

CHAPTER - 1.

INTRODUCTION TO SYSTEM :

The **E-Waiter system** is a comprehensive digital solution designed to modernize and streamline the order management process for hotels and restaurants. This application empowers hotel owners to manage their business operations more efficiently by replacing traditional pen-and-paper methods with a digital platform accessible through smartphones, tablets, and other devices. It aims to enhance productivity, reduce human errors, and improve overall customer service, making it an indispensable tool for modern hospitality businesses.

By digitizing critical processes such as order-taking, billing, and staff management, **E-Waiter** helps establishments maintain smooth and accurate operations, even during peak hours. The platform is designed with a user-friendly interface that simplifies tasks like managing menus, tracking orders, and monitoring staff performance, making it accessible to both tech-savvy users and those with minimal digital experience.

The system offers dedicated functionalities for both hotel owners and waiters, ensuring seamless communication and coordination. Hotel owners can register their establishments, add and update menu items, introduce special offers, view detailed billing records, and monitor the performance of their waitstaff. Waiters, on the other hand, can register using the owner's unique ID, take customer orders digitally, and send them directly to the kitchen, ensuring faster and more accurate service.

Key benefits of **E-Waiter** include:

- **Enhanced Operational Efficiency:** Automates routine tasks such as order management, billing, and reporting, allowing staff to focus on delivering exceptional service.
- **Error Reduction:** Minimizes human errors in order-taking and billing, ensuring accurate and consistent service.
- **Real-Time Tracking:** Provides instant updates on order status, helping both kitchen staff and waiters stay synchronized.
- **Faster Service:** Reduces the time required to take and process orders, leading to quicker service and improved customer satisfaction.
- **User-Friendly Interface:** Simple and intuitive design ensures ease of use for both owners and waitstaff, with minimal training required.
- **Data Security:** Ensures that sensitive business information is securely stored and accessible only to authorized users.
- **Scalability:** Designed to accommodate businesses of all sizes, from small cafes to large restaurants, with the flexibility to scale as the business grows.

By automating essential tasks and improving communication within the establishment, **E-Waiter** helps hotels and restaurants deliver faster, more accurate, and more efficient service. This not only enhances the dining experience for customers but also boosts the overall profitability and reputation of the business.

CHAPTER - 2.

INTRODUCTION TO PROJECT :

2.1 ABOUT PROJECT :

The **E-Waiter** project is developed to provide a comprehensive digital platform for hotel owners and waiters to manage daily business operations with ease. The primary goal is to eliminate traditional pen-and-paper methods, replacing them with a streamlined, smartphone-based system that improves efficiency, accuracy, and communication within the establishment. By digitizing the order-taking and billing processes, the system enhances operational workflows, reduces human errors, and ensures faster service, ultimately improving the overall dining experience.

The system is designed with two main user roles:

- **Hotel Owner:**
 - Registers on the platform using a secure account.
 - Manages subscription payments to access the system's features.
 - Adds and updates menu items, including categories, prices, and availability.
 - Creates and manages special offers and discounts.
 - Views detailed billing records and generates reports for financial analysis.
- **Waiter:**
 - Registers using the unique `owner_id` provided by the hotel owner.
 - Takes customer orders digitally using a smartphone or tablet, selecting tables, menu items, and quantities.

- Sends orders directly to the kitchen, reducing communication delays and minimizing order errors.
- Ensures that the billing process is completed accurately and the final bill is sent to the counter for customer checkout.

This project simplifies communication between waiters and the kitchen staff, ensuring that orders are received instantly and accurately, which reduces errors caused by miscommunication. The automated billing system eliminates manual calculations, ensuring faster checkouts and reducing waiting times for customers. The platform's real-time tracking feature allows both waiters and kitchen staff to monitor the status of each order, improving coordination and efficiency.

2.2 OBJECTIVE OF SYSTEM :-

The primary objective of the **E-Waiter** system is to modernize and optimize the order management process in hotels and restaurants by providing a digital platform for owners and waiters. By automating essential tasks such as order-taking, billing, and communication, the system aims to improve efficiency, reduce errors, and enhance customer service.

The system consists of many objectives as follows:

1. **Digitize Order Management:** Replace traditional pen-and-paper methods with a digital platform that allows waiters to take orders using smartphones or tablets, ensuring faster and more accurate order processing.
2. **Automate Billing Process:** Generate bills automatically after order completion and transfer them directly to the billing counter, reducing manual calculations and speeding up the checkout process.
3. **Reduce Manual Errors:** Minimize human errors in order-taking and billing by providing a structured digital workflow, ensuring that orders are recorded accurately and invoices are calculated correctly.
4. **Eliminate Pen and Paper Usage:** Ensure that waiters no longer rely on pen and paper, promoting a faster, cleaner, and more efficient order-taking process that reduces clutter and paper waste.
5. **Enhance Communication:** Improve communication between waiters, kitchen staff, and the billing counter by providing real-time order updates and notifications, ensuring that everyone is informed of the order status.
6. **Improve Customer Service:** Speed up order-taking and billing processes, reducing wait times and enhancing the overall dining experience for customers.
7. **Simplify Menu Management:** Allow hotel owners to easily add, update, and remove menu items, including prices and special

offers, ensuring that waiters always have access to the latest menu information.

8. **Provide Role-Based Access:** Ensure that each user has access only to the features relevant to their role, enhancing system security and preventing unauthorized access.
9. **Enable Real-Time Order Tracking:** Allow waiters and kitchen staff to track the status of each order in real time, ensuring that food is prepared and delivered promptly.
10. **Monitor Waiter Performance:** Provide hotel owners with tools to monitor waiter performance, including the number of orders taken and the time required to complete each order.
11. **Store Business Data Securely:** Store order details, billing records, and other business data in a secure cloud-based database, ensuring that information is protected and accessible from any device with an internet connection.
12. **Generate Reports for Business Analysis:** Allow hotel owners to generate detailed reports on sales, revenue, and order trends, helping them make informed decisions to improve their business operations.
13. **Support Multiple Devices:** Ensure compatibility with a wide range of devices, including smartphones, tablets, and desktop computers, allowing users to access the system from anywhere.
14. **Increase Operational Efficiency:** Streamline the entire order management process, allowing hotels to serve more customers in less time without compromising service quality.
15. **Facilitate Easy Onboarding:** Simplify the registration process for both owners and waiters, ensuring that new users can start using the system with minimal training.
16. **Offer Scalability:** Design the system to accommodate businesses of all sizes, from small cafes to large restaurants, with the flexibility to scale as the business grows.
17. **Minimize Operational Costs:** Reduce costs associated with printing order slips, managing paper records, and correcting manual errors, improving overall profitability.

18. **Ensure Data Privacy and Security:** Protect sensitive business information through secure authentication methods, encrypted data storage, and role-based access controls.

By achieving these objectives, **E-Waiter** aims to help hotels and restaurants improve their operational efficiency, reduce costs, and deliver exceptional service to their customers.

2.3 SCOPE OF SYSTEM :

The **E-Waiter** system is designed to digitalize and optimize order management in hotels and restaurants. Its scope covers various functionalities and user roles to ensure smooth and efficient operations. The key aspects of the system's scope include:

❖ **Owner Management:**

Hotel owners can register on the platform and manage their subscription by making secure online payments.

Owners can add, update, and delete menu items and offers as needed.

Owners have the ability to register and manage waiters by providing them with an `owner_id`.

❖ **Waiter Management:**

Waiters can register using the provided `owner_id`, ensuring a secure and structured registration process.

Waiters can take customer orders digitally using smartphones, selecting tables, menu items, and quantities.

❖ **Order and Billing Process:**

Orders are instantly communicated to the kitchen, reducing delays and miscommunication.

Bills are generated automatically after order completion and transferred to the billing counter for faster payment processing.

❖ **System Accessibility:**

The system is accessible through smartphones, making it portable and convenient for waiters.

Both owners and waiters can use the system with minimal technical knowledge, ensuring ease of use.

❖ **Business Benefits:**

Reduces manual errors and enhances operational efficiency.

Speeds up the order-taking and billing process, improving customer service.

Eliminates the need for pen and paper, promoting a modern and eco-friendly approach to order management.

2.4 USER REQUIREMENT :-

The **E-Waiter** system has two main user roles: **Hotel Owner** and **Waiter**, each with specific requirements to ensure efficient order management.

1. Hotel Owner Requirements:

- Ability to register on the platform using email and password.
- Secure online payment system to manage subscriptions.
- Option to add, update, and delete menu items and offers.
- Functionality to register and manage waiters using a unique `owner_id`.
- Access to view order details and bills generated by waiters.

2. Waiter Requirements:

- Ability to register using the unique `owner_id` provided by the hotel owner.
- Simple interface to select tables, menu items, and quantities while taking orders.
- Instant order placement, ensuring quick communication with the kitchen.
- Automated bill generation upon order completion, with bills sent directly to the billing counter.

CHAPTER – 3.

INVESTIGATION PHASE :

3.1 EXISTING SYSTEM :

Several digital systems similar to **E-Waiter** are already available in the market, offering solutions to help waiters take orders and manage billing processes. These systems aim to improve efficiency, reduce errors, and provide better customer service. Some common features of existing systems include:

- **Digital Order Taking:** Allowing waiters to use mobile devices to take orders.
- **Real-Time Communication:** Instant order transmission to the kitchen and billing counter.
- **Automated Billing:** Generating bills automatically to reduce calculation errors.
- **Menu Management:** Enabling restaurant owners to update menus and prices.

However, despite these features, most existing systems have certain drawbacks:

- **Complex Registration Process:** Many systems have complicated registration processes, making it difficult for waiters to join the system.
- **Limited Owner-Waiter Connection:** Some systems do not provide a unique way for owners to manage and identify their waiters.
- **Lack of Subscription Management:** Few systems offer built-in subscription management for owners.
- **Dependence on Expensive Hardware:** Some systems require specialized devices, increasing costs for hotels.

WHAT'S NEW IN OUR SYSTEM?

The **E-Waiter** system introduces several unique features that differentiate it from existing solutions:

- **Simple Waiter Registration Using Owner_ID:** Waiters can easily register by using the unique `owner_id` provided by the hotel owner, ensuring a secure and quick registration process.
- **Integrated Subscription Payment System:** Hotel owners can manage their subscriptions directly within the system using secure online payments, ensuring uninterrupted access.
- **Smartphone Compatibility:** The system is designed to work on smartphones, eliminating the need for expensive hardware.
- **Instant Bill Transfer to Counter:** After order completion, the bill is automatically transferred to the billing counter, reducing wait time and manual errors.
- **User-Friendly Interface:** The system is designed with a simple and intuitive interface that requires minimal training for both owners and waiters.

3.2 PROPOSED SYSTEM:

The **E-Waiter** system is a comprehensive digital platform designed to simplify hotel order management and improve efficiency. By replacing traditional order-taking and billing processes with a smartphone-based solution, the system enhances operational workflows, reduces manual errors, and improves communication between waiters, kitchen staff, and the billing counter.

Hotel owners can register their businesses on the platform, manage subscriptions, and have full control over menus and special offers. Waiters register using the owner's unique `owner_id`, ensuring a secure connection between the staff and the establishment. Orders are taken digitally using smartphones or tablets, instantly transmitted to the kitchen and counter, and automatically billed upon order completion. The final bill is then sent to the counter for quick payment processing, reducing waiting times and improving customer satisfaction.

The system offers real-time updates, a user-friendly interface, and secure data management, ensuring that both owners and staff can operate the platform with ease. By digitizing key processes, **E-Waiter** eliminates the need for pen-and-paper methods, minimizes human errors, and enhances the overall efficiency of hotel operations.

The proposed system includes the following key features and benefits:

1. **Digital Order-Taking:** Waiters can take orders using smartphones, selecting menu items, quantities, and table numbers with just a few taps. Orders are instantly sent to the kitchen, eliminating communication delays and reducing order errors.
2. **Automated Billing Process:** Bills are generated automatically after the order is completed, with accurate calculations based on the selected items and quantities. The system also applies discounts and special offers where applicable.
3. **Real-Time Order Tracking:** Both waiters and kitchen staff receive real-time updates on the status of each order, ensuring

that food is prepared and delivered promptly. This feature reduces delays and enhances workflow coordination.

4. **Instant Bill Transfer:** Once the order is completed, the system sends the final bill to the billing counter for quick payment processing, reducing waiting times for customers and improving service efficiency.
5. **Secure User Registration:** Hotel owners and waiters register using secure login credentials. Waiters use the unique `owner_id` provided by the hotel owner to connect their accounts, ensuring that only authorized users can access the system.
6. **Menu Management:** Hotel owners can easily add, update, and remove menu items, including categories, prices, and availability. They can also create and manage special offers, ensuring that waiters always have access to the latest menu information.
7. **Role-Based Access Control:** The system provides different access levels for hotel owners and waiters. Owners have full control over the system, while waiters can only access features related to order-taking and billing.
8. **User-Friendly Interface:** The platform is designed with a simple and intuitive interface, ensuring that both owners and waiters can navigate the system with minimal training. Clear menus, easy-to-use buttons, and real-time notifications enhance the user experience.
9. **Real-Time Notifications:** The system sends instant notifications to waiters when an order is ready for delivery, ensuring that food is served promptly and customers receive their meals at the right temperature.
10. **Waiter Performance Monitoring:** Hotel owners can track waiter performance by viewing the number of orders taken, order completion times, and customer feedback, helping them identify top-performing staff and areas for improvement.
11. **Billing and Reporting:** The system automatically records all orders and payments, allowing hotel owners to generate detailed reports on sales, revenue, and order trends. These reports

help owners analyze business performance and make data-driven decisions.

12. **Secure Data Storage:** All business data, including order details, billing records, and user information, is securely stored in a cloud-based database. The system uses encryption and secure authentication methods to protect sensitive information.
13. **Multi-Device Compatibility:** The platform is compatible with smartphones, tablets, and desktop computers, allowing users to access the system from any device with an internet connection.
14. **Improved Customer Satisfaction:** By speeding up the order-taking and billing processes, reducing errors, and ensuring timely food delivery, the system enhances the overall dining experience, leading to higher customer satisfaction and repeat business.
15. **Cost Savings:** By eliminating the need for printed order slips and reducing manual errors, the system helps hotels save on operational costs and improve profitability.
16. **Eco-Friendly Solution:** By replacing paper-based processes with a digital platform, the system reduces paper usage, supporting environmentally friendly business practices.

In summary, the **E-Waiter** system offers a modern and efficient solution for hotel order management, streamlining operations, reducing costs, and enhancing customer service. Its user-friendly design, real-time updates, and secure data management make it an essential tool for hotels and restaurants looking to improve their efficiency and competitiveness in today's digital age.

CHAPTER – 4.

REQUIREMENT ANALYSIS :

4.1 HARDWARE AND SOFTWARE REQUIREMENT :-

Hardware requirement :

❖ **For Hotel Owners:**

- Device: Desktop, laptop, tablet, or smartphone for managing the system
- Processor : Intel Core i3 or higher
- RAM : 4 GB
- Storage: Minimum : 100 GB of free space
- Monitor : 15 Colour Monitor

❖ **For Waiters:**

- Device: Smartphone, Laptop or tablet for taking orders
- Operating System: Android or iOS
- RAM: Minimum 2 GB
- Smart Printer: A thermal or smart printer connected to the billing counter for printing bills instantly upon order completion

Software requirement :

1. Operating System : Window 11

2. Developing Tool : VScode, PyCharm

3. Development Environment:

- **Programming Language:** Python
- **Framework:** Flask (for backend development)

- **Frontend:** HTML, CSS, JavaScript

4. Database:

- **Database System:** MySQL
- **Database Connector:**

mysql-connector-python or SQLAlchemy (for MySQL integration)

5. Libraries & Dependencies:

- **Backend:**
 - Flask (for building web applications)
 - Flask-SQLAlchemy (for database interaction)
 - Flask-Login (for user authentication)
 - Flask-WTF (for form handling)

6. Additional Tools:

- **Web Browser:** Google Chrome or Firefox (for testing UI)

4.2 TECHNOLOGY USED :-

The E-Waiter is developed using Python, a general-purpose, high-level programming language known for its readability and efficiency. Python supports multiple programming paradigms, including object-oriented, procedural, and functional programming, making it suitable for building robust applications.

Python provides extensive libraries and frameworks for web development, such as Django and Flask, which are utilized in this project for backend development. The application interacts with a database using MySQL to manage user details, bus pass records, and transactions efficiently.

Need for Python Programming:-

Python is widely chosen for web and software development due to its simplicity, flexibility, and extensive library support. **Key reasons for using Python in this project include:**

Rapid Development: Python enables quick prototyping and development with fewer lines of code.

Extensive Libraries: It provides numerous built-in libraries for database management, web development, and security.

Cross-Platform Compatibility: Python applications can run on various operating systems with minimal modifications.

platforms, ensuring reliability and scalability.

Flask Framework :

Flask is a lightweight and flexible web framework for Python. Known for its simplicity and scalability, Flask is widely used for building web applications, APIs, and microservices. It follows the **WSGI (Web**

Server Gateway Interface) standard and is built on the **Werkzeug** toolkit and **Jinja2** template engine.

KEY FEATURES OF FLASK

1. **Lightweight and Minimalistic:**

- Flask is designed to be simple and lightweight, with no unnecessary features or components.
- Developers can add extensions based on project requirements.

2. **Flexible and Scalable:**

- Flask provides flexibility in structuring the application, allowing developers to create both small projects and large, complex systems.

3. **Built-in Development Server:**

- It comes with a built-in development server that simplifies testing and debugging during development.

4. **Routing System:**

- Flask allows developers to define URL routes using decorators, making it easy to handle different pages and functionalities.

5. **Template Engine (Jinja2):**

- The Jinja2 template engine allows developers to create dynamic HTML pages using templates with placeholders and logic.

6. **Request and Response Handling:**

- Flask makes it easy to handle HTTP requests, form data, and query parameters using the `request` object.

7. **Session and Cookies:**

- Built-in support for managing user sessions and cookies for maintaining user data across requests.

8. **RESTful API Support:**

- Flask is commonly used to build RESTful APIs, allowing seamless communication between different systems.

9. **Database Integration:**

- Flask supports integration with databases like MySQL, SQLite, PostgreSQL, and MongoDB using libraries such as SQLAlchemy and PyMySQL.

Object-Oriented Concepts in Python:-

Abstraction: The system abstracts complex user interactions, allowing passengers to apply for and renew passes seamlessly.

Encapsulation: User data is securely stored and accessed only through defined methods.

Inheritance: Different user roles (e.g., students, senior citizens, general public) inherit common functionalities while having specific features.

Polymorphism: The system supports multiple payment methods and bus pass types dynamically.

CHAPTER-5.

SYSTEM ANALYSIS AND DESIGN :

5.1 FEASIBILITY STUDY :-

1. TECHNICAL FEASIBILITY

- **Technology Stack:** The project uses Flask, MySQL, and HTML/CSS/JavaScript, which are reliable and widely used technologies.
- **Compatibility:** The system can be hosted on local servers or cloud platforms, ensuring flexibility in deployment.
- **Development Skills:** Since the developer is skilled in Python, Flask, and MySQL, there are no major technical barriers.

2. ECONOMIC FEASIBILITY

- **Development Cost:** The cost is minimal since open-source tools like Flask, MySQL, and VS Code are used.
- **Maintenance Cost:** Low, as regular updates and bug fixes can be handled independently.
- **ROI (Return on Investment):** Hotel owners benefit from improved order management, reducing human errors and increasing efficiency, thus leading to better profits.

3. OPERATIONAL FEASIBILITY

- **User-Friendliness:** The system is designed to be intuitive for both hotel owners and waiters, ensuring quick adoption.
- **Workflow Improvement:** Automating order placement and billing reduces manual work, improving overall service speed and accuracy.

- **Scalability:** The system can easily scale to accommodate more users, tables, and menu items as the business grows.

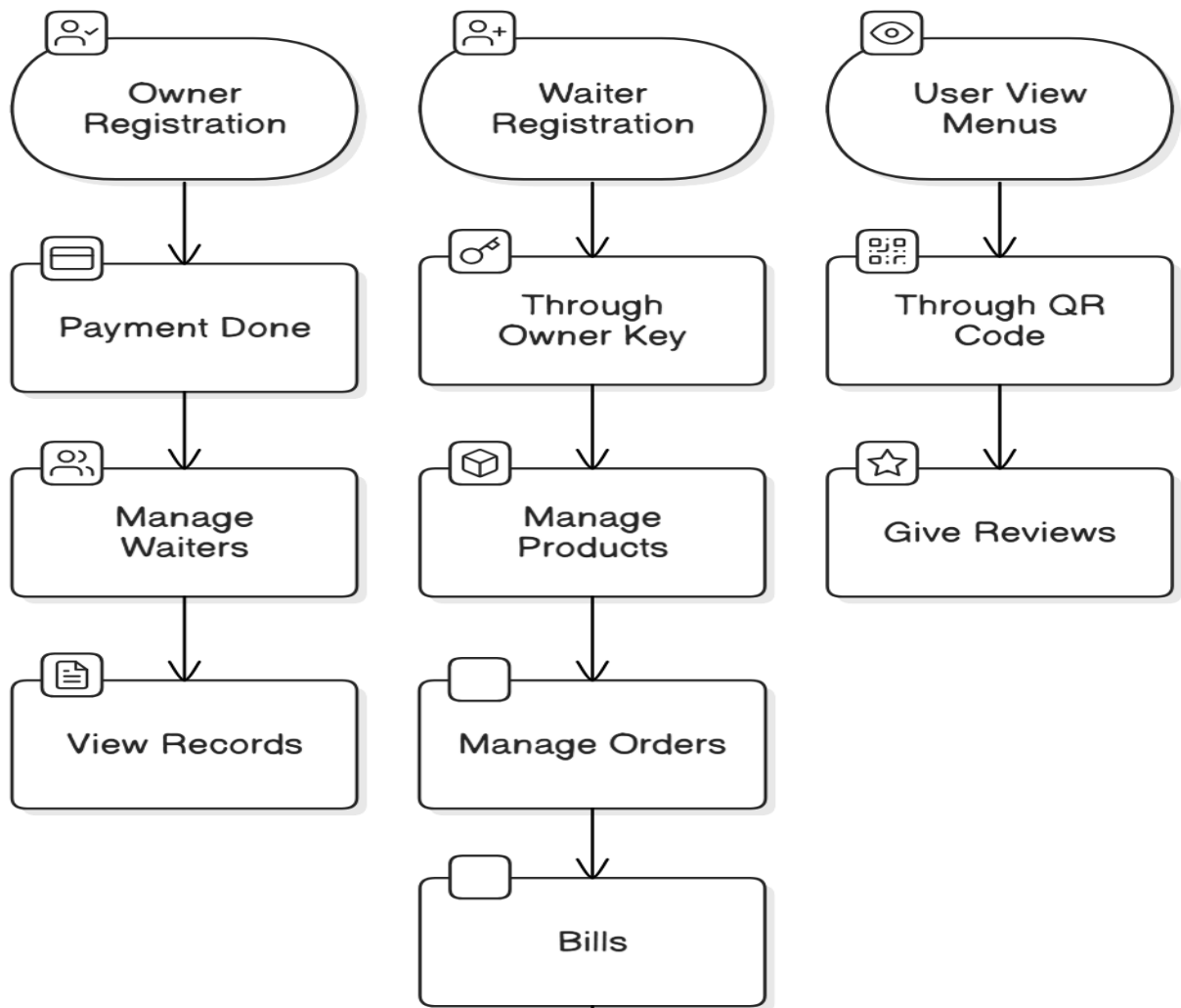
4. SCHEDULE FEASIBILITY

- **Project Timeline:** The development timeline is realistic, considering the scope and complexity of the project.
- **Milestones:** Key milestones include developing core functionalities (owner and waiter registration, order placement, billing) and testing the system before deployment.

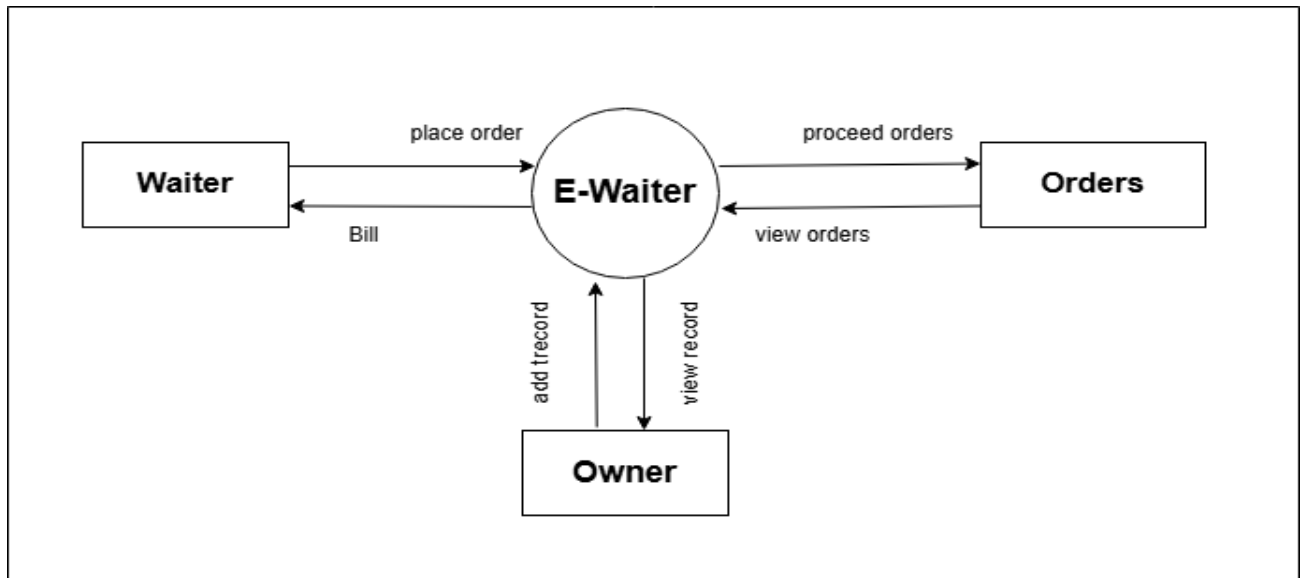
5. LEGAL AND ETHICAL FEASIBILITY

- **Data Privacy:** The system ensures that sensitive information like passwords and payment data is securely stored.
- **Compliance:** The project follows relevant regulations for data storage and user privacy.

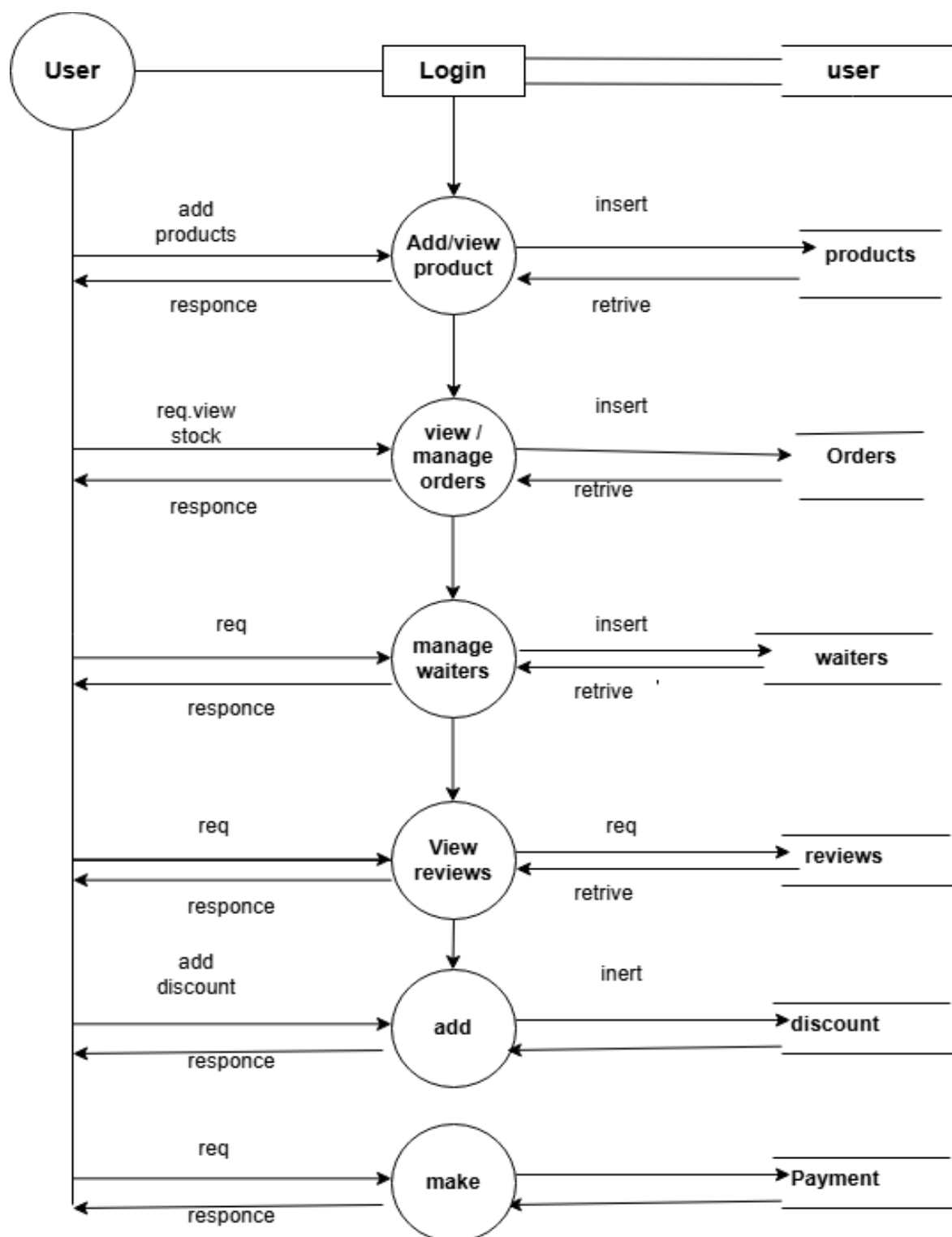
5.2 DATAFLOW DIAGRAM :-



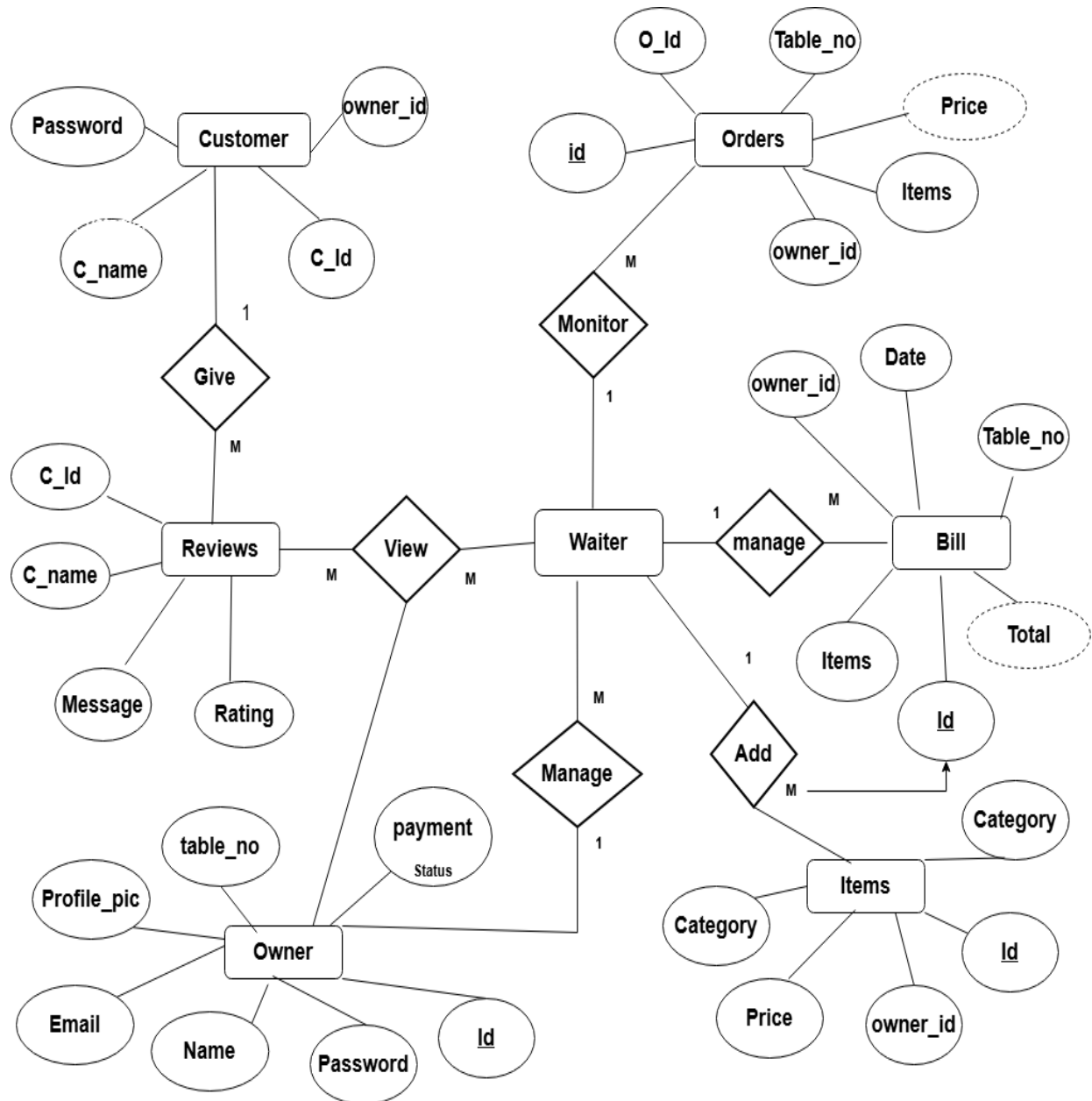
1. ZERO LEVEL DFD :



2. FIRST LEVEL DFD :



5.3 ENTITY RELATIONSHIP DIAGRAM :-



5.4 DATABASE STRUCTURE :-

❖ Owner Table :

| Field | Type | Null | Key | Default | Extra |
|---------------------|----------------------------------|------|-----|---------------------|----------------|
| owner_id | int(11) | NO | PRI | NULL | auto_increment |
| owner_name | varchar(50) | NO | | NULL | |
| email | varchar(100) | NO | UNI | NULL | |
| password | varchar(255) | NO | | NULL | |
| created_at | timestamp | NO | | current_timestamp() | |
| subscription_status | enum('trial','active','expired') | NO | | trial | |

❖ Waiters Table :

| Field | Type | Null | Key | Default | Extra |
|-------------|--------------|------|-----|---------------------|----------------|
| waiter_id | int(11) | NO | PRI | NULL | auto_increment |
| waiter_name | varchar(50) | NO | | NULL | |
| password | varchar(255) | NO | | NULL | |
| owner_id | int(11) | NO | MUL | NULL | |
| joined_at | timestamp | NO | | current_timestamp() | |

❖ Table_session Table:

| Field | Type | Null | Key | Default | Extra |
|------------|-------------------------|------|-----|---|----------------|
| id | int(11) | NO | PRI | NULL | auto_increment |
| table_id | int(11) | NO | | NULL | |
| owner_id | int(11) | NO | | NULL | |
| waiter_id | int(11) | NO | | NULL | |
| status | enum('free','occupied') | NO | | occupied | |
| data | text | YES | | [] | |
| created_at | datetime | YES | | current_timestamp() | |
| updated_at | datetime | YES | | current_timestamp() on update current_timestamp() | |
| total | int(11) | NO | | NULL | |

❖ Bills Table :

| Field | Type | Null | Key | Default | Extra |
|--------------|----------|------|-----|---------------------|----------------|
| bill_id | int(11) | NO | PRI | NULL | auto_increment |
| table_id | int(11) | NO | | NULL | |
| owner_id | int(11) | NO | MUL | NULL | |
| waiter_id | int(11) | NO | MUL | NULL | |
| total_amount | float | NO | | NULL | |
| bill_date | datetime | YES | | current_timestamp() | |
| details | text | YES | | NULL | |

❖ Menus Table :

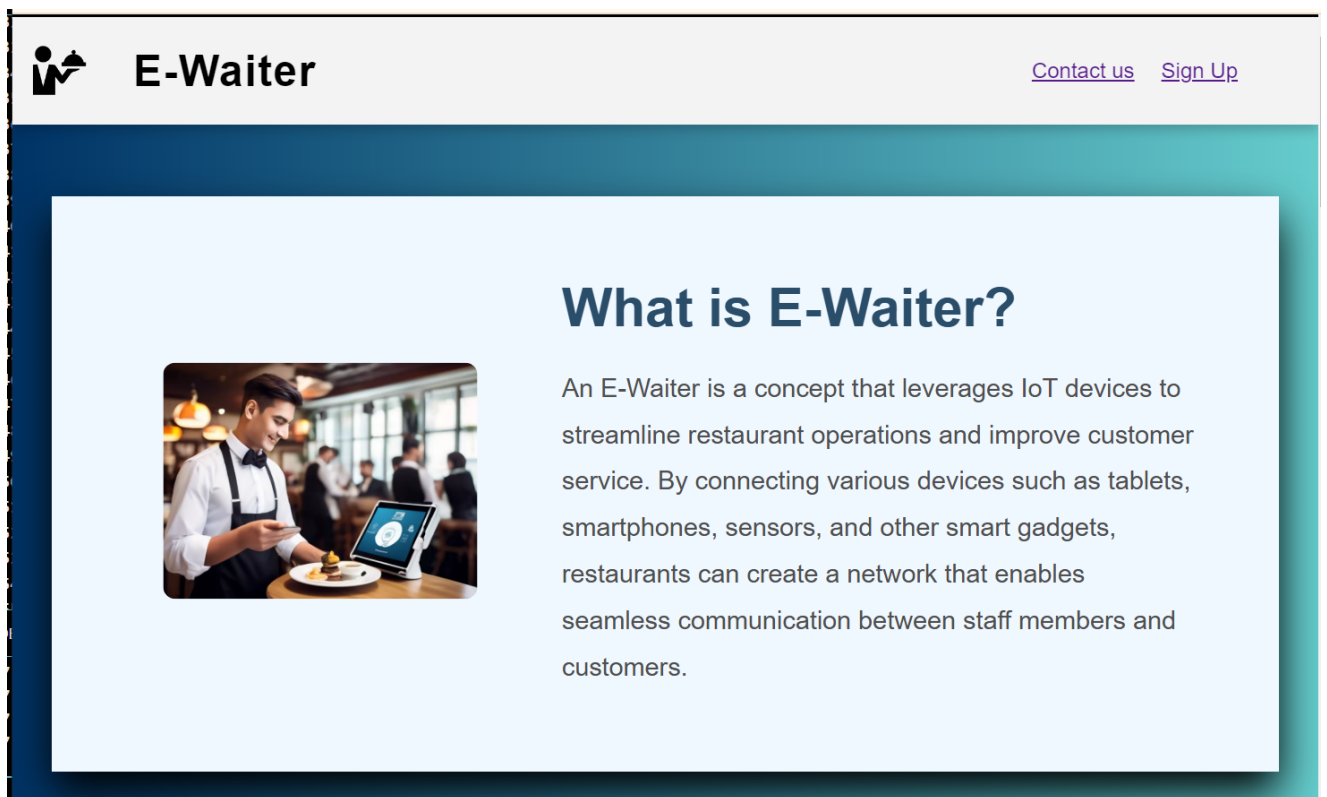
| Field | Type | Null | Key | Default | Extra |
|-----------|---------------|------|-----|---------|----------------|
| item_id | int(11) | NO | PRI | NULL | auto_increment |
| owner_id | int(11) | YES | MUL | NULL | |
| item_name | varchar(50) | NO | | NULL | |
| price | decimal(10,2) | NO | | NULL | |
| category | varchar(50) | YES | | NULL | |

CHAPTER - 6 .


USER INTERFACE SCREENS :

6.1 FORM DESIGN :

➤ Home Page :



➤ Owner Registration form :

 **E-Waiter**

[Contact us](#) [Sign Up](#)

Create Account

Username:

Password:

Email:


Register as:

Owner

Create Account

Already have an account? [Login here](#)

➤ **Waiter Registration :**

 **E-Waiter**

[Contact us](#) [Sign Up](#)

Register as Waiter

Username:

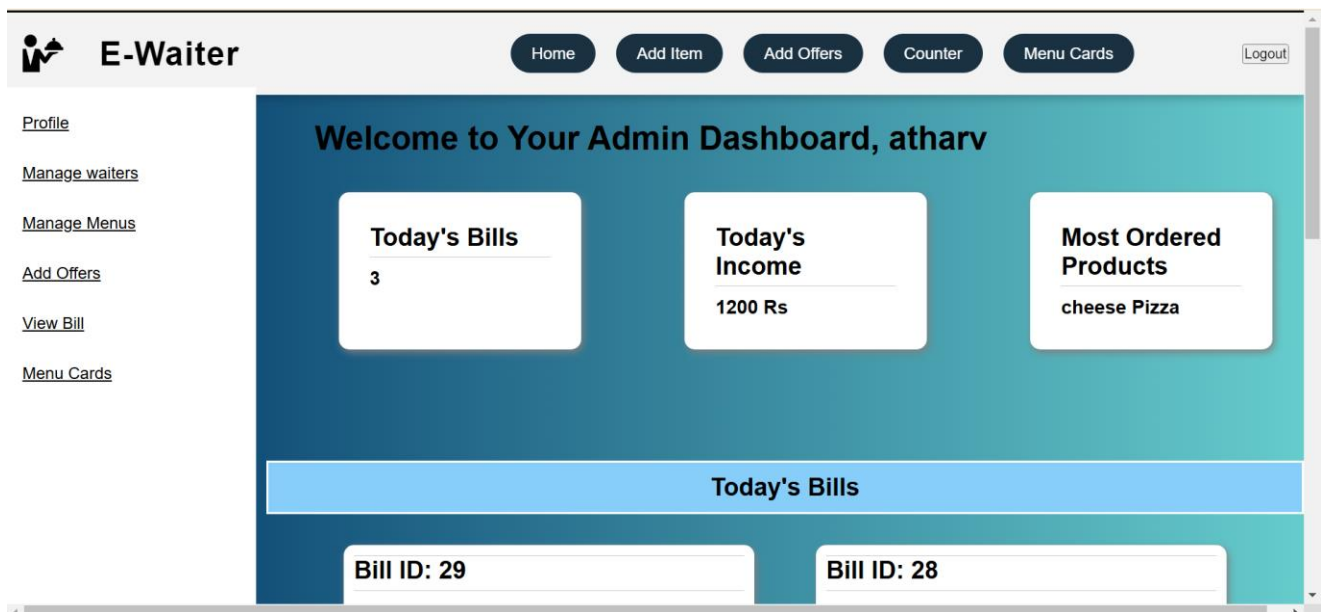
Password:

owner_id:

Create Account

Already have an account? [Login here](#)

➤ **Owner Dash board :**



The screenshot shows the E-Waiter Admin Dashboard. The top navigation bar includes links for Home, Add Item, Add Offers, Counter, Menu Cards, and a Logout button. A sidebar on the left contains links for Profile, Manage waiters, Manage Menus, Add Offers, View Bill, and Menu Cards. The main content area features a welcome message for 'atharv' and three summary cards: 'Today's Bills' (3), 'Today's Income' (1200 Rs), and 'Most Ordered Products' (cheese Pizza). Below these is a section titled 'Today's Bills' showing two bill entries with IDs 29 and 28.

E-Waiter

Home Add Item Add Offers Counter Menu Cards Logout

Profile

Manage waiters

Manage Menus

Add Offers

View Bill

Menu Cards

Welcome to Your Admin Dashboard, atharv

Today's Bills
3

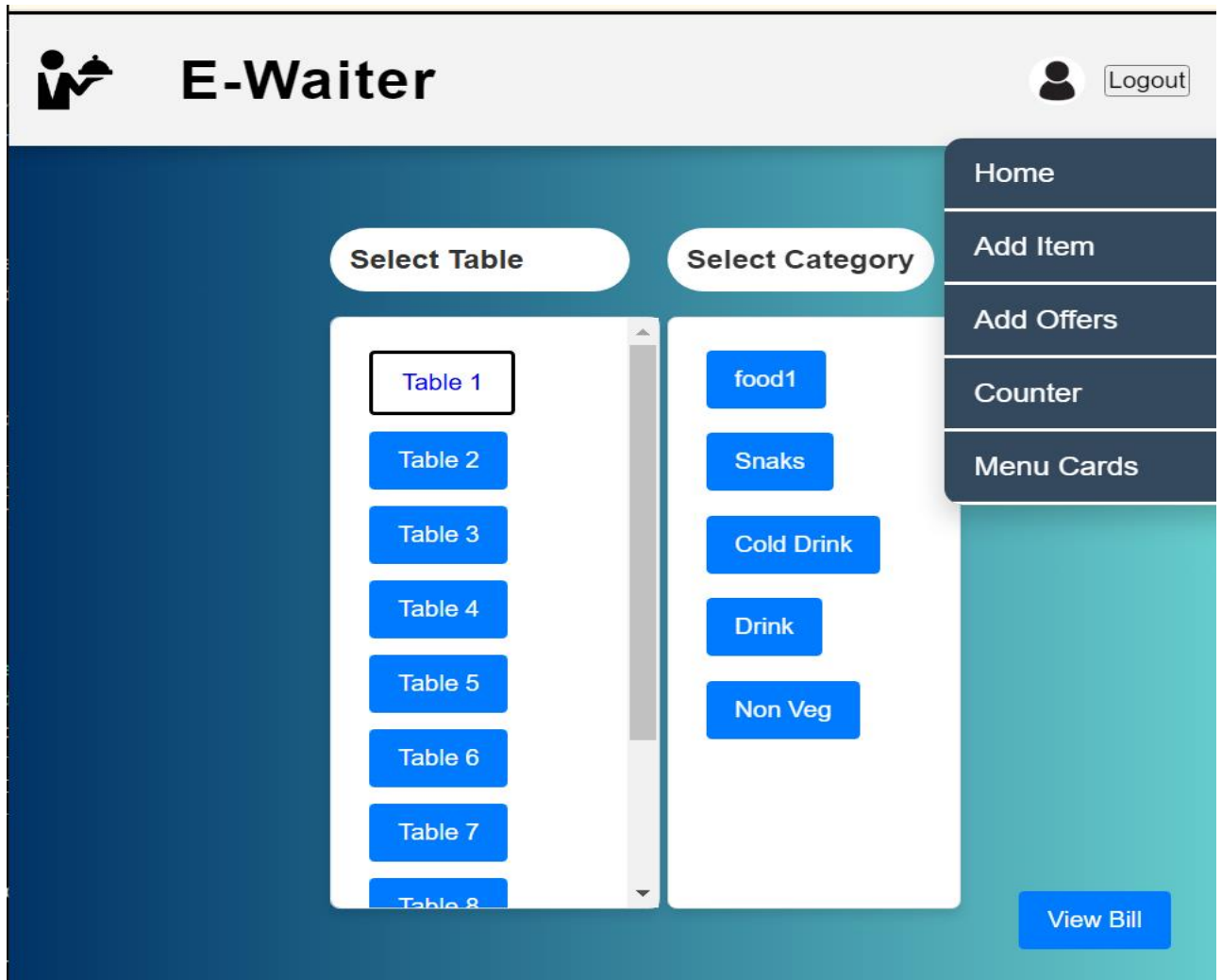
Today's Income
1200 Rs

Most Ordered Products
cheese Pizza

Today's Bills

Bill ID: 29 Bill ID: 28

➤ Waiter Dashboard :



The screenshot shows the E-Waiter Waiter Dashboard. The top navigation bar includes the E-Waiter logo, a user profile icon, and a Logout button. The main content area features two selection panels: 'Select Table' with a list of tables (Table 1 to Table 8) and 'Select Category' with a list of categories (food1, Snaks, Cold Drink, Drink, Non Veg). A sidebar on the right contains links for Home, Add Item, Add Offers, Counter, and Menu Cards. A 'View Bill' button is located at the bottom right.

E-Waiter

Logout

Home

Add Item

Add Offers

Counter

Menu Cards

Select Table

Table 1

Table 2

Table 3

Table 4

Table 5

Table 6

Table 7

Table 8

Select Category

food1

Snaks



Cold Drink

Drink

Non Veg

View Bill

➤ Bill Generation :

 **E-Waiter**  Logout

Bill (Table: Table 1) X

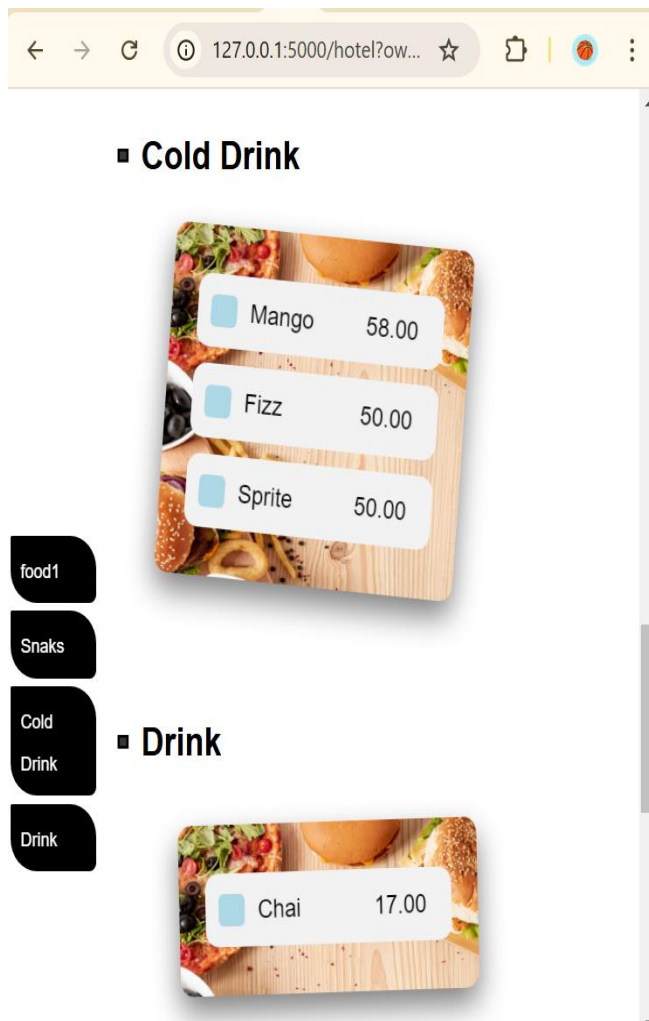
| | | | |
|---------------|-------|-------------|---------------------|
| Burger | ₹ 150 | Quantity: 1 | Remove |
| Nuddles | ₹ 100 | Quantity: 3 | Remove |
| cheese Pizza | ₹ 180 | Quantity: 4 | Remove |
| Fry Chicken | ₹ 120 | Quantity: 1 | Remove |
| Total 550.00: | | | |

Proceed Complete

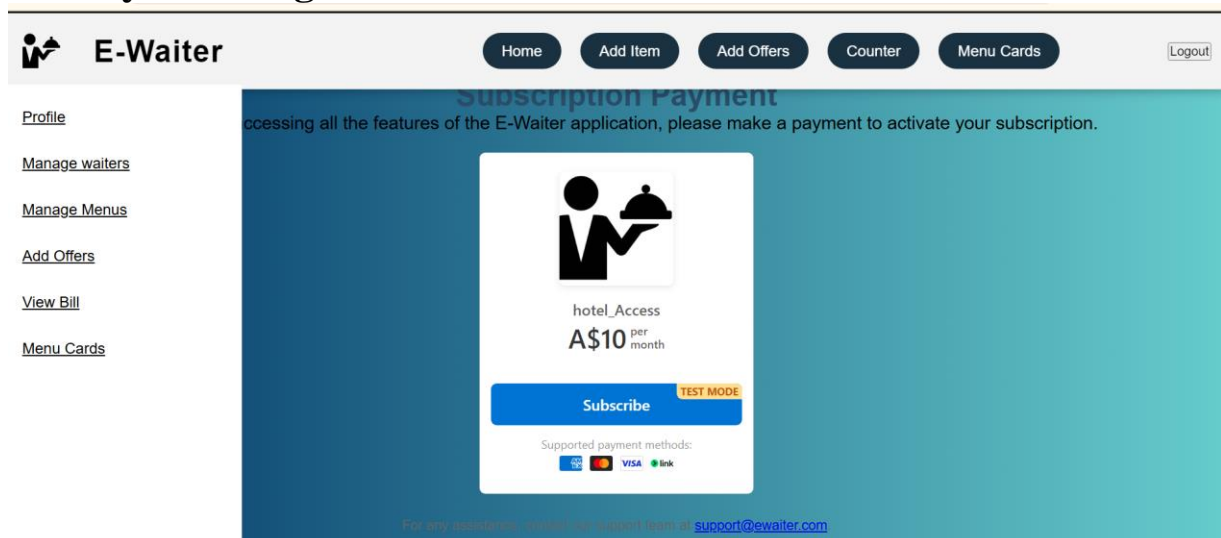
Table 7


View Bill

➤ QR Code Based Menu Card :



➤ Payment Page :




 e-waiter **TEST MODE**

Subscribe to hotel_Access

A\$10.00 per month

Pay amount and manage your hotel easily and keep track of all records easily



Pay with  link

Or pay with card

Email

Card information

1234 1234 1234 1234



MM / YY

CVC



Cardholder name

Full name on card



Country or region

India



☐ Securely save my information for 1-click checkout
Pay faster on e-waiter and everywhere Link is accepted.

➤ Products Page :

 **E-Waiter**  Logout

Profile

Manage waiters

Manage Menus

Add Offers

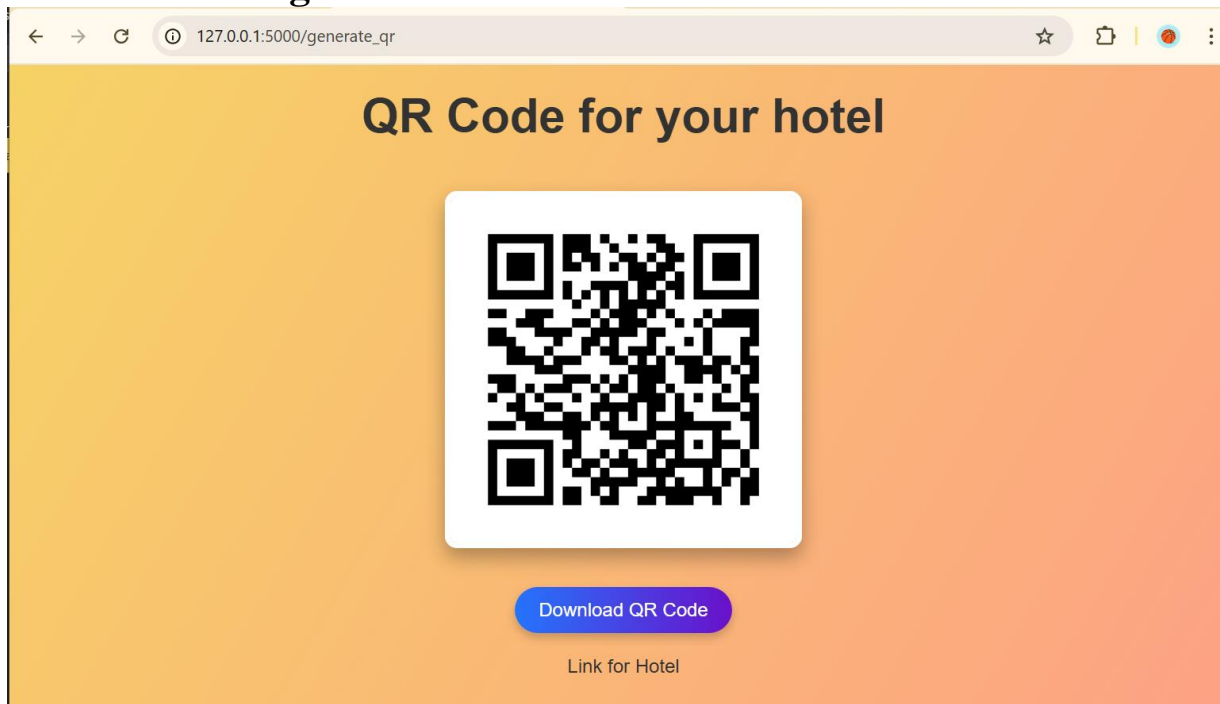
View Bill

Menu Cards

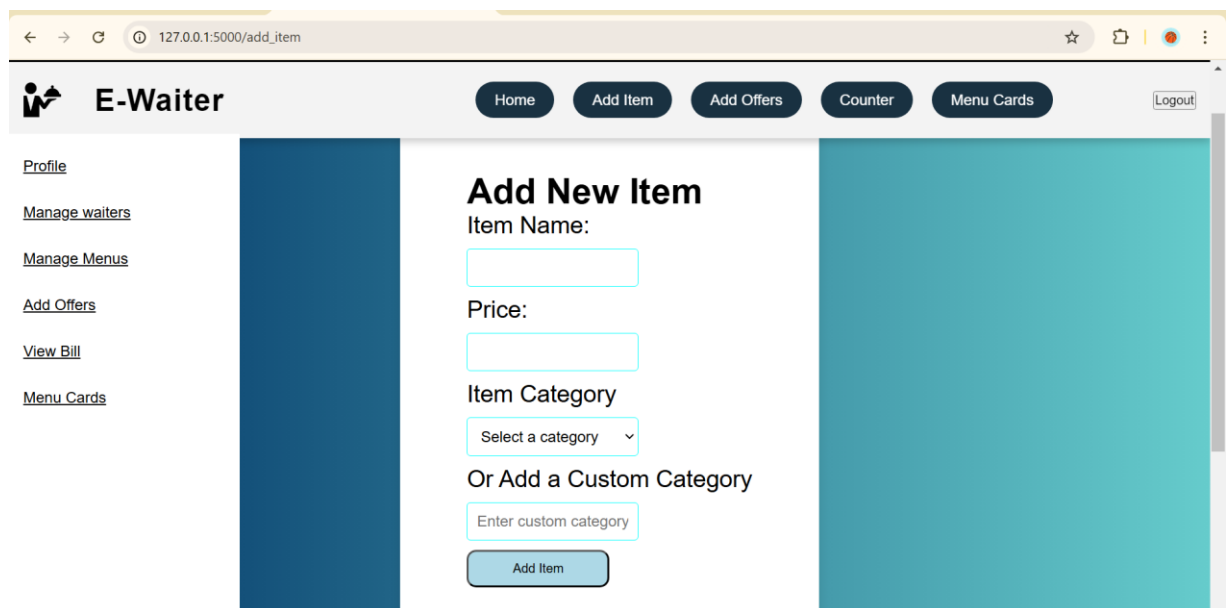
Manage Menu's

| ID | Item Name | Price | Category | Action |
|----|-----------|--------|------------|---|
| 1 | pizza | 96.00 | food1 | <button>Edit</button> <button>Delete</button> |
| 2 | Burger | 150.00 | Snaks | <button>Edit</button> <button>Delete</button> |
| 3 | Nuddles | 100.00 | Snaks | <button>Edit</button> <button>Delete</button> |
| 4 | Sandwich | 120.00 | Snaks | <button>Edit</button> <button>Delete</button> |
| 5 | Mango | 58.00 | Cold Drink | <button>Edit</button> <button>Delete</button> |
| 6 | Fizz | 50.00 | Cold Drink | <button>Edit</button> <button>Delete</button> |

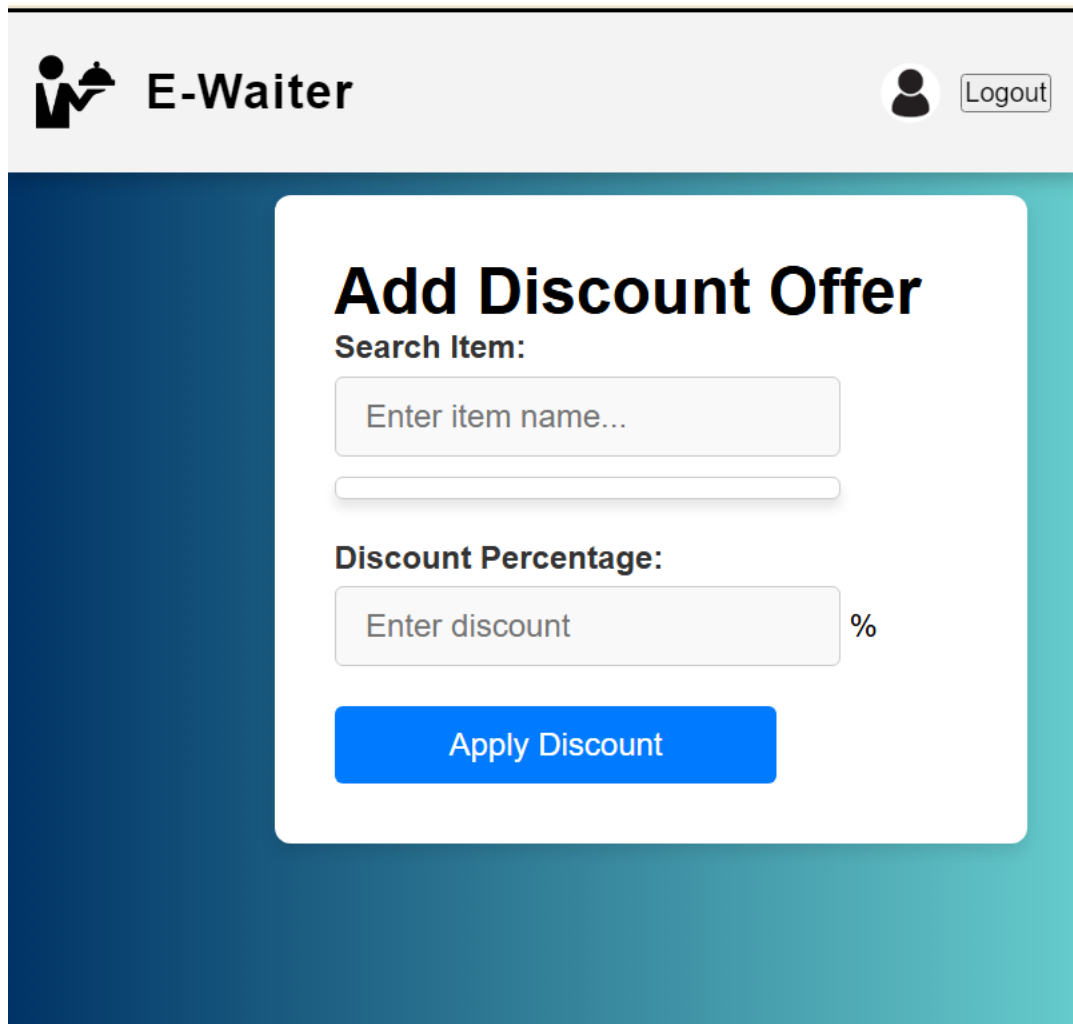
➤ OR code Page for Menu cards :



➤ Add Product Page :



➤ Add offer Page :



The screenshot shows the 'Add Discount Offer' page of the E-Waiter application. The page has a dark blue sidebar on the left and a light blue header. The header contains the E-Waiter logo and a 'Logout' button. The main content area is white and contains the following elements:

- Add Discount Offer** (Section Header)
- Search Item:** (Label)
- (Text Input)
- (Empty Text Input)
- Discount Percentage:** (Label)
- (Text Input)
- % (Percentage Symbol)
- Apply Discount** (Blue Button)

6.2 CODING :

A] App.py :

```
from flask import Flask,render_template, request, jsonify,
session,redirect,url_for,make_response,flash,send_file,json
from flask_login import LoginManager, login_user, logout_user,
login_required, current_user
from flask_login import UserMixin
import uuid
from flask_sqlalchemy import SQLAlchemy
from itsdangerous import URLSafeSerializer,BadSignature
from flask_bcrypt import Bcrypt
from datetime import timedelta,datetime,date,timezone
datetime.now(timezone.utc)
from werkzeug.utils import secure_filename
import qrcode,base64
from io import BytesIO
from decimal import Decimal
from flask_mail import Mail, Message
import os
import stripe

app = Flask(__name__)
app.secret_key = 'ask'
app.config['SQLALCHEMY_DATABASE_URI'] =
'mysql+pymysql://root:@localhost/hotel'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
app.config['PERMANENT_SESSION_LIFETIME'] = timedelta(minutes=30)
app.config['SESSION_COOKIE_SECURE'] = True # Use secure cookies
app.config['SESSION_COOKIE_HTTPONLY'] = True # Make cookies
inaccessible to JavaScript
app.config['SESSION_COOKIE_SAMESITE'] = 'Lax' # Prevent CSRF in
cross-site requests
app.config['SESSION_PERMANENT'] = False # Session expires when the
browser is closed
app.config['MAIL_USERNAME'] = 'ak1074834@gmail.com'
app.config['MAIL_PASSWORD'] = '/ask.in/kumbhar'
YOUR_DOMAIN = "http://localhost:5000"
```

```
stripe.api_key =  
"sk_test_51QgBIRGyyghmbto2tKxFVoWbXjmdOToBjou9dnkKBeoe7A3FHF  
J10tHkFIUG7OY3xwzKh9tAfQm8TPok73RAinDT00NST6StcD"
```

```
webhook_secret = 'whsec_mwRYZMAVuOtuXykwJJWwvss89J7w0hSx'
```

```
UPLOAD_FOLDER = 'static/uploads/' # Folder to store uploaded photos  
ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg', 'gif'}
```

```
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
```

```
login_manager = LoginManager()  
login_manager.init_app(app)  
login_manager.login_view = 'home'
```

```
db = SQLAlchemy(app)  
bcrypt = Bcrypt(app)  
serializer = URLSafeSerializer(app.config['SECRET_KEY'])
```

```
app.config['MAIL_SERVER'] = 'smtp.gmail.com'  
app.config['MAIL_PORT'] = 587  
app.config['MAIL_USE_TLS'] = True  
mail = Mail(app)
```

```
@app.after_request  
def add_no_cache_headers(response):  
    if response.content_type and 'text/html' in response.content_type:  
        response.headers['Cache-Control'] = 'no-store, no-cache, must-revalidate,  
proxy-revalidate'  
        response.headers['Pragma'] = 'no-cache'  
        response.headers['Expires'] = '0'  
    return response
```

```
@login_manager.user_loader  
def load_user(user_id):  
    owner = Owner.query.filter_by(owner_id=user_id).first()  
    user = User.query.filter_by(user_id=user_id).first()  
    waiter = Waiter.query.filter_by(waiter_id=user_id).first()  
    if owner:  
        return owner
```

```
elif user:
    return user
elif waiter:
    return waiter
return None # Explicitly return None if no user is found

def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in
ALLOWED_EXTENSIONS
Route's :

@app.route('/')
def home():
    if current_user.is_authenticated and isinstance(current_user, Owner):
        # Check subscription status
        if current_user.subscription_status == 'expired':
            flash("Your subscription has expired. Please renew.", "warning")
            return redirect(url_for('payment_page')) # Redirect to payment

    owner_id = current_user.owner_id
    today_start = datetime.combine(date.today(), datetime.min.time())
    today_end = datetime.combine(date.today(), datetime.max.time())

    # Fetch today's bills
    bills = Bill.query.filter(
        Bill.owner_id == owner_id,
        Bill.bill_date >= today_start,
        Bill.bill_date <= today_end
    ).order_by(Bill.bill_date.desc()).all()

    # Calculate income
    today_income = sum(bill.total_amount for bill in bills)

    # Convert details to JSON if stored as a string
    for bill in bills:
        if isinstance(bill.details, str):
            bill.details = json.loads(bill.details)

    return render_template(
        'index.html',
```

```
        owner=current_user.owner_name,
        owner_id=owner_id,
        bills=bills,
        total_amount=today_income,
        todays_bills=len(bills)
    )

    return render_template('index.html')

@app.route('/waiter_register', methods=['GET'])
def waiter_register():
    # Render the waiter registration page
    return render_template('waiter_register.html')

# Registration route
@app.route('/register', methods=['POST', 'GET'])
def register():
    if request.method == 'GET':
        owner_id = request.args.get('owner_id', None)
        # Render the registration page for GET request
        return render_template('register.html', owner_id=owner_id)

    if request.method == 'POST':
        user_type = request.form.get('user_type') # 'owner', 'user', or 'waiter'
        username = request.form.get('username')
        email = request.form.get('email', None) # Email might not be required for
waiter
        password = request.form.get('password')

        if user_type == 'owner':
            # Check if owner already exists
            existing_owner_email =
Owner.query.filter_by(email=email).first()
            existing_owner_name =
Owner.query.filter_by(owner_name=username).first()
            if existing_owner_email:
                flash("Email already registered for owner", "error")
                return redirect(url_for('register'))
            if existing_owner_name:
                flash("Username already registered for owner", "error")
                return redirect(url_for('register'))
```



```
# Hash the password
hashed_password =
bcrypt.generate_password_hash(password).decode('utf-8')

# Create a new owner
new_owner = Owner(owner_name=username, email=email,
password=hashed_password)
db.session.add(new_owner)
db.session.flush() # Flush to get the owner_id before committing

# Add subscription details

trial_days = 1 # Set trial period duration
subscription = Subscription(
    subscription_id=str(uuid.uuid4()), # Ensure unique ID
    owner_id=new_owner.owner_id,
    owner_name=new_owner.owner_name,
    owner_email=new_owner.email,
    status='trial',
    start_date=datetime.utcnow(),
    end_date=datetime.utcnow() + timedelta(days=trial_days)
)

db.session.add(subscription)

# Commit all changes
db.session.commit()

flash("Registration successful! Your free trial has started.",
"success")
return redirect(url_for('owner_login'))

elif user_type == 'user':
    # User registration: owner_id is required
    owner_id = request.form.get('owner_id') # Required field
    if not owner_id:
        return jsonify({'message': 'Owner ID is required for user
registration.'}), 400
```

```
# Validate owner existence
decrypted_owner_id = serializer.loads(owner_id)
owner = Owner.query.get(decrypted_owner_id)
if not owner:
    flash("Invalid owner ", "error")
    return redirect(url_for('register'))

# Check if user already exists
existing_user_email = User.query.filter_by(email=email).first()
existing_user_name = User.query.filter_by(username=username).first()
if existing_user_email:
    return jsonify({'message': 'Email already registered for a user.'}), 409
if existing_user_name:
    flash("User name already registered", "error")
    return redirect(url_for('register'))

# Hash the password
hashed_password =
bcrypt.generate_password_hash(password).decode('utf-8')

# Create a new user
new_user = User(owner_id=decrypted_owner_id, username=username,
email=email, password=hashed_password)
db.session.add(new_user)
db.session.commit()
return redirect(url_for('user_login'))

elif user_type == 'waiter':
    # Get form inputs
    owner_id = request.form.get('owner_id')
    username = request.form.get('username') # Ensure username is
retrieved
    password = request.form.get('password') # Ensure password is
retrieved

    # Validate inputs
    if not owner_id or not username or not password:
        flash("Owner ID, username, and password are required",
"error")
        return redirect(url_for('waiter_register'))
```

```
# Check if owner exists
    owner = Owner.query.get(owner_id)
    if not owner:
        flash("Invalid owner ID", "error")
        return redirect(url_for('waiter_register'))

# Check if waiter already exists
    existing_waiter_name =
Waiter.query.filter_by(waiter_name=username).first()
    if existing_waiter_name:
        flash("Username already registered for a waiter, Try another",
"error")
        return redirect(url_for('waiter_register'))

# Hash the password
    try:
        hashed_password =
bcrypt.generate_password_hash(password).decode('utf-8')
    except Exception as e:
        flash(f"Error hashing password: {e}", "error")
        return redirect(url_for('waiter_register'))

# Create and add a new waiter
    try:
        new_waiter = Waiter(owner_id=owner_id,
waiter_name=username, password=hashed_password)
        db.session.add(new_waiter)
        db.session.commit()
        flash("Waiter registered successfully!", "success")
        return redirect(url_for('waiter_login'))
    except Exception as e:
        db.session.rollback() # Rollback in case of an error
        flash(f"Error registering waiter: {e}", "error")
        return redirect(url_for('waiter_register'))

else:
    return jsonify({'message': 'Invalid user type.'}), 400
```

```
@app.route('/waiter_dashboard', methods=['GET', 'POST'])
@login_required
def waiter_dashboard():
    items_by_category = {}
    owner_name = "Unknown Owner"
    items = []
    if isinstance(current_user, Owner) and not isinstance(current_user, Waiter):
        # Redirect to an error page or deny access
        return render_template('error.html', title="Access Denied",
message="Only waiters can perform this action."), 403

    if isinstance(current_user, Waiter) :
        # Fetch items for the waiter if authenticated
        owner = db.session.get(Owner, current_user.owner_id)

        if owner:
            items = Items.query.filter_by(owner_id=current_user.owner_id).all()
            owner_name = owner.owner_name

# No owner ID provided, no items to display

# Group items by category
for item in items:
    if item.category not in items_by_category:
        items_by_category[item.category] = []
    items_by_category[item.category].append({
        "item_name": item.item_name,
        "price": item.price,
        "id": item.item_id # Include other fields if needed
    })

waiter_id=current_user.waiter_id
owner_id=current_user.owner_id
try :
    owner_profile =
Owner_profile.query.filter_by(owner_id=current_user.owner_id).first()
    table_count = owner_profile.number_of_tables
except Exception as e :
    table_count=0
# Render the waiter dashboard template
```

```
    return render_template('waiter_dashboard.html',
owner_id=owner_id,waiter_id=waiter_id,items_by_category=items_by_category,table_count=table_count)
```

```
@app.route('/hotel', methods=['GET', 'POST'])
```

```
def hotel():
```

```
    items_by_category = { }
```

```
    owner_name = "Unknown Owner"
```

```
    if current_user.is_authenticated and isinstance(current_user, (User,
Owner,Waiter)):
```

```
        # Fetch the owner and items for authenticated users
```

```
        owner = db.session.get(Owner, current_user.owner_id)
```

```
        items = Items.query.filter_by(owner_id=current_user.owner_id).all()
```

```
        owner_name = owner.owner_name if owner else "Unknown Owner"
```

```
    else:
```

```
        # Fetch the owner and items for non-authenticated users
```

```
        owner_id = request.args.get('owner_id')
```

```
        if owner_id:
```

```
            owner_id = serializer.loads(owner_id) # Decode the owner ID
```

```
            owner = db.session.get(Owner, owner_id)
```

```
            items = Items.query.filter_by(owner_id=owner_id).all()
```

```
            owner_name = owner.owner_name if owner else "Unknown Owner"
```

```
        else:
```

```
            items = [] # No owner ID provided, so no items to display
```

```
    # Group items by category
```

```
    for item in items:
```

```
        if item.category not in items_by_category:
```

```
            items_by_category[item.category] = [] # Initialize a new list for the
category
```

```
            items_by_category[item.category].append(item) # Append item to the
correct category
```

```
    # Render the template
```

```
    return render_template('hotel.html', owner=owner_name,
items_by_category=items_by_category)
```

```
@app.route('/add_item', methods=['GET', 'POST'])
```

```
@login_required
def add_item():
    if not isinstance(current_user, Owner) and not isinstance(current_user,
Waiter):
        # Redirect to an error page or deny access
        return render_template('error.html', title="Access Denied",
message="Only owners can perform this action."), 403

    items_by_category = { }
    items = Items.query.filter_by(owner_id=current_user.owner_id).all()
    for item in items:
        if item.category not in items_by_category:
            items_by_category[item.category] = []
        items_by_category[item.category].append(item)

    if request.method == 'GET':
        return render_template('add_product.html', items=items,
items_by_category=items_by_category)

    if request.method == 'POST':
        # Handle form submission
        item_name = request.form.get('itemName')
        item_price = request.form.get('itemPrice')
        selected_category = request.form.get('itemCategory')
        custom_category = request.form.get('customCategory')

        # Determine which category to use
        item_category = custom_category if custom_category.strip() else
selected_category

        if not item_category:
            flash("Please select or enter a category.", "error")
            return render_template('add_product.html', items=items,
items_by_category=items_by_category)

        # Create and save the new item
        new_item = Items(
            owner_id=current_user.owner_id,
            item_name=item_name,
            price=item_price,
            category=item_category
        )
```

```
db.session.add(new_item)
db.session.commit()
```

```
flash("Item added successfully", "success")
return redirect(url_for('add_item'))
```

```
@app.route('/add_menu_item', methods=['POST'])
@login_required
def add_menu_item():
    data = request.json
    table_id = data['table_id']
    waiter_id = data['waiter_id']
    owner_id = data['owner_id']
    item_name = data['item_name']
    price = data['price']
    quantity = data['quantity']

    # Find the table session for the given table and owner
    session = TableSession.query.filter_by(table_id=table_id,
owner_id=owner_id).first()

    if not session:
        return jsonify({'message': 'Table not found or unauthorized access.'}), 403

    # Add item to session data (the bill)
    session_data = json.loads(session.data)
    # Check if the item already exists in the session data, then update quantity
    item_found = False
    for item in session_data:
        if item['item_name'] == item_name:
            item['quantity'] += quantity # Increase quantity if the item already
exists
            item_found = True
            session.total = (session.total or 0) + price
            break

    if not item_found:
        # If the item doesn't exist, add it to the session data
        session.total = (session.total or 0) + price
```

```
        session_data.append({'item_name': item_name, 'price': price, 'quantity':  
quantity})
```

```
    session.data = json.dumps(session_data)  
    db.session.commit()
```

```
    return jsonify({'message': 'Item added/updated successfully.', 'data':  
session_data}), 200
```

```
@app.route('/complete_bill', methods=['POST'])
```

```
@login_required
```

```
def complete_bill():
```

```
    data = request.json
```

```
    table_id = data['table_id']
```

```
    owner_id = data['owner_id']
```

```
    total_amount = data.get('total_amount', 0.0) # Assuming the total amount is  
sent in the request
```

```
    session = TableSession.query.filter_by(table_id=table_id,  
owner_id=owner_id).first()
```

```
    if not session:
```

```
        return jsonify({'message': 'Table not found.'}), 404
```

```
    # Calculate the total amount from session.data
```

```
    session_data = json.loads(session.data) # Convert JSON string to Python list  
of dictionaries
```

```
    total_amount = sum(item['price'] * item['quantity'] for item in session_data)
```

```
    # Calculate total
```

```
    # Add a new bill to the Bills table
```

```
    new_bill = Bill(  
        table_id=table_id,  
        owner_id=owner_id,  
        waiter_id=session.waiter_id,  
        total_amount=total_amount,  
        details=session.data  
    )
```

```
    db.session.add(new_bill)
```



```
# Delete the session
db.session.delete(session)
db.session.commit()
```

```
return jsonify({'message': f'Table {table_id} is now free, and the bill has
been saved.'}), 200
```

```
if __name__ == '__main__':
    app.run(debug=True)
```

B] index.html :

```
{% extends "layout.html" %}
{% block body %} <!-- Main Content -->
<main>
    <br><br><br>
    {% if current_user.is_authenticated and current_user.is_owner %}
        {% if current_user.subscription_status == 'trial' or
current_user.subscription_status == 'active' %}
            <div>
                <h1 style="position: absolute; left: 300px;">Welcome to Your
Admin Dashboard, {{ current_user.owner_name }}</h1>

                {% if current_user.subscription_status == 'trial' %}
                    <p style="color: orange; font: 100; position: relative; left:200px;
top:300px;">
                        You are currently on a free trial. Your trial will expire in <s>
                        <strong>{{ trial_days_remaining }}</strong> days.
                    </p>
                {% endif %}

                <!-- Dashboard content for trial or active users -->
                <div class="homebody">
                    <section class="stats-section">
                        <div class="stat-box">
                            <h2>Today's Bills</h2>
                            <h3 id="totalOrders">{{ todays_bills }}</h3>
                        </div>
                        <div class="stat-box">
```

```
<h2>Today's Income</h2>
<h3 id="totalIncome">{{ total_income }} 1200 Rs</h3>
</div>
<div class="stat-box">
  <h2>Most Ordered Products</h2>
  <h3 id="totalOrders"> cheese Pizza</h3>
</div>
</section>

<!-- Chart Section -->
<br><br>
<h2 style="text-align: center; border: 2px solid white; background-
color: lightskyblue; padding: 10px;">Today's Bills</h2>
<div class="bill-container">
  {% for bill in bills %}
    <div class="bill-section stat-box">
      <h2 style="border-top: 1px solid #ccc;">Bill ID: {{
bill.bill_id }}</h2>
      <div class="bill-details">
        <p><strong>Table Number: {{ bill.table_id
}}</strong></p>
        <p><strong>Waiter ID: {{ bill.waiter_id
}}</strong></p>
        <h4>Bill Details:</h4>
        <table style="width: 100%; border-collapse: collapse;">
          <thead>
            <tr>
              <th style="text-align: left; border-bottom: 2px
solid #ddd; padding: 8px;">Item Name</th>
              <th style="text-align: right; border-bottom: 2px
solid #ddd; padding: 8px;">Quantity</th>
              <th style="text-align: right; border-bottom: 2px
solid #ddd; padding: 8px;">Price</th>
              <th style="text-align: right; border-bottom: 2px
solid #ddd; padding: 8px;">Total</th>
            </tr>
          </thead>
          <tbody>
            {% for item in bill.details %}
```

```
<tr>
    <td style="padding: 8px; border-bottom: 1px
solid #eee;">{{ item.item_name }}</td>
    <td style="padding: 8px; text-align: right;
border-bottom: 1px solid #eee;">{{ item.quantity }}</td>
    <td style="padding: 8px; text-align: right;
border-bottom: 1px solid #eee;">{{ "%.2f" | format(item.price) }}</td>
    <td style="padding: 8px; text-align: right;
border-bottom: 1px solid #eee;">{{ "%.2f" | format(item.price * item.quantity)
}}</td>

</tr>
    {% endfor %}
</tbody>
</table>
</div>
</div>
    {% endfor %}
</div>
    <canvas id="orderChart"></canvas>
</div>
</div>

{% elif current_user.subscription_status == 'expired' %}
    <div class="stat-box" style="position: absolute; left: 25%;">

        <h1>Subscription Expired</h1>
        <h2 style="color: red;">
            Your free trial or subscription has expired. Please <a href="{{
url_for('payment_page') }}">make a payment</a> to continue using the
service.
        </h2>
    </div>
{% endif %}
{% else %}

<!-- About Section -->
<div class="about-container">
    <section class="smart-waiter">
```

```
<div style="max-width: 1200px; margin:auto; display: flex; align-items: center; gap: 20px; flex-wrap: wrap;">
```

```
<!-- Image on the Left -->
```

```
<div class="about-image" style="flex: 1; min-width: 300px; text-align: center; padding: 20px;">
```

```

```

```
</div>
```

```
<!-- Text on the Right -->
```

```
<div id="what-is-smart-waiter" style="flex: 2; min-width: 300px; padding: 10px;">
```

```
<h1 style="color: #2a4d69; font-size: 2.5rem; margin-bottom: 20px;">What is E-Waiter?</h1>
```

```
<p style="color: #4f4f4f; font-size: 1.2rem; line-height: 1.8;">
```

An E-Waiter is a concept that leverages IoT devices to streamline restaurant operations and improve customer service. By connecting various devices such as tablets, smartphones, sensors, and other smart gadgets, restaurants can create a network that enables seamless communication between staff members and customers.

```
</p>
```

```
</div>
```

```
</div>
```

```
</section>
```

```
<section class="about-section">
```

```
<div class="about-content reverse">
```

```
<div class="about-text">
```

```
<h2>Our Vision</h2>
```

<p style="font-size: 1.2rem;">We envision a world where hotel management is seamless and efficient. Our mission is to empower hotels to enhance their guest experience through streamlined processes and cutting-edge technology.</p>

```
</div>
```

```
<div class="about-image">
```

```

```

```
</div>
</div>

<div class="about-content reverse">
  <div class="about-text">
    <h2>Who We Are</h2>
    <p style="font-size: 1.2rem;">We are a group of dedicated college students
passionate about technology and innovation. This project is part of our final
year coursework, developed using Python and MySQL. Our goal is to simplify
hotel management through an intuitive platform that empowers owners to
manage menus, offers, and waiter registrations seamlessly.</p>
  </div>
  <div class="about-image">
    
  </div>
</div>

<div class="about-values">
  <h2>Our Core Values</h2>
  <div class="values-grid">
    <div class="value-item">
      <h3>Innovation</h3>
      <p>We strive to stay ahead by creating innovative solutions
tailored to the needs of modern hotels.</p>
    </div>
    <div class="value-item">
      <h3>Reliability</h3>
      <p>We are committed to delivering robust, user-friendly, and
dependable tools for hotel operations.</p>
    </div>
    <div class="value-item">
      <h3>Customer Focus</h3>
      <p>Your success is our success. We prioritize your needs and
deliver solutions that exceed expectations.</p>
    </div>
  </div>
</div>
```

```
<section class="benefit" id="smart-waiter" style="background-color:
#f0f8ff; padding: 50px;">
  <div id="benefits-smart-waiter" style="margin-bottom: 40px;">
    <h2 style="color: #2a4d69; font-size: 2rem; margin-bottom: 20px;
text-align: center;">Benefits of Smart Waiter Technology</h2>
    <ul style="list-style-type: disc; padding-left: 40px; color: #4f4f4f;
font-size: 1.2rem; line-height: 1.8;">
      <li><strong>Enhanced Customer Experience:</strong>
Customers can place orders, request service, and provide feedback more
efficiently, improving satisfaction and loyalty.</li>
      <li><strong>Mobile Ordering:</strong> Customers can use
their smartphones or tablets to place orders, eliminating physical menus and
reducing wait times.</li>
      <li><strong>Increased Efficiency:</strong> Automation of
tasks like order processing, inventory management, and table turnover results
in faster service and reduced waiting times.</li>
      <li><strong>Data-driven Insights:</strong> IoT devices collect
valuable data on customer preferences, peak hours, and popular dishes,
enabling better decision-making.</li>
      <li><strong>Cost Savings:</strong> Reduces labor costs and
minimizes food waste through automation and inventory management.</li>
    </ul>
  </div>
</section>
</section>
</div>
{ % endif % }
</main>
{ % endblock % }
```

Payment.HTML :

```
{ % extends "layout.html" % }
{ % block body % }
<main style="padding: 20px; text-align: center; font-family: Arial, sans-
serif;">
  <h1 style="color: #2a4d69;">Subscription Payment</h1>
  <p style="font-size: 1.2rem; margin-bottom: 20px;">
```

To continue accessing all the features of the E-Waiter application, please make a payment to activate your subscription.

</p>

```
<script async
src="https://js.stripe.com/v3/buy-button.js">
</script>
```

```
<stripe-buy-button
buy-button-id="buy_btn_1QvnD2Gyyghmbto2caYtXRp0"
publishable-
key="pk_test_51QgBIRGyyghmbto2uH4lC2HVDmRCawS0JKcKXmWxmY
GoVe1GzgBtRnxCDBjB11f9HpCEPPfNpWf1PyomaWzWEn800JKNroef"
>
</stripe-buy-button>
```

<p style="margin-top: 20px; font-size: 0.9rem; color: #666;">

For any assistance, contact our support team at support@ewaiter.com.

</p>

</main>

```
<script async src="https://js.stripe.com/v3/buy-button.js"></script>
{ % endblock % }
```

6.3 REPORTS :-

▪ Bill :

[Home](#) [Add Item](#) [Add Offers](#) [Counter](#) [Menu Cards](#)

Today's Bills

Bill ID: 29
Table ID: 1
Waiter ID: 10
Date: 2025-02-24 21:39:39

Bill Details:

| Item Name | Quantity | Price | Total |
|--------------|----------|--------|------------------------------|
| Burger | 1 | 150.00 | 150.00 |
| Nuddles | 3 | 100.00 | 300.00 |
| cheese Pizza | 4 | 180.00 | 720.00 |
| Fry Chicken | 1 | 120.00 | 120.00 |
| | | | Total Amount: 1290.00 |

Print

[Home](#) [Add Item](#) \ [Add Offers](#) [Counter](#) [Menu Cards](#)

Today's Bills

Bill ID: 30
Table ID: 5
Waiter ID: 10
Date: 2025-02-24 22:31:18

Bill Details:

| Item Name | Quantity | Price | Total |
|-------------|----------|--------|------------------------------|
| Mango | 1 | 58.00 | 58.00 |
| Fizz | 8 | 50.00 | 400.00 |
| Sprite | 7 | 50.00 | 350.00 |
| Fry Chicken | 5 | 120.00 | 600.00 |
| | | | Total Amount: 1408.00 |

Print

[Home](#) [Add Item](#) \ [Add Offers](#) [Counter](#) [Menu Cards](#)

Today's Bills


Bill ID: 31
Table ID: 1
Waiter ID: 11
Date: 2025-02-24 22:39:13

Bill Details:

| Item Name | Quantity | Price | Total |
|--------------|----------|--------|---------|
| pizza | 1 | 200.00 | 200.00 |
| sabdwich | 7 | 150.00 | 1050.00 |
| cheese Pizza | 3 | 150.00 | 450.00 |
| Chai | 1 | 14.00 | 14.00 |

Total Amount: 1714.00

Print

 **E-Waiter**

[Home](#) [Add Item](#) [Add Offers](#) [Counter](#) [Menu Cards](#) [Logout](#)

[Profile](#)
[Manage waiters](#)
[Manage Menus](#)
[Add Offers](#)
[View Bill](#)
[Menu Cards](#)

Welcome to Your Admin Dashboard, Hotel Alankar

Today's Bills
5

Today's Income
1200 Rs

Most Ordered Products
cheese Pizza

Today's Bills

Bill ID: 30

Bill ID: 29

CHAPTER -7

USER MANUAL :

The **E-Waiter** system is a digital platform designed to help hotel owners and waiters manage order-taking, billing, and communication. For Hotel Owners:

1. Registration:

- Open the E-Waiter platform on a browser.
- Click on the **Register** button.
- Enter your name, email, and password, then click **Submit**.

2. Login:

- Click on the **Login** button.
- Enter your email and password, then click **Submit**.

3. Subscription Payment:

- Go to the **Subscription** section.
- Select a subscription plan.
- Click **Pay Now** and enter payment details to complete the transaction.

4. Manage Waiters:

- Go to the **Waiter Management** section.
- Provide the `owner_id` to waiters for registration.
- View and manage the list of waiters under your hotel.

5. Manage Menus and Offers:

- Navigate to the **Menu Management** section.
- Add, update, or delete menu items and special offers.
- Ensure all changes are saved for real-time updates.

For Waiters:

1. Registration:

- Open the E-Waiter platform on a smartphone.
- Click **Register as Waiter**.
- Enter your name, password, and the `owner_id` provided by the hotel owner.
- Click **Submit** to complete the registration.

2. Login:

- Click **Login as Waiter**.
- Enter your credentials and click **Submit**.

3. Taking Orders:

- Select the table number.
- Choose menu items and specify quantities.
- Review the order and click **Complete Bill** to send it to the counter.

4. Billing:

- After order completion, the system automatically generates the bill.
- The bill is instantly transferred to the counter for printing using the smart printer.

5. SYSTEM WORKFLOW

1. **Owner Registers and Makes Payment:** Gains access to all system features.
2. **Waiters Register Using Owner_ID:** Establishes a secure connection between owner and waiters.
3. **Order Placement:** Waiters take orders digitally, reducing errors and delays.
4. **Order Completion:** System generates the bill automatically.
5. **Bill Transfer:** The bill is instantly sent to the counter for printing and payment collection.

CHAPTER - 8

FUTURE ENHANCEMENT :

- **Dedicated Kitchen Interface:**

- Introduce a dedicated interface for the kitchen staff to receive and manage orders in real-time.
- Display order details, preparation status, and priority levels, helping the kitchen team work more efficiently.

- **Multi-Language Support:**

- Enable the system to support multiple languages, allowing both hotel owners and waiters to use the platform in their preferred language. This will make the system more accessible to users from diverse regions.

- **Table Reservation Module:**

- Enable customers to reserve tables online, helping hotels manage their seating more effectively.

- **Advanced Analytics and Reports:**

- Provide hotel owners with detailed analytics on sales, popular menu items, waiter performance, and customer feedback to help optimize business operations.

CHAPTER – 9

ADVANTAGES OF SYSTEM :

- **Digital Order Management:** Eliminates the need for pen and paper, reducing errors and improving order accuracy.
- **Faster Service:** Orders are instantly sent to the kitchen, speeding up food preparation and delivery.
- **Automated Billing:** Bills are generated automatically and transferred to the billing counter, saving time and minimizing calculation errors.
- **Easy Waiter Registration:** Waiters can quickly register using the owner's unique `owner_id`, ensuring a secure connection.
- **Real-Time Updates:** Menu changes and special offers are instantly updated for all users, ensuring everyone has the latest information.
- **Improved Communication:** Reduces communication gaps between waiters, kitchen staff, and the billing counter, ensuring smoother operations.
- **User-Friendly Interface:** Simple and intuitive design makes it easy to use for both hotel owners and waiters.
- **Cost-Effective:** Compatible with smartphones and standard hardware, eliminating the need for expensive devices.
- **Secure Payments:** Online subscription payments are processed securely, ensuring data protection.
- **Eco-Friendly:** Reduces paper usage, promoting environmentally friendly practices.
- **Enhanced Customer Experience:** Faster service and accurate billing improve customer satisfaction and loyalty.
- **Flexible Access:** Owners and waiters can access the system from any device with an internet connection, ensuring flexibility and convenience.

CHAPTER -10

LIMITATION OF SYSTEM :

- **Internet Dependency:** The system requires a stable internet connection for real-time communication, which may cause disruptions if the connection is lost.
- **Device Compatibility:** The system relies on smartphones and tablets, which may not be available or affordable for all hotels and waiters.
- **Technical Knowledge Required:** Users need basic technical knowledge to operate the system, which might be challenging for non-tech-savvy individuals.
- **Initial Setup Cost:** Although cost-effective, the initial setup may involve expenses for smart printers, devices, and subscriptions.
- **Limited Offline Functionality:** The system has limited functionality in offline mode, affecting order-taking and communication during network outages.
- **Customization Limitations:** The system may have limited customization options, which might not meet the specific needs of all hotels.

CHAPTER -11

CONCLUSION :

The **E-Waiter** system provides a modern, digital solution for managing hotel operations, replacing traditional pen-and-paper methods with a more efficient and accurate process. By allowing hotel owners to register, manage waiters, control menus, and handle subscriptions, the system enhances overall business operations.

Waiters can register using the owner's unique `owner_id` and take orders digitally via smartphones, ensuring faster communication with the kitchen. The automated billing process reduces manual errors and ensures bills are instantly transferred to the counter for quick payment.

With its user-friendly interface, real-time updates, and secure data management, the **E-Waiter** system improves service speed, reduces errors, and enhances customer satisfaction. Although it has some drawbacks, such as internet dependency and setup costs, its advantages, including faster service, automated processes, and reduced paperwork, outweigh these limitations. The system is a reliable, cost-effective solution that streamlines daily operations and contributes to the growth and efficiency of the hospitality industry.

CHAPTER -12

BIBLIOGRAPHY :

- **Books:**

- *Database Management Systems* by Raghu Ramakrishnan – For designing relational databases.
- *Web Development with Flask* by Miguel Grinberg – For building web applications using Flask.

- **Web Resources:**

- Stripe Official Website – For understanding online payment integration and webhooks.
- MySQL Official Documentation – For database design and management.
- Flask Official Documentation – For backend development.

- **Other Sources:**

- Online tutorials and coding platforms like YouTube, Stack Overflow, and GitHub – For troubleshooting and implementing specific features.
- E-Waiter System Project Guidelines – For aligning the development process with project requirements.
- Chatgpt .