Hackathon Submission Template (Level-1-Solution)

Use Case Title: Online food ordering system

Student Name: Nishandhini.B

Register Number: C 2 S 2 7 5 O 2

Institution: Theni Kammavar Sangam College Of Arts &

Science

Department: Bachalor Of Computer Application

Date of Submission: 20-03-2025

1. Problem Statement

Limited accessibility - Customers must call or visit restaurants, leading to miscommunication and errors in order details.Inefficient order management - Restaurants struggle with handling high volumes of phone orders, resulting in delays and mistakes.Payment issues - Many restaurants rely on cash payments, which can be inconvenient for customers preferring digital transactions.Lack of real-time tracking - Customers often have no visibility into order status, leading to frustration, Limited restaurant options - Customers may not have easy access to various restaurants in one platform.

2. Proposed Solution

Providing a web and mobile-based platform where customers can browse menus, customize orders, and place them with ease.Offering multiple payment options, including online payments and cash on delivery.Enabling restaurants to efficiently manage incoming orders and reduce errors.Implementing real-time order tracking for better transparency and customer satisfaction.Offering a centralized platform where users can explore multiple restaurants and their offerings

3.Technologies & Tools Considered

The development of an online food ordering system requires a combination of frontend, backend, database, security, and cloud technologies to ensure a seamless user experience. For the frontend, technologies like React.js, Angular, or Vue.js can be used to create an interactive and responsive UI, supported by HTML, CSS, JavaScript, and frameworks like Bootstrap or Tailwind CSS. The backend can be powered by Node.js with Express.js, Django, Flask, or Spring Rest, handling business logic and API interactions. Data storage solutions include MySQL or PostgreSQL for structured data and MongoDB or Firebase for flexible, real-time storage. Secure payment integration is achieved through Stripe, PayPal, or Razorpay, while authentication is managed using JWT or OAuth with encrypted passwords. Cloud services such as AWS, Google Cloud, or Azure offer scalability, and real-time features like Socket.io and Firebase Cloud Messaging enable instant order updates. For mobile apps, cross-platform frameworks like React Native or Flutter provide a smooth user experience. DevOps tools like Docker, Kubernetes, Jenkins, and GitHub Actions streamline deployment and scalability. Together, these technologies ensure a fast, secure, and scalable food ordering system that enhances user satisfaction and operational efficiency.

4. Database Schema & Data Flow

The database schema for an online food ordering system is designed to efficiently store and manage data related to users, restaurants, menus, orders, and payments. Key tables include Users (storing customer details, authentication credentials), Restaurants (storing restaurant information, location, and contact details), Menu Items (linked to restaurants with details like name, price, and availability), Orders (tracking order details, status, and timestamps), Order_Items (linking orders with menu items and quantities), Payments (storing transaction details, payment methods, and status), and Delivery (managing delivery personnel and tracking order status

5. Feasibility & Challenges

- **Feasibility:**

The feasibility of an online food ordering system is high, considering its technical, economic, and operational viability. modern technologies like React.js, Node.js, Django, Firebase, and cloud platforms (AWS, Google Cloud), ensuring scalability, security, and real-time order management. Economically, it offers a profitable business Technically, the system can be developed using model through restaurant commissions, delivery charges, subscriptions, and advertisements, making it a sustainable investment. Operationally, it enhances efficiency by reducing manual errors, optimizing deliveries, and improving customer experience with real-time tracking and notifications.

●Challenges:

However, there are challenges such as high competition from established platforms, ensuring secure transactions, managing peak-hour traffic, and coordinating restaurants and delivery partners efficiently. These can be addressed using robust security protocols, AI-driven order management, efficient load balancing, and strategic customer engagement. With the right approach, the system is both feasible and scalable, providing convenience to users and growth opportunities for businesses.

6. Expected Outcome & Impact

The expected outcome of an online food ordering system is a seamless, efficient, and user-friendly platform that enhances the food ordering experience for customers while streamlining operations for restaurants and delivery partners. Customers will benefit from convenient access to multiple restaurants, real-time order tracking, secure online payments, and personalized recommendations, leading to increased satisfaction and retention. Restaurants can expand their customer base, reduce order processing errors, and optimize kitchen workflows, resulting in higher revenue and improved efficiency. Delivery partners will experience better route optimization and order management, leading to faster deliveries and increased earnings. The overall impact includes growth in the food service industry, increased employment opportunities, and digital transformation in the restaurant sector. Additionally, with data-driven insights, businesses can improve customer engagement, enhance service quality, and optimize pricing strategies, ensuring long-term success in the competitive food delivery market

7.Future Enhancements

Future enhancements for an online food ordering system can focus on AI-driven personalization, automation, and advanced delivery options to improve user experience and efficiency. AI and machine learning can be used to offer personalized recommendations based on user preferences and order history. Voice-assisted ordering using AI-powered chatbots and virtual assistants like Alexa or Google Assistant can make the process more convenient. Augmented Reality (AR) can be integrated to provide virtual menu previews, allowing customers to see dishes before ordering. For delivery, advancements like drone and robotic deliveries can reduce delivery time and operational costs. Blockchain technology can enhance security and transparency in transactions, while IoT (Internet of Things) can improve real-time order tracking and kitchen automation. Additionally, sustainability-focused features, such as eco-friendly packaging choices and carbon footprint tracking, can appeal to environmentally conscious consumers. These enhancements will make the system more intelligent, efficient, and customer-centric, ensuring its growth and competitiveness in the evolving digital food industry.

GitHub repository Link:

https://github.com/mku60422sca002/N**M_hackthon.git**

Hackthon 2st Online food ordering system

```
--- Create Table

CREATE DATABASE FoodOrderingDB;
USE FoodOrderingDB;

--- User Table

CREATE TABLE Users (
    user_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    phone VARCHAR(15),
    address TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

--- Restaurants Table

CREATE TABLE Restaurants (
    restaurant_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    location VARCHAR(255),
    contact VARCHAR(20),
    opening_hours VARCHAR(50),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

```
--- Menu Items Table

CREATE TABLE Menu_Items (
  item_id INT AUTO_INCREMENT PRIMARY KEY,
  restaurant_id INT,
  name VARCHAR(100) NOT NULL,
  description TEXT,
  price DECIMAL(10,2) NOT NULL,
  availability BOOLEAN DEFAULT TRUE,
  FOREIGN KEY (restaurant_id) REFERENCES Restaurants(restaurant_id) ON
DELETE CASCADE
);

--- Order Table

CREATE TABLE Orders (
  order_id INT AUTO_INCREMENT PRIMARY KEY,
  user_id INT,
  restaurant_id INT,
  order_status ENUM('Pending', 'Preparing', 'Out for Delivery', 'Delivered',
'Cancelled') DEFAULT 'Pending',
  total_amount DECIMAL(10,2),
  order_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (user_id) REFERENCES Users(user_id) ON DELETE CASCADE,
  FOREIGN KEY (restaurant_id) REFERENCES Restaurants(restaurant_id) ON
DELETE CASCADE
);
--- Order_Items Table

CREATE TABLE Order_Items (
  order_item_id INT AUTO_INCREMENT PRIMARY KEY,
  order_id INT,
```

```
    item_id INT,
    quantity INT NOT NULL,
    price DECIMAL(10,2) NOT NULL,
    FOREIGN KEY (order_id) REFERENCES Orders(order_id) ON DELETE CASCADE,
    FOREIGN KEY (item_id) REFERENCES Menu_Items(item_id) ON
DELETE CASCADE
);

--- Payments Table

CREATE TABLE Payments (
    payment_id INT AUTO_INCREMENT PRIMARY KEY,
    order_id INT,
    payment_method ENUM('Credit Card', 'Debit Card', 'UPI', 'Cash on Delivery',
'Wallet'),
    payment_status ENUM('Pending', 'Completed', 'Failed') DEFAULT 'Pending',
    transaction_id VARCHAR(255),
    amount DECIMAL(10,2),
    payment_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (order_id) REFERENCES Orders(order_id) ON DELETE CASCADE
);

--- Delivery Table

CREATE TABLE Delivery (
    delivery_id INT AUTO_INCREMENT PRIMARY KEY,
    order_id INT,
    delivery_status ENUM('Pending', 'On the Way', 'Delivered') DEFAULT 'Pending',
    delivery_agent VARCHAR(100),
    estimated_time TIME,
    FOREIGN KEY (order_id) REFERENCES Orders(order_id) ON DELETE CASCADE
);
```

--- Insert Users

```
INSERT INTO Users (name, email, password_hash, phone, address)
VALUES ('John Doe', 'john@example.com', 'hashedpassword123', '9876543210',
'123 Street, City');
```

--- Insert Restaurants

```
INSERT INTO Restaurants (name, location, contact, opening_hours)
VALUES ('Pizza Palace', 'Downtown', '9876543210', '10:00 AM - 11:00 PM');
```

--- Insert Menu Items

```
INSERT INTO Menu_Items (restaurant_id, name, description, price, availability)
VALUES (1, 'Margherita Pizza', 'Classic cheese pizza', 8.99, TRUE);
```

--- Insert Orders

```
INSERT INTO Orders (user_id, restaurant_id, order_status, total_amount)
VALUES (1, 1, 'Pending', 20.50);
```

--- Insert Order items

```
INSERT INTO Order_Items (order_id, item_id, quantity, price)
VALUES (1, 1, 2, 17.98);
```

--- Insert Payment Details

```
INSERT INTO Payments (order_id, payment_method, payment_status,
```

```
transaction_id, amount)
VALUES (1, 'UPI', 'Completed', 'TXN12345UPI', 20.50);

--- Insert Delivery Details

INSERT INTO Delivery (order_id, delivery_status, delivery_agent, estimated_time)
VALUES (1, 'On the Way', 'David', '00:30:00');

--- Retrieve All Users

SELECT * FROM Users;

--- Get All Menu Items For a Specific Restaurant

SELECT name, description, price FROM Menu_Items WHERE restaurant_id = 1;


--- Retrieve All Orders With User Details
SELECT Orders.order_id, Users.name, Users.email, Orders.order_status,
Orders.total_amount, Orders.order_time
FROM Orders
JOIN Users ON Orders.user_id = Users.user_id;


--- Check Order Stats and Payment Details

SELECT Orders.order_id, Orders.order_status, Payments.payment_status,
Payments.payment_method
FROM Orders
JOIN Payments ON Orders.order_id = Payments.order_id
WHERE Orders.order_id = 1;
```

- - - Find All Pending Deliveries

```
SELECT Delivery.order_id, Users.name AS Customer, Delivery.delivery_status,
Delivery.delivery_agent
FROM Delivery
JOIN Orders ON Delivery.order_id = Orders.order_id
JOIN Users ON Orders.user_id = Users.user_id
WHERE Delivery.delivery_status = 'Pending';
```