YUMHUB:

A Food Stall Application

Web Development Framework Using Python (23AI002)

Submitted by

Ayush Bansal (2210993778)

Manya Saini (2210993814)

Semester: 4th

BE-CSE (Artificial Intelligence)

Submitted To

Rajan Kumar

Assistant Professor, Department of CSE(AI),

CUIET, Chitkara University

CHITKARA UNIVERSITY INSITUTE OF ENGINEERING & TECHNOLOGY

CHITKARA UNIVERSITY, RAJPURA

March, 2024

Table of Contents

1. Introduction

The food industry has experienced significant growth and transformation in recent years, driven by changing consumer preferences and advancements in technology. However, small-scale food stall owners often face numerous challenges in managing their orders and analyzing their profits due to limited resources, infrastructure, and technological capabilities. These limitations hinder their ability to effectively compete with larger establishments and optimize their business operations.

To address these challenges, we have undertaken the development of a web-based application that aims to provide small food stall owners with a comprehensive solution for order management. The application leverages modern web technologies to create a user-friendly and intuitive platform that empowers food stall owners to efficiently manage their orders, and track sales.

The primary objective of this project is to develop a cost-effective and accessible solution that caters specifically to the needs of small food stall owners. By implementing this application, we aim to streamline their order management processes, enhance their decision-making capabilities, and ultimately improve their overall profitability.

In this report, we will provide a detailed account of the project's objectives, methodology, implementation, and key features. Additionally, we will highlight the potential benefits this application offers to small food stall owners, including increased efficiency, and improved customer satisfaction.

Through the development and implementation of this web-based application, we envision enabling small food stall owners to compete effectively in the dynamic food industry, while empowering them to achieve their business goals and drive sustainable growth.

2. Methodology

The methodology employed in the development of YUMHUB: A Food Stall Application, encompasses several key steps and strategies aimed at ensuring the successful implementation of the project's objectives.

2.1 Requirement Analysis:

- Conducted thorough market research to understand the challenges faced by small-scale food vendors in the industry.
- Gathered requirements through stakeholder interviews, surveys, and feedback sessions to identify key functionalities and features needed in the platform.
- Analyzed competitor solutions to identify gaps and opportunities for differentiation.

2.2 Technology Selection:

- Chose Django, a high-level Python web framework, for its robust features, scalability, and extensive built-in functionalities.
- Selected SQLite as the default database management system for its lightweight nature and seamless integration with Django.
- Utilized HTML, CSS, and JavaScript for frontend development to ensure a responsive and interactive user interface.

2.3 Architecture Design:

- Designed a scalable and modular architecture using microservices to decouple components and enable independent deployment and scaling.
- Divided functionalities such as user authentication, menu management, and order processing into separate services for better maintainability and flexibility.
- Defined clear and consistent RESTful APIs for communication between frontend and backend components, adhering to REST principles for resource naming and representation.
- Designed the database schema using SQL, ORM, ensuring efficient data storage and retrieval.

2.4 Implementation:

- Developed the backend logic using Python and Django’s built-in features for implementing core functionalities such as user authentication, menu creation, order processing, and payment integration.
- Designed and implemented the user interface using HTML, CSS, and JavaScript, focusing on usability and responsiveness across different devices.

- Integrated third-party libraries where necessary to enhance functionality and streamline development.
- Conducted thorough testing at each stage of development to ensure the reliability, performance, and security of the platform.
- Utilized Django's ORM (Object-Relational Mapping) to interact with the database and perform CRUD operations, ensuring data integrity and security.

By adhering to a systematic methodology encompassing requirement analysis, technology selection, architecture design, implementation, YUMHUB is developed to meet the diverse needs of small scale food vendors, seeking a method, to diversify their business in order stand close to high scale food distributors.

3. Tools and Technologies

YUMHUB leverages a robust selection of tools and technologies to deliver an innovative food stall application tailored for small scaled vendors' needs. These tools are thoughtfully chosen to ensure efficiency, scalability, and robustness throughout the project lifecycle, and ensure seamless integration with existing food industry. Below is a detailed overview of each component:

3.1 Database Management System:

- ***SQLite***: YUMHUB utilizes SQLite as its primary database management system. It is a lightweight, serverless, and self-contained database engine integrated seamlessly with Django, the web framework used for development. SQLite provides simplicity and efficiency in data storage and retrieval, making it ideal for small to medium-sized web applications.

3.2 Frontend Technologies:

- ***HTML***: Hypertext Markup Language is the standard markup language used for structuring the content of web pages in YUMHUB. HTML provides the foundation for presenting text, images, and other elements on the user interface.
- ***CSS*****:** Cascading Style Sheets are employed for styling and layout purposes in YUMHUB. CSS enables developers to customize the appearance of HTML elements, ensuring a visually appealing and user-friendly interface.
- ***JavaScript*****:** JavaScript enhances the interactivity and dynamic behavior of YUMHUB's frontend. It enables features such as form validation, dynamic content updates, and asynchronous communication with the server, enhancing the overall user experience.

3.3 Backend Framework:

- ***Django*****:** YUMHUB is built using the Django web framework, which follows the Model-View-Template (MVT) architecture pattern. Django provides a high-level, Python-based framework for rapid development of secure and scalable web applications. It offers built-in features for authentication, database management, URL routing, and template rendering, streamlining the development process.

3.4 Django-Admin:

- ***User Authentication***: Implemented user authentication using Django's built-in authentication framework, which provides robust features for user login, logout, password management, and session handling.

- ***Access Control***: Utilized Django's authentication system to control access to different views and functionalities within the application based on user authentication status and permissions. This ensured that only authenticated users could access certain features and that authorized users had appropriate levels of access.
- ***Customization***: Customized the authentication system to fit the specific requirements of the project, such as extending the user model with additional fields or customizing the login and registration forms to include extra information. This allowed for a tailored user experience while maintaining the security and integrity of the authentication process.

3.5 Version Control:

- ***Git:*** Git is employed as the version control system for managing and tracking changes to the source code of YUMHUB. It facilitates collaboration among team members, allows for branching and merging of code, and ensures the integrity and reliability of the project's codebase.

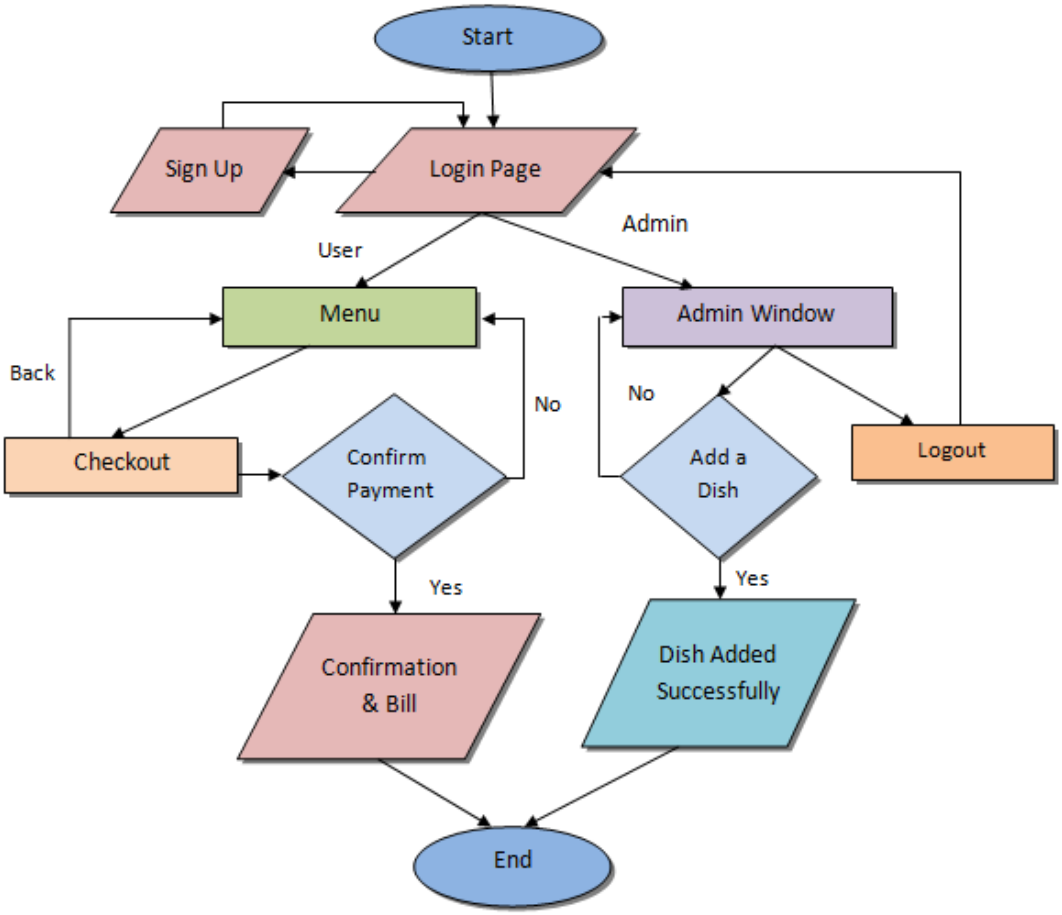
3.6 Development Environment:

- ***Visual Studio Code***: Visual Studio Code serves as the primary integrated development environment (IDE) for YUMHUB development. It offers a lightweight yet powerful code editor with features such as syntax highlighting, code completion, and debugging support, enhancing productivity and code quality.

In developing YUMHUB, a comprehensive range of tools and technologies were employed to ensure robustness, scalability, and efficiency. The project leveraged Django, a high-level Python web framework, for its backend development, along with its integrated SQLite database management system. Frontend development was facilitated by HTML, CSS, and JavaScript, which provided the foundation for user interface design and interactivity. Other essential tools and technologies included Django-User for user authentication, as well as various Python libraries and packages for additional functionalities. Through the strategic selection and integration of these tools and technologies, YUMHUB was able to achieve its objectives of enhancing the profitability and efficiency of small food vendors in a competitive market landscape.

4. Flowchart

YUMHUB follows the beneath flow for seamless food ordering for customers and proper administration of the dishes and orders by the admin:



4.1 Flowchart

5. Code (Python File Only)

Django Project is developed using two directories, i.e, Project and App. The provided code will be divided into two sections one for files in app directory named YumHub, and then in the project directory named Project.

YumHub (App directory)

- **urls.py**

```
from django.contrib import admin
from django.urls import path,include
from . import views

urlpatterns = [
    path("",views.index,name='index'),
    path("login/",views.login,name='login'),
    path("signup/",views.signup,name='signup'),
    path("menu/",views.menu,name='menu'),
    path("checkout/",views.checkout,name='checkout'),
    path("payment/",views.payment,name='payment'),
    path("bill/",views.bill,name='bill'),
   path("admin_dashboard/",views.admin_dashboard,name='admin_dashbo
   ard'),
    path("logout/",views.logout_view,name="logout")
]
```

- **views.py**

```
from django.shortcuts import *
from .models import *
from django.http import *
from django.contrib.auth import authenticate, login as auth_login,
logout
import json
import ast
from decimal import Decimal
from .backends import ProfileBackend
from django.contrib.auth.decorators import login_required
import base64
from django.contrib import messages

# Create your views here.
def index(request):
    return render(request,'index.html')
```

```
def login(request):
    if request.method == 'POST':
        username = request.POST.get('username')
        password = request.POST.get('password')
        user_type = request.POST.get('user_type')

        user = authenticate(request, username=username,
password=password)
        if user is not None:
            profile = Profile.objects.get(user=user)
            if profile.user_type == 'admin':
                auth_login(request, user)
                return redirect('admin_dashboard')
            else:
                auth_login(request, user)
                return redirect('menu')

    return render(request, 'login.html')

def signup(request):
    if request.method == 'POST':
        name = request.POST.get('name')
        phone = request.POST.get('phone')
        username = request.POST.get('username')
        password = request.POST.get('password')
        user_type = request.POST.get('user_type')

        # Create a new User instance
        user = User.objects.create_user(username=username,
password=password)

        # Create a new Profile instance and link it to the user
        profile = Profile.objects.create(name=name, phone=phone,
user_type=user_type, username=username, user=user)

        return redirect('login')

    return render(request, 'signup.html')

@login_required
def menu(request):
    dishes = Dish.objects.all()
    for dish in dishes:
```

```
        with open(dish.image.path, "rb") as img_file:
            dish.image_data =
base64.b64encode(img_file.read()).decode('utf-8')
    return render(request, 'menu.html', {'dishes': dishes})

@login_required
def checkout(request):
    if request.method == 'POST':
        total_orders = {}
        total_amount = 0
        dishes = Dish.objects.all()

        for dish in dishes:
            quantity = int(request.POST.get(f'dish_{dish.id}'))
            if quantity > 0:
                total_orders[dish.id] = quantity
                total_amount += quantity * dish.price

        processed_orders = []
        for dish_id, quantity in total_orders.items():
            dish = Dish.objects.get(pk=dish_id)
            processed_orders.append({'name': dish.name, 'price':
dish.price, 'quantity': quantity})

        return render(request, 'checkout.html', {'total_orders':
total_orders, 'total_amount': total_amount, 'processed_orders':
processed_orders})

    return redirect('menu')


@login_required
def payment(request):
    if request.method == 'POST':
        total_amount = request.POST.get('total_amount')
        processed_orders =
request.POST.getlist('processed_orders[]')
        tax = float(total_amount)*0.10
        total=float(total_amount)+tax

        return render(request, 'payment.html', {'total_amount':
total_amount, 'processed_orders': processed_orders,'total':total})

    total_amount=None
```

```
    processed_orders=[]
    total = 0
    return render(request, 'payment.html', {'total_amount':
total_amount,'processed_orders':processed_orders,'total':total})


    return redirect('menu')

@login_required
def bill(request):
    if request.method == 'POST':
        total_amount = request.POST.get('total_amount')
        processed_orders =
request.POST.getlist('processed_orders[]')
        tax = float(total_amount)*0.10
        total=float(total_amount)+tax
        # for order in processed_orders:
        #     o_total=order.price*order.quantity
        #     order.append(o_total)
        def preprocess_order_string(order_str):
            return order_str.replace("Decimal(", "").replace(")",
"")

        processed_orders_dicts = []
        for order_str in processed_orders:
            order_str = preprocess_order_string(order_str)
            order_dict = ast.literal_eval(order_str)

order_dict['o_total']=float(order_dict['quantity']*float(order_dict[
'price']))
            order_dict['price'] = Decimal(order_dict['price'])
            processed_orders_dicts.append(order_dict)

        return render(request, 'bill.html', {'total_amount':
total_amount, 'processed_orders':
processed_orders_dicts,'tax':tax,'total':total})

    total_amount=None
    processed_orders=[]
    tax=None
    total = None
    return render(request, 'bill.html', {'total_amount':
total_amount, 'processed_orders':
processed_orders,'tax':tax,'total':total})
```

```
def logout_view(request):
    logout(request)
    return redirect('login')

@login_required
def admin_dashboard(request):
    if request.method == 'POST':
        dish_name = request.POST.get('dish_name')
        dish_description = request.POST.get('dish_description')
        dish_image = request.FILES.get('dish_image')
        dish_price = request.POST.get('dish_price')

        new_dish = Dish(name=dish_name, image=dish_image,
price=dish_price, quantity = 0)
        new_dish.save()

        messages.success(request, 'Dish added successfully!')

        return redirect('admin_dashboard')

    return render(request, 'admin_dashboard.html')
```

- **models.py**

```
from django.db import models
from django.contrib.auth.models import User
from django.contrib.auth.hashers import make_password, check_password
from django.utils import timezone

from django.contrib.auth.models import User
from django.db import models

class Profile(models.Model):
    USER_TYPE_CHOICES = [
        ('user', 'User'),
        ('admin', 'Admin'),
    ]

    name = models.CharField(max_length=100)
    phone = models.CharField(max_length=15)
    username = models.CharField(unique=True, max_length=50)
    user_type = models.CharField(choices=USER_TYPE_CHOICES,
default='user', max_length=10)
    user = models.OneToOneField(User, on_delete=models.CASCADE,
default=None, null=True)
```

```
    def __str__(self):
        return self.username

class Dish(models.Model):
    id = models.AutoField(primary_key=True)
    name = models.CharField(max_length=100)
    image = models.ImageField(upload_to='dishes')
    price = models.DecimalField(max_digits=6, decimal_places=2)
    quantity = models.PositiveIntegerField()

    def __str__(self):
        return self.name


class Order(models.Model):
    id = models.AutoField(primary_key=True)
    user = models.ForeignKey(User, on_delete=models.CASCADE
    status = models.CharField(max_length=50, choices=(('placed',
'Placed'), ('cancelled', 'Cancelled'), ('completed', 'Completed')),
default='placed')
    created_at = models.DateTimeField(auto_now_add=True)


class OrderItem(models.Model):
    order = models.ForeignKey(Order, on_delete=models.CASCADE)  # Order
this item belongs to
    dish = models.ForeignKey(Dish, on_delete=models.CASCADE)  # Dish in
the order
    quantity = models.PositiveIntegerField()
```

Project (Project directory)

- **urls.py**

```
from django.contrib import admin
from django.urls import path,include
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    path("admin/", admin.site.urls),
    path("",include('YumHub.urls')),

] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

- **settings.py**

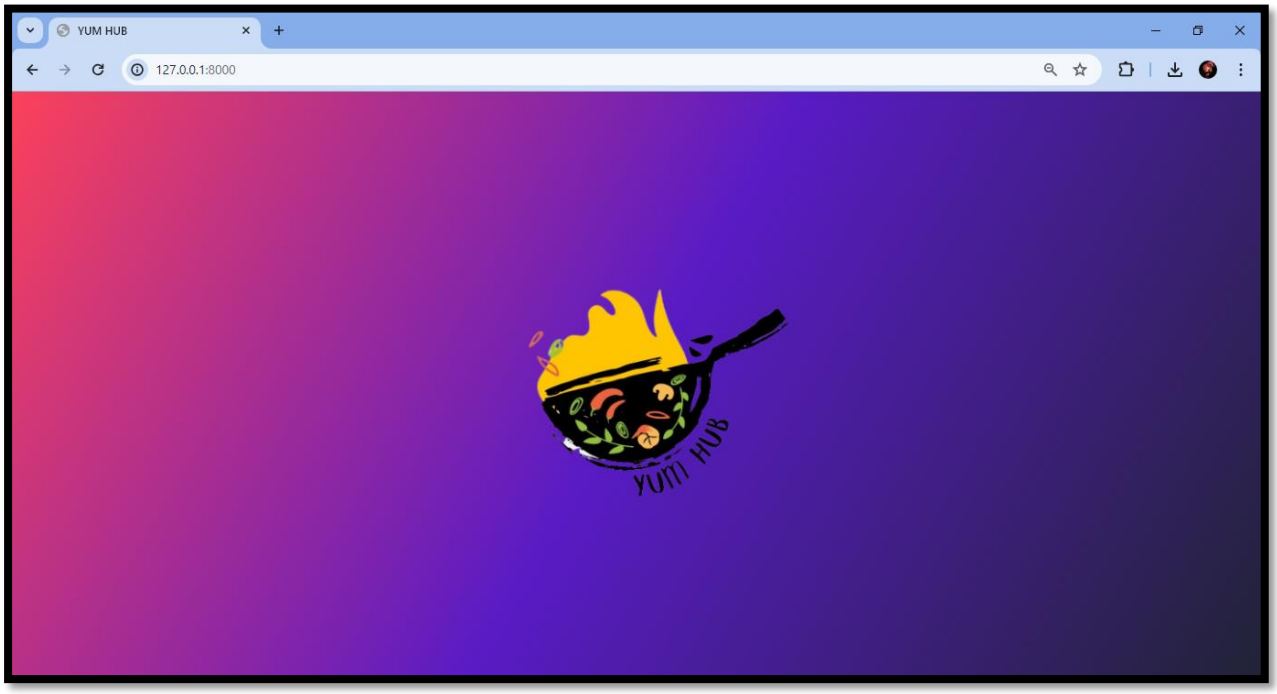
```
from pathlib import Path
import os

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent


# Quick-start development settings - unsuitable for production
# See
https://docs.djangoproject.com/en/5.0/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = "django-insecure-
mv1n*i_#v1bu1c49n%gt+9kzoua_%f_pzehej7odm=68y3%3b2"

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = ['127.0.0.1']


# Application definition

INSTALLED_APPS = [
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    "django.contrib.staticfiles",
    "YumHub"
]

MIDDLEWARE = [
    "django.middleware.security.SecurityMiddleware",
    "django.contrib.sessions.middleware.SessionMiddleware",
    "django.middleware.common.CommonMiddleware",
    "django.middleware.csrf.CsrfViewMiddleware",
    "django.contrib.auth.middleware.AuthenticationMiddleware",
    "django.contrib.messages.middleware.MessageMiddleware",
    "django.middleware.clickjacking.XFrameOptionsMiddleware",
]
```

```
ROOT_URLCONF = "Project.urls"

AUTHENTICATION_BACKENDS = [
    # 'YumHub.backends.ProfileBackend',  # Add your custom backend
here
    'django.contrib.auth.backends.ModelBackend',  # Default Django
authentication backend
    # Other authentication backends if needed
]

TEMPLATES = [
    {
        "BACKEND":
"django.template.backends.django.DjangoTemplates",
        "DIRS": [os.path.join(BASE_DIR, "templates")],
        "APP_DIRS": True,
        "OPTIONS": {
            "context_processors": [
                "django.template.context_processors.debug",
                "django.template.context_processors.request",
                "django.contrib.auth.context_processors.auth",

"django.contrib.messages.context_processors.messages",
                "django.template.context_processors.media",
            ],
        },
    },
]

WSGI_APPLICATION = "Project.wsgi.application"


# Database
# https://docs.djangoproject.com/en/5.0/ref/settings/#databases

DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.sqlite3",
        "NAME": BASE_DIR / "db.sqlite3",
    }
}
```

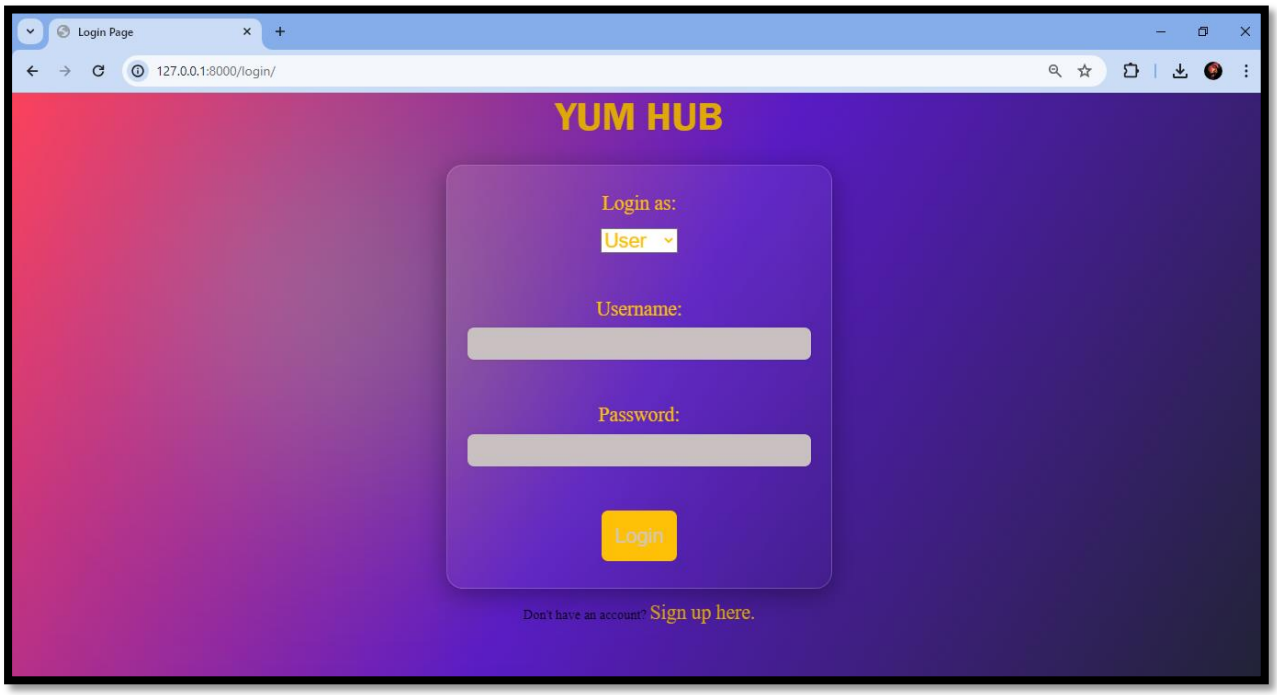
```
# Password validation
# https://docs.djangoproject.com/en/5.0/ref/settings/#auth-
password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        "NAME":
"django.contrib.auth.password_validation.UserAttributeSimilarityVa
lidator",
    },
    {"NAME":
"django.contrib.auth.password_validation.MinimumLengthValidator",}
,
    {"NAME":
"django.contrib.auth.password_validation.CommonPasswordValidator",
},
    {"NAME":
"django.contrib.auth.password_validation.NumericPasswordValidator"
,},
]


# Internationalization
# https://docs.djangoproject.com/en/5.0/topics/i18n/

LANGUAGE_CODE = "en-us"

TIME_ZONE = "UTC"

USE_I18N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/5.0/howto/static-files/

STATIC_URL = "static/"

STATICFILES_DIRS=[
    os.path.join(BASE_DIR,"static")
]

# Default primary key field type
```

```
# https://docs.djangoproject.com/en/5.0/ref/settings/#default-
auto-field

DEFAULT_AUTO_FIELD = "django.db.models.BigAutoField"

MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
MEDIA_URL = '/media/'
```

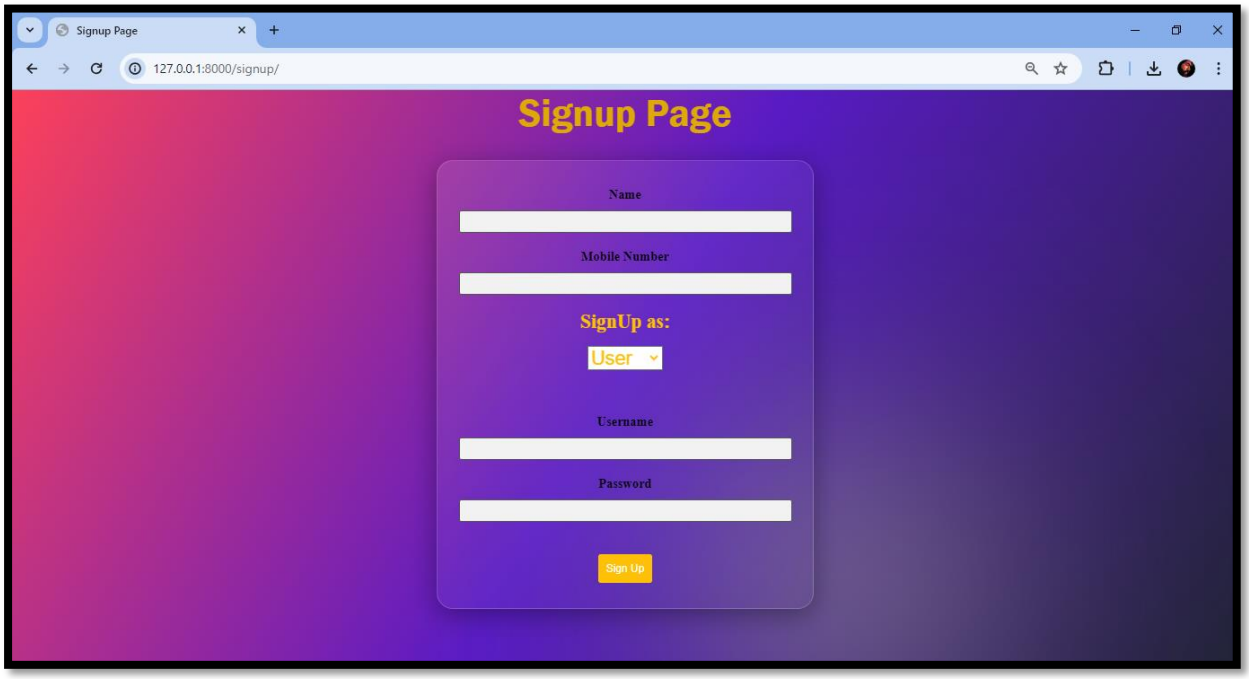
6. Major Findings/Outcomes/Output/Results



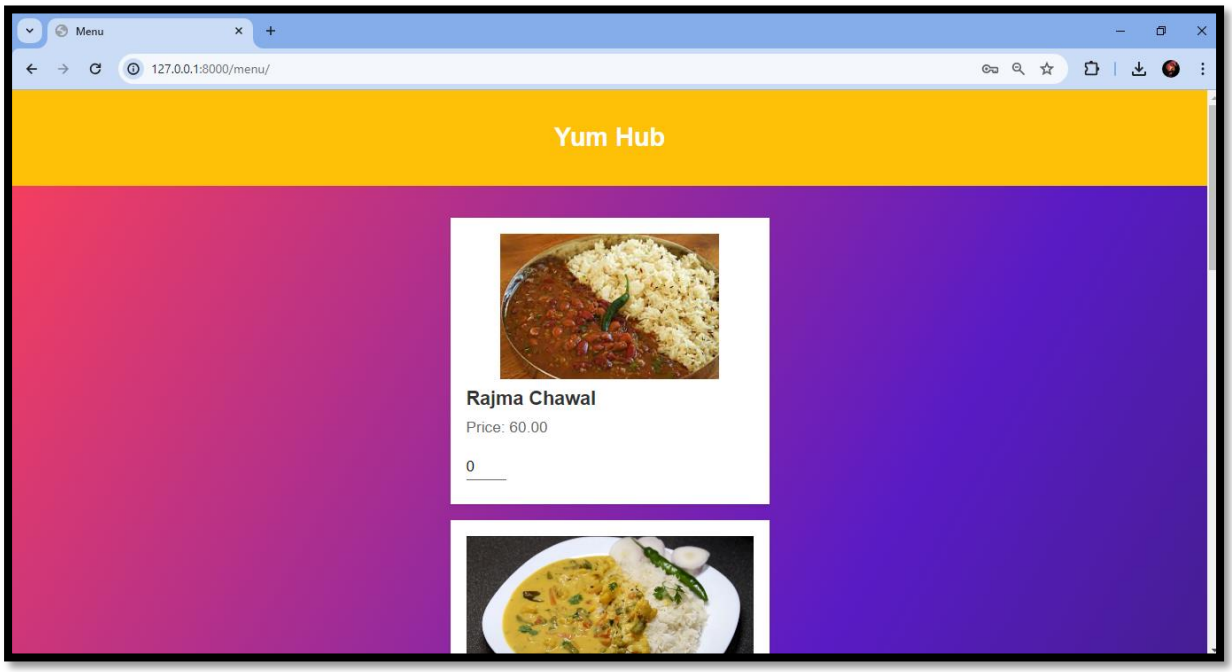
6.1 Logo Page



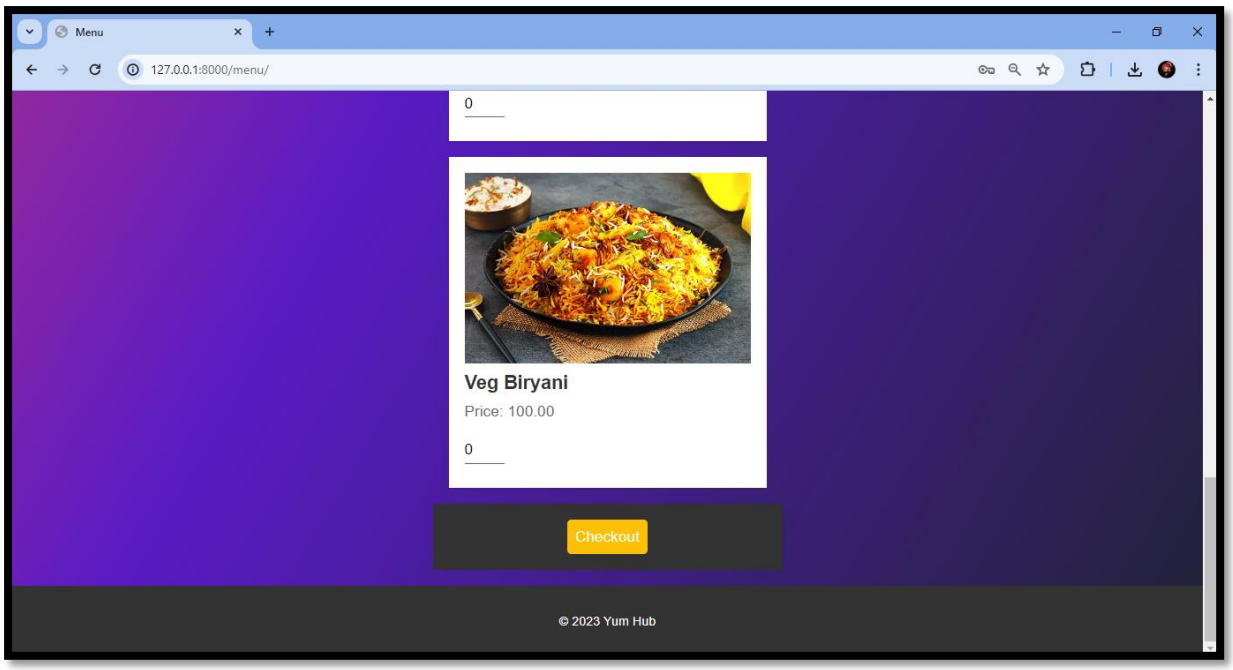
6.2 Login Page



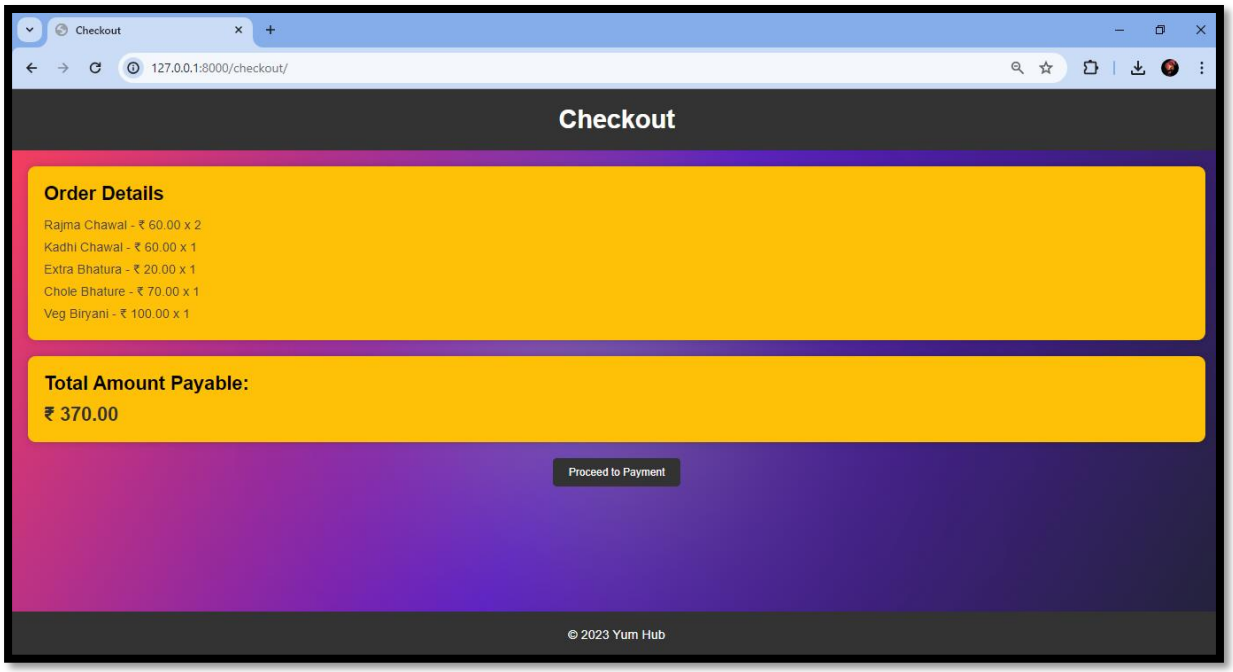
6.3 Signup Page



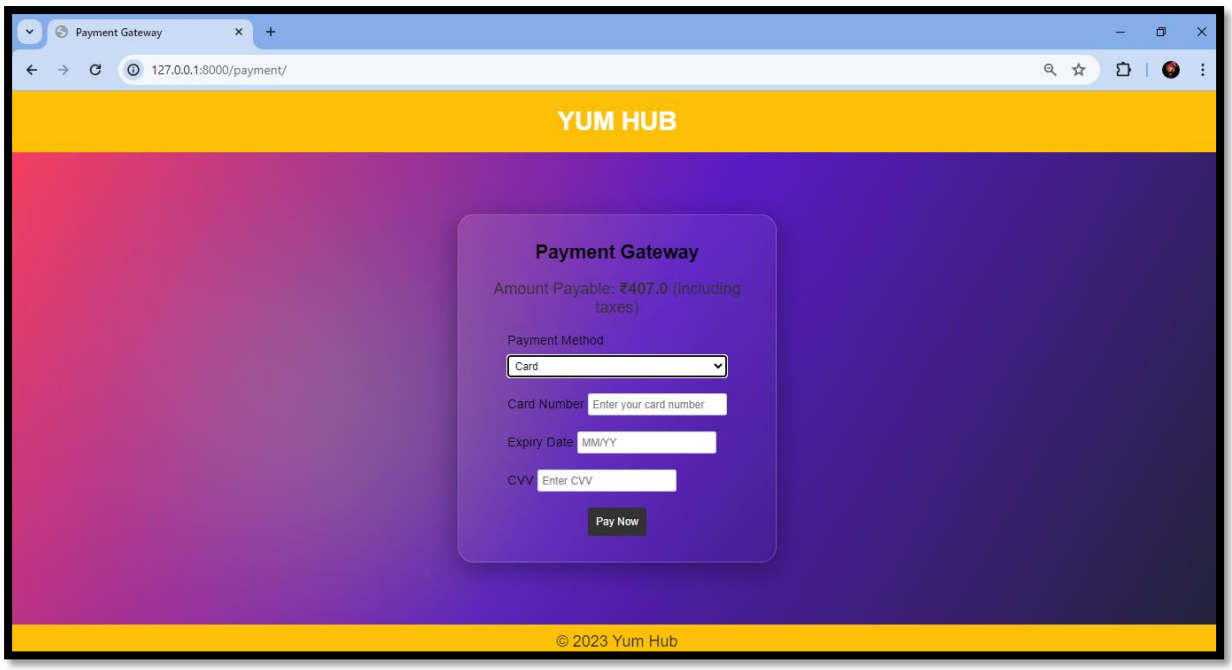
6.4 Menu Page (1)



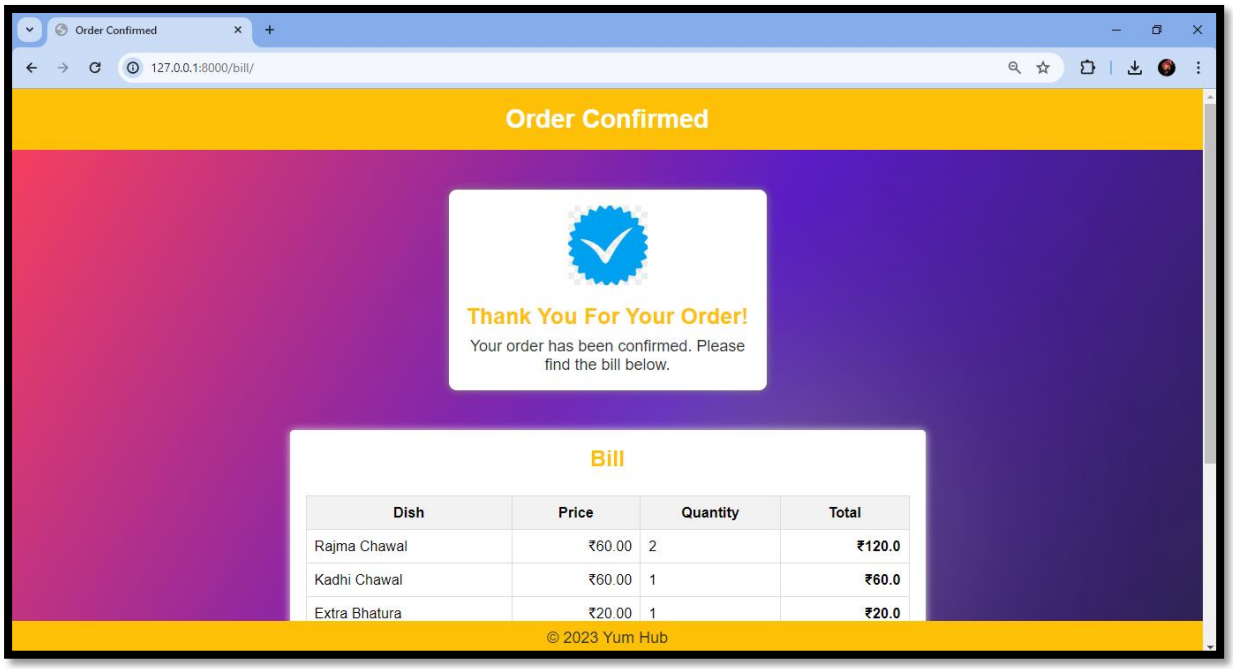
6.5 Menu Page (2)



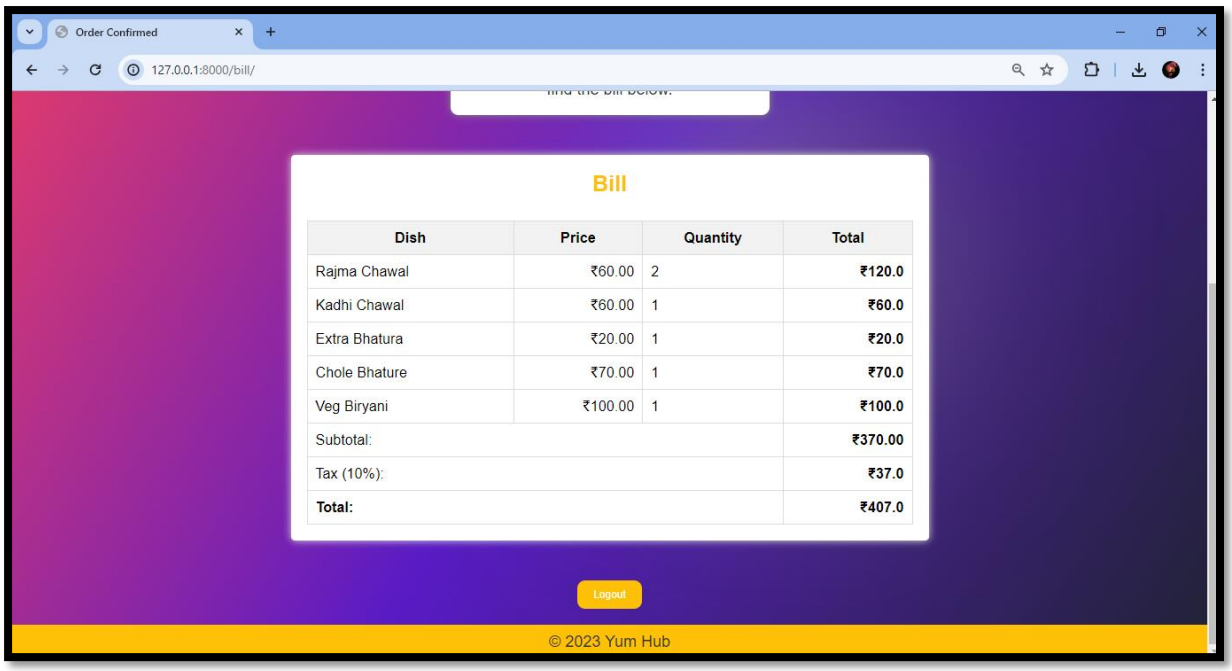
6.6 Checkout Page



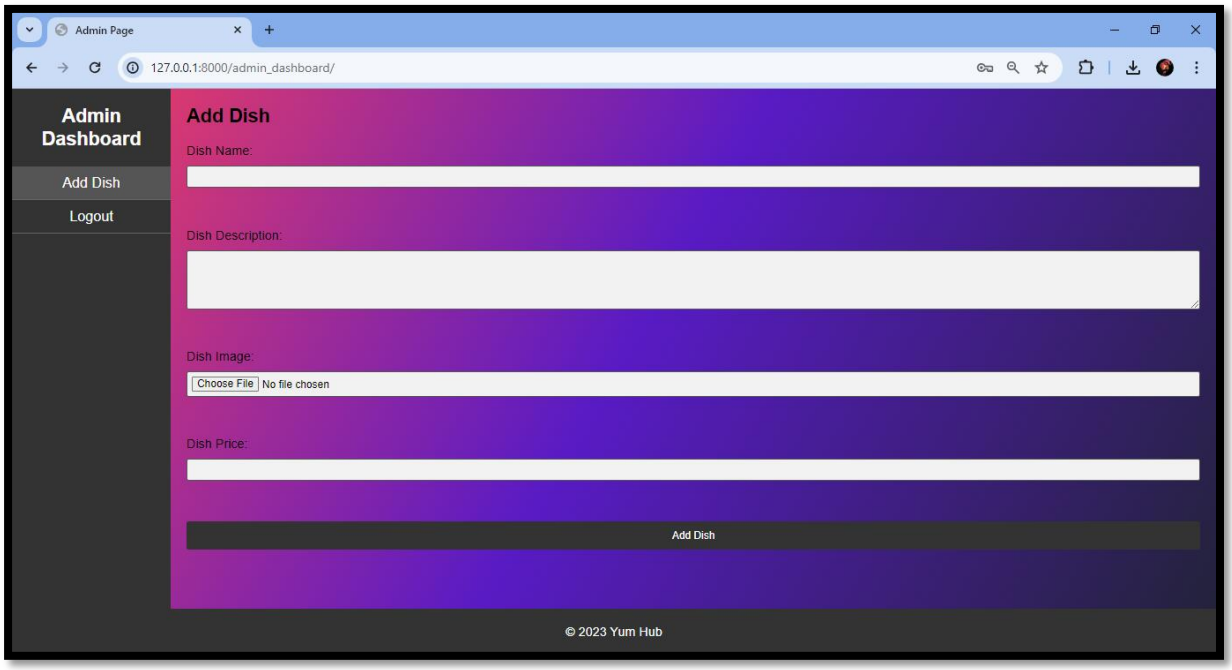
6.7 Payment Window



6.8 Order Confirmation



6.9 Order Bill



6.10 Add a Dish (Admin Dashboard)

7. Conclusion and Future Scope

In conclusion, YUMHUB: A Food Stall Application represents a significant milestone in empowering small-scale food vendors to thrive in a competitive market environment. Through the implementation of a robust web-based solution, the project has successfully addressed key challenges faced by vendors, enabling them to enhance profitability and efficiency in their operations. By leveraging Django, SQLite, and Django Rest Framework, along with other essential tools and technologies, YUMHUB provides a user-friendly platform for managing menus, processing orders, and engaging with customers.

While the current version of the application offers comprehensive features for ordering and managing food dishes, there are several avenues for future enhancement and expansion:

- **Timed Order Tracking:** Implement a feature to track orders over a specific time period, enabling vendors to analyze sales trends and performance of each dish. Provide insights into popular items, peak ordering hours, and seasonal variations to optimize menu offerings. Allow vendors to make data-driven decisions and adjust their strategies to meet customer demands effectively.
- **Mobile Application**: Developing a dedicated mobile application for the food management system would extend its accessibility and usability, allowing users to make and track orders on the go.
- **Integration of AI Technologies:** Explore the use of artificial intelligence algorithms to provide predictive insights on orders and customer behavior. Leverage machine learning models to forecast future trends, suggest menu optimizations, and automate aspects of order processing. Enhance the platform's capabilities with AI-driven analytics tools to empower vendors with actionable insights and recommendations.
- **Integration of Payment APIs:** Integrate payment APIs and methods to facilitate seamless transactions between vendors and customers. Offer multiple payment options, including credit/debit cards, digital wallets, and online banking, to enhance convenience and accessibility. Ensure secure and reliable payment processing to instill trust and confidence among users.
- **Scalability and Outreach:** Expand the reach of YUMHUB to onboard more food vendors and cater to a broader audience. Collaborate with government agencies and regulatory bodies to obtain necessary authorizations and certifications for operating the platform. Continuously optimize and refine the platform's features and functionalities to accommodate growing user demands and ensure scalability.

Ultimately, the goal of YUMHUB is to continue scaling its impact, reaching more vendors, and empowering them to succeed in the food industry. By embracing innovation and leveraging technology, YUMHUB aims to revolutionize the way small food vendors operate, ensuring their long-term sustainability and growth.

8. References

YUMHUB draws upon several technologies and libraries to achieve its functionalities. Below are the key references for the tools and technologies used in the project:

- **Django Documentation:** Official documentation for the Django web framework.
  Link: https://docs.djangoproject.com/en/5.0/

- **Django-Admin Documentation:** Documentation for Django-Admin, a Django extension that provides user session management.
  Link: https://docs.djangoproject.com/en/5.0/topics/auth/

- **Python Documentation:** Official documentation for the Python programming language.
  Link: https://docs.python.org/3/

- **HTML, CSS, and JavaScript Documentation:** W3Schools provides comprehensive documentation and tutorials for web technologies.
  Link: https://www.w3schools.com/

- **SQLite Documentation:** Official documentation for SQLite, the database engine used in the project.
  Link: https://www.sqlite.org/docs.html

These references serve as invaluable resources for understanding and implementing the core functionalities of YUMHUB. They provide guidance on leveraging the capabilities of Django, SQLite to develop a robust food stall web application.