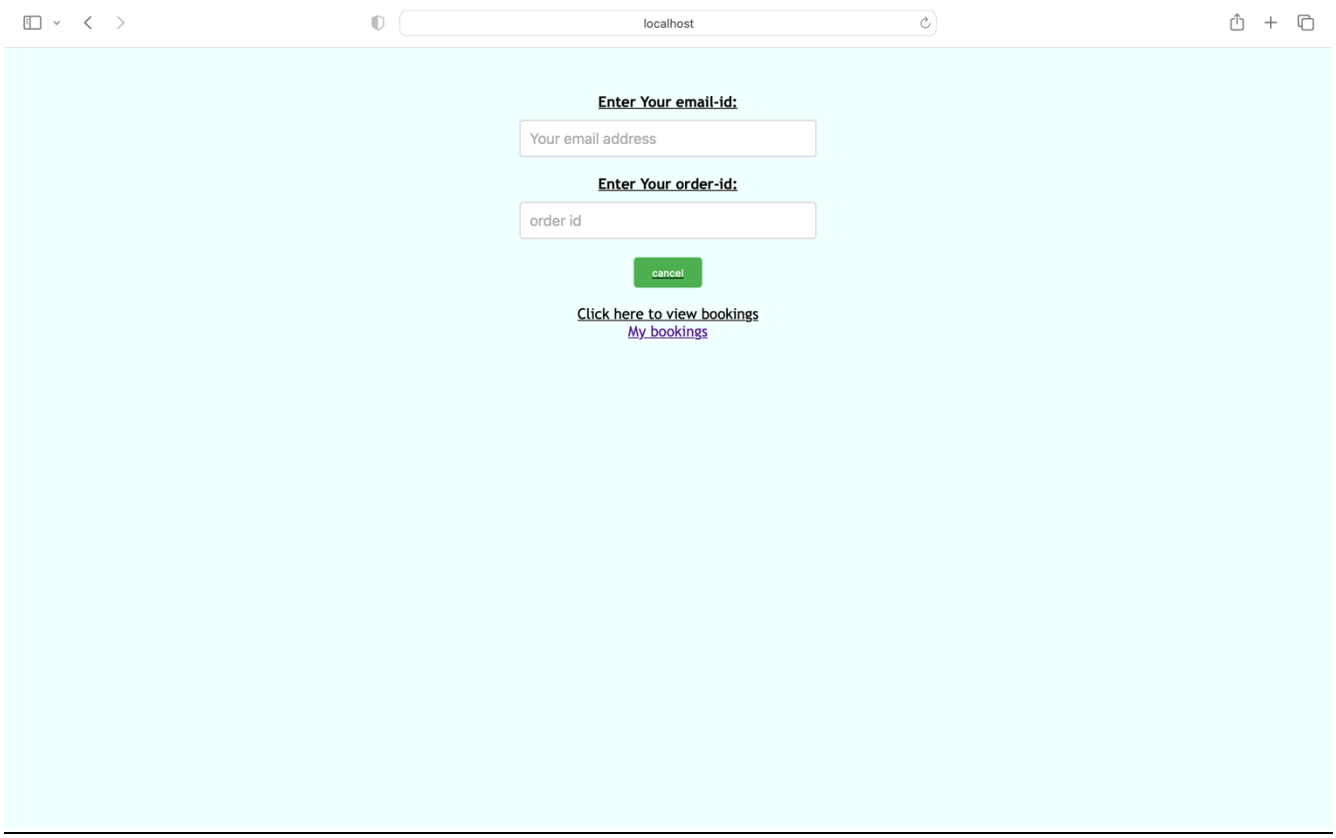
Order food:



View orders:



Cancel order:



Implementation of the database:

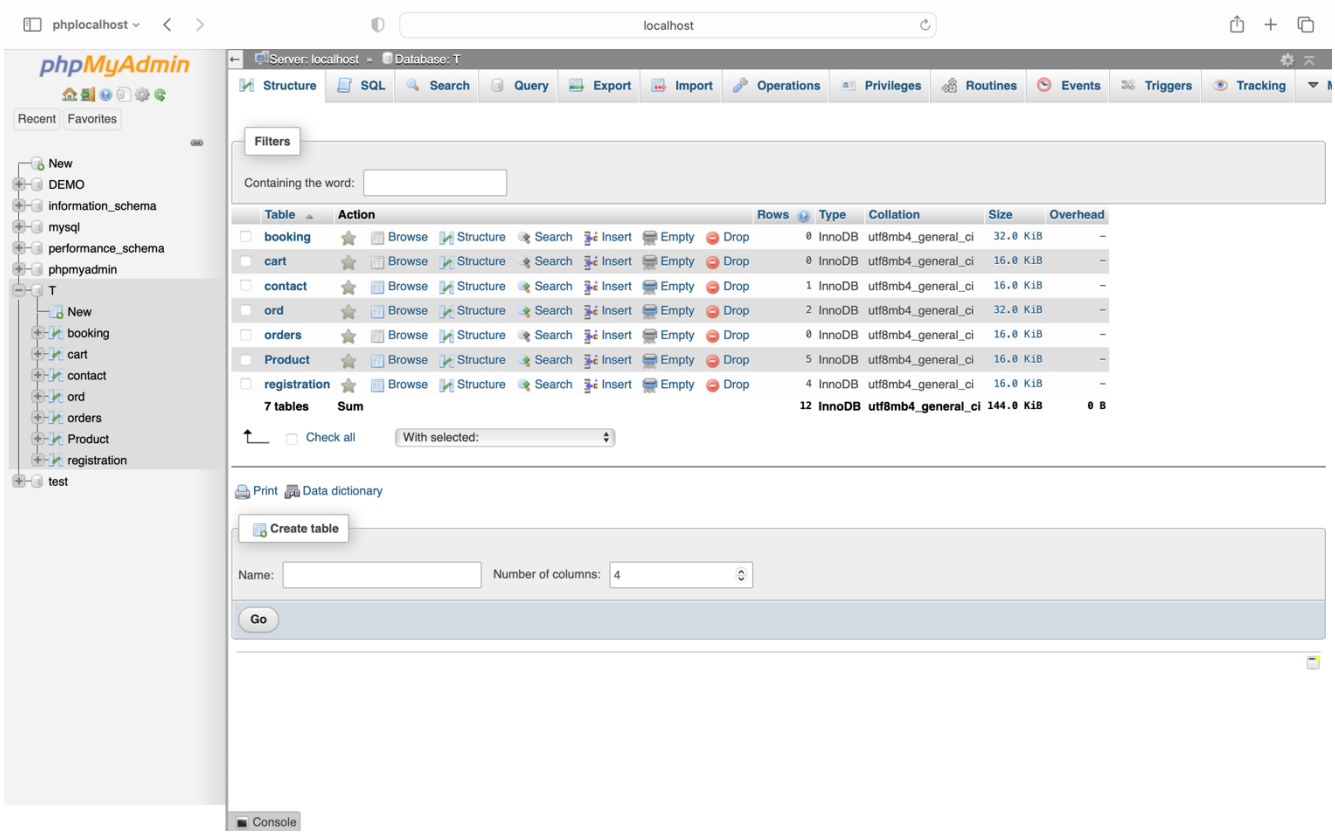
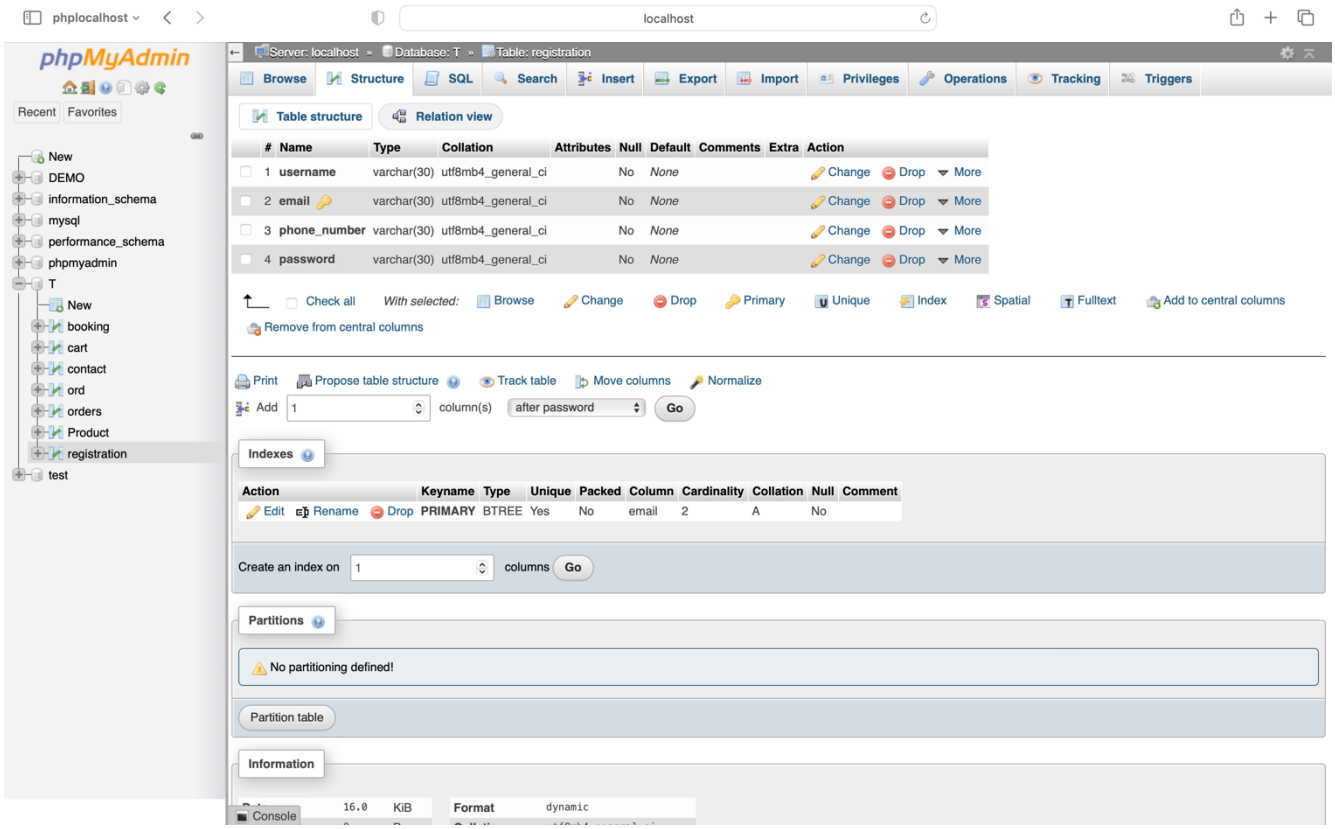


Figure represent the created database and tables inside it.

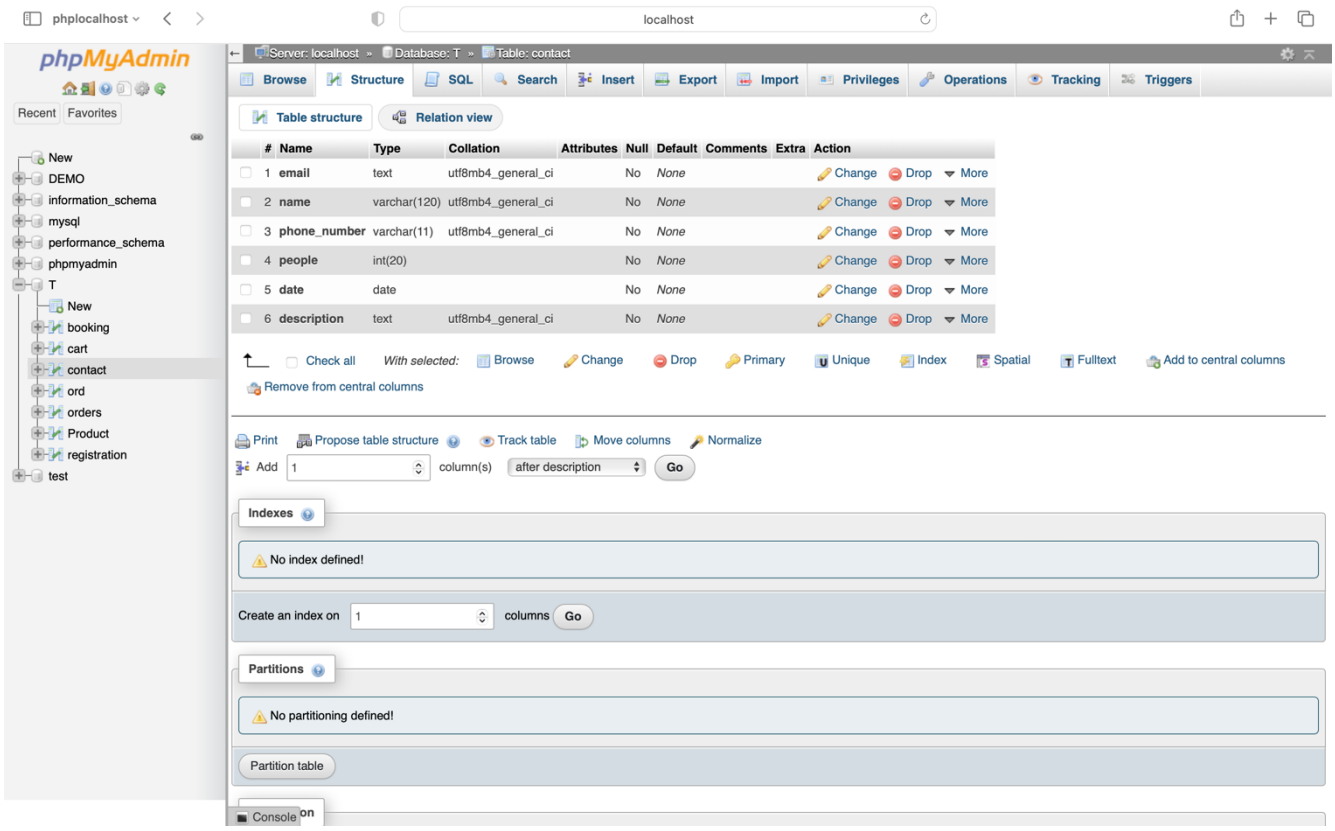
RAMAIAH UNIVERSITY OF APPLIED SCIENCES

Schema of the tables:

Registration:



Orders:



RAMAIAH UNIVERSITY OF APPLIED SCIENCES

Contact: