



## **WebX Lab Mini Project**

### **Restaurant Ordering Web Application with Flask and MongoDB**

**Submitted By :**

Name: **Himesh Pathai**

Roll No: **34**

Division: **D15A**

Third Year (Semester-VI)

Bachelor of Engineering in Information Technology  
(Autonomous Program)

Department of Information Technology

Vivekanand Education Society's Institute of Technology

An Autonomous Institute Affiliated to University of  
Mumbai

Academic Year: 2023–2024

## **Certificate**

This is to certify that I have completed the project report on the topic **Simple Blog Web Application using Flask and MongoDB** satisfactorily in partial fulfilment of the requirements for the award of **Mini Project in WebX** Lab of Third Year (Semester-VI) in Information Technology under the guidance of **Ms. Dipti Karani** during the academic year 2023–2024, as prescribed by An Autonomous Institute Affiliated to University of Mumbai.

**Supervisor/ Examiner**

## **Table of Contents**

1. Problem Statement
2. Introduction
3. Methodology
4. Objective
5. Tools and Technologies Used
6. Features of the Project
7. Folder Structure of the Project
8. Working of the Project
9. Screenshots
10. Sample Code Snippets
11. Challenges Faced
12. Conclusion
13. Future Improvements
14. References
15. GitHub Link

# 1. Problem Statement

Create a simple and efficient blog web application where users can read, search, and manage blog posts, while providing a secure admin panel for authenticated users to create, update, and delete posts. The system should store data in a NoSQL database (MongoDB) and handle user authentication securely.

## 2. Introduction

This project is a web-based blogging platform developed using **Flask** (a Python web framework) and **MongoDB** (a NoSQL database). It allows users to read and search blog posts, and lets admin users create, update, or delete content. This system works like a mini **Content Management System (CMS)** where only authenticated users can manage blog content. The design is simple, responsive, and beginner-friendly.

## 3. Methodology :

1. **Flask Setup:** Use Flask to handle routing, rendering templates, and managing requests.
2. **MongoDB Integration:** Connect Flask to MongoDB using pymongo for storing user data and food posts.
3. **Authentication:** Implement user registration and login using sessions and password hashing.
4. **Admin Dashboard:** Only logged-in admins can access post management features (CRUD).
5. **Search Feature:** Allow users to search blog posts by keywords or titles.
6. **UI Design:** Build simple HTML templates with Bootstrap or similar for frontend pages.
7. **Testing & Debugging:** Test functionality and fix any errors to ensure smooth user experience.

## 4. Objective

The main goal of this project is to:

- Build a dynamic and easy-to-use blog system.
- Allow only registered users (admin) to create and manage blog posts.
- Store all data securely in MongoDB.
- Provide a search feature so users can find posts easily.
- Learn how to create real web apps using Flask.

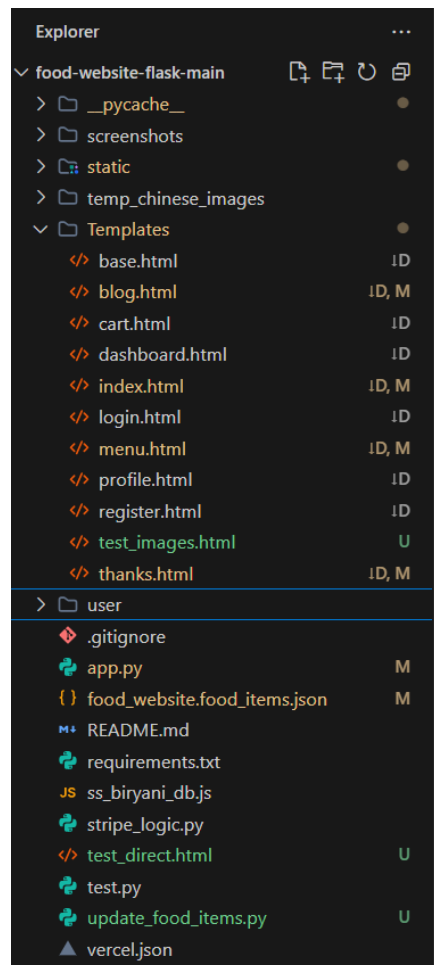
## 5. Tools and Technologies Used

Tools/Technology	Purpose
Python	Main Programming Language
Flask	Web framework to build the application
MongoDB	Database to store blog data
HTML/CSS	For designing the user interface
VS Code	Code editor
Jinja2	Templating engine in Flask

## 6. Features of the Project

Feature	Description
User Registration	New users can register on the platform.
Login System	Only registered users can log in.
Admin Dashboard	Logged-in users can manage blog posts.
Create/Edit/Delete Posts	Admin can add new blog posts or change existing ones.
Search Functionality	Users can search for blog posts by title.
Responsive UI	Clean and simple design

## 7. Folder Structure of the Project



## Explanation :

- `__pycache__/` – Stores compiled Python bytecode to improve performance.
  - `screenshots/` – Contains images used for documentation or demonstrations.
  - `static/` – Holds static assets such as CSS, JavaScript, and image files.
  - `Templates/` – Contains HTML templates used by Flask to render web pages (should be named templates for compatibility).
  - `user/` – Likely includes code related to user management such as authentication and profile handling.
- 

### HTML Templates (inside Templates/)

- `base.html` – The base template extended by other HTML pages for consistency in layout.
  - `blog.html` – Displays blog entries or updates.
  - `cart.html` – Shows the contents of a user's shopping cart.
  - `dashboard.html` – Provides a dashboard interface for users or administrators.
  - `index.html` – The landing page or homepage of the website.
  - `login.html` – Interface for users to log into their accounts.
  - `menu.html` – Displays the list of available food items.
  - `profile.html` – Shows the logged-in user's profile information.
  - `register.html` – Form for new users to create an account.
  - `thanks.html` – Acknowledgement page displayed after successful actions.
- 

### Python Files

- `app.py` – The main application file that initializes and runs the Flask server.
  - `stripe_logic.py` – Handles payment processing using the Stripe API.
  - `test.py` – Used to perform testing or debugging operations during development.
- 

### Other Files

- `.gitignore` – Specifies files and directories that should be ignored by Git version control.
- `food_website.food_items.json` – JSON file storing structured data about food items.
- `README.md` – Provides a summary and documentation for the project.
- `requirements.txt` – Lists all Python dependencies required to run the project.
- `ss_biryani_db.js` – JavaScript file used to seed or update data in the MongoDB database.
- `vercel.json` – Configuration file for deployment on the Vercel platform.

## 8. Working of the Project

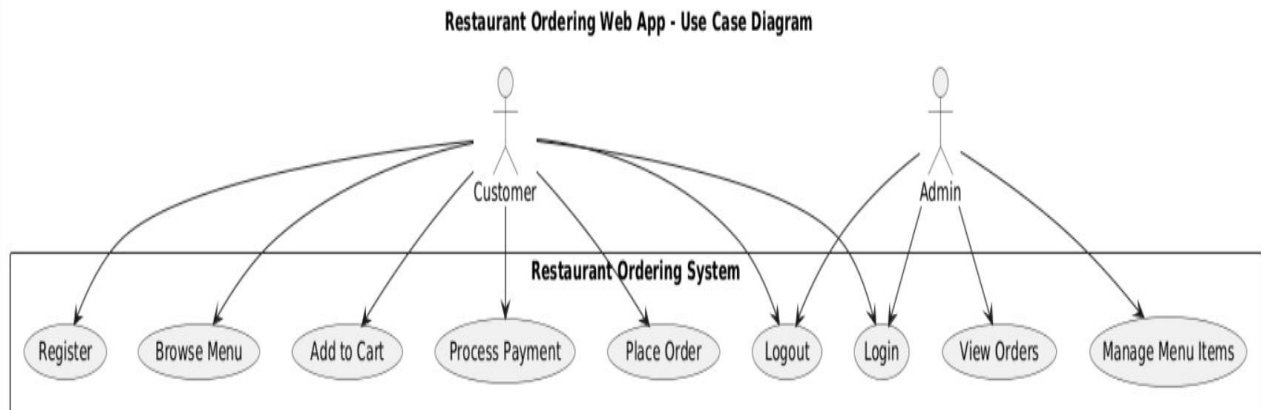
### 8.1 UML Diagrams

To understand the system architecture and flow, the following UML diagrams are used:

#### Use Case Diagram

The Use Case Diagram illustrates the interaction between the users (normal users and admin) and the application.

It highlights the actions like viewing, searching, creating, editing, and deleting blog posts based on the user's role.



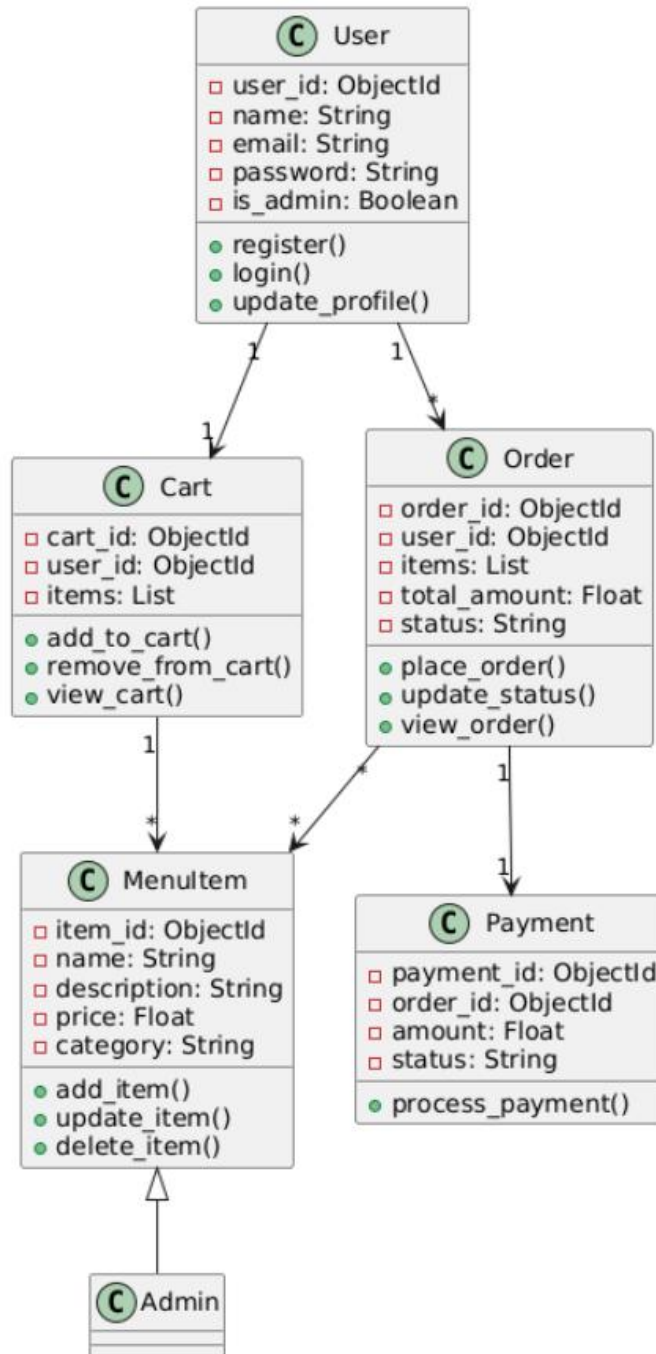
**Figure 1 – Use Case Diagram of the Blog Web Application**



# Class Diagram

The Class Diagram explains how different classes or entities in the application (like User and Post) are structured. It shows attributes and operations related to each class and how they are related to each other.

**Restaurant Ordering Web App - Class Diagram**



**Figure 2 – Class Diagram of the Blog Web Application**

## 7.2 Step-by-Step Working

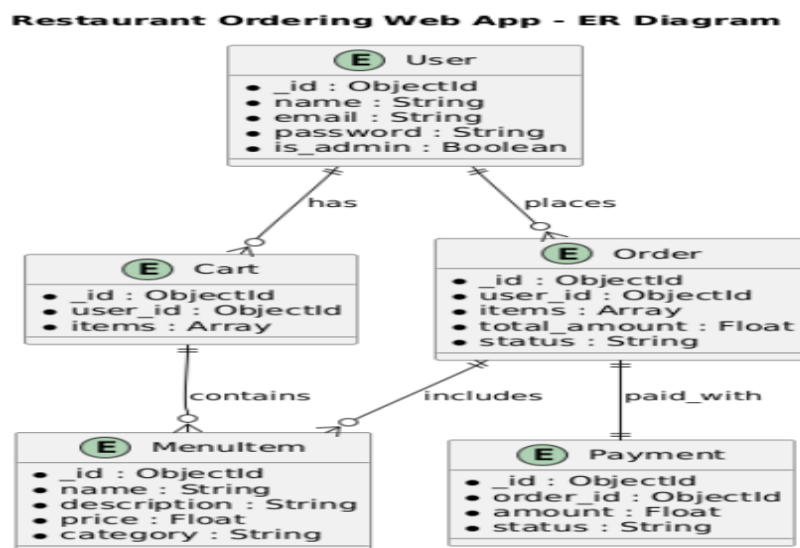
### Step-by-Step Working of the Restaurant Ordering Web Application

1. The user lands on the homepage and can browse available food items on the menu without logging in.
2. To place an order or access the dashboard, the user must register or log in.
3. After login, the user can add items to the cart and proceed to checkout.
4. Payment is securely handled through Stripe integration.
5. Upon successful payment, a confirmation page is displayed, and the order is recorded.
6. The user can view and update their profile, while admin users can manage food items and user data.
7. All user and order data is securely stored in MongoDB.

## 7.3 Database Representation Diagram

The below Entity-Relationship (ER) diagram represents the internal structure of the MongoDB database used in the Restaurant Ordering Web Application project. There are **three collections**:

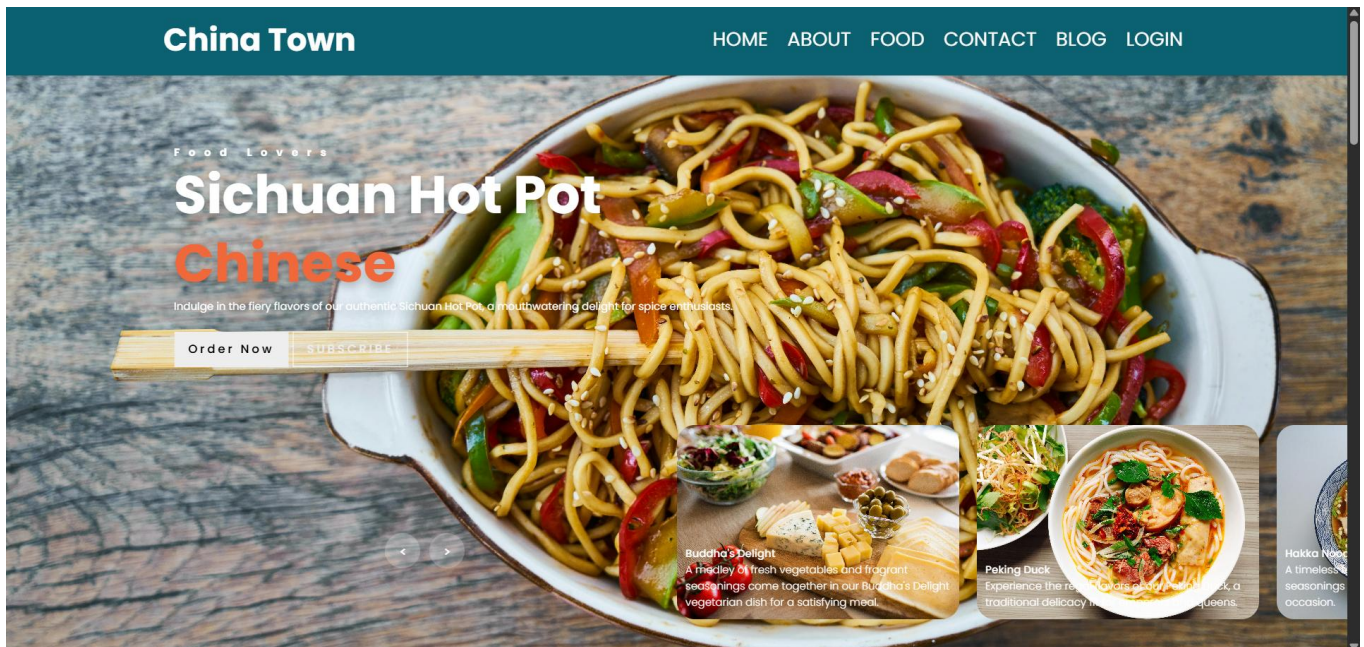
- **login** – stores user credentials including username and password.
- **pro** – stores blog post-related data such as name and content.
- **admin** – stores admin login details and other related information.



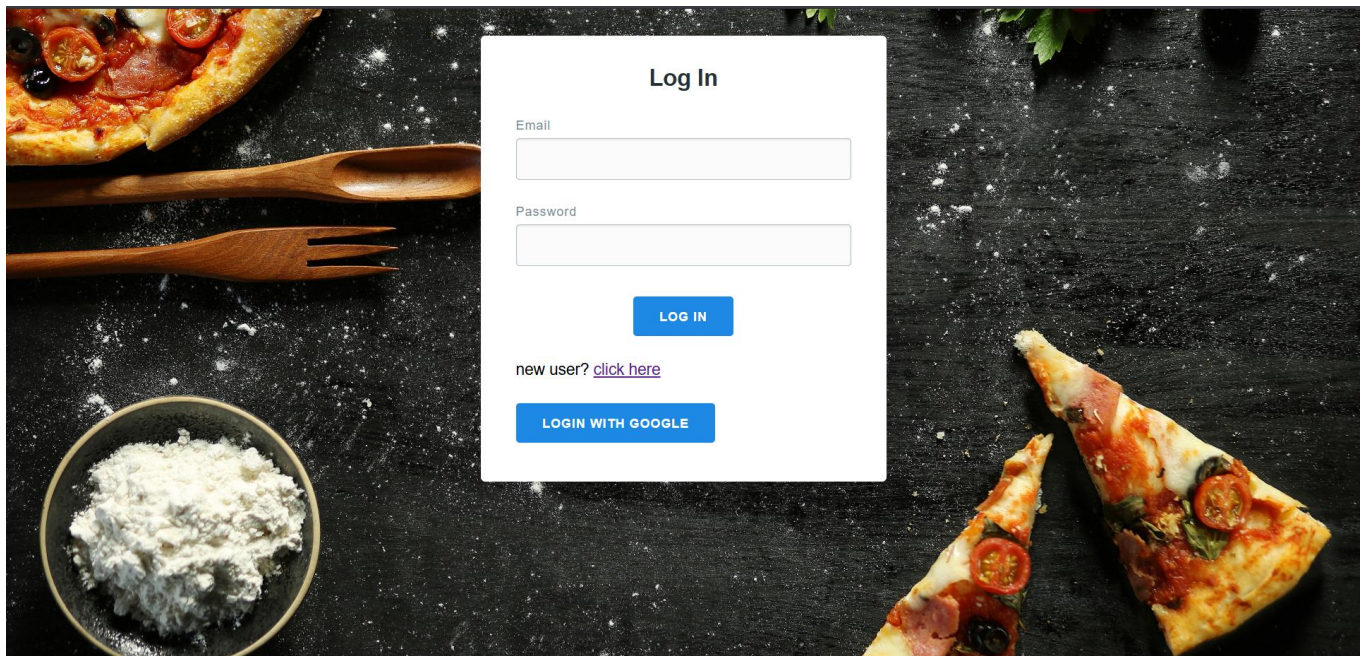
**This diagram helps visualize how the data is organized and stored in MongoDB.**

## 9. Screenshots:

1. Home Page : Displays all published blog posts and the food dishes without login in

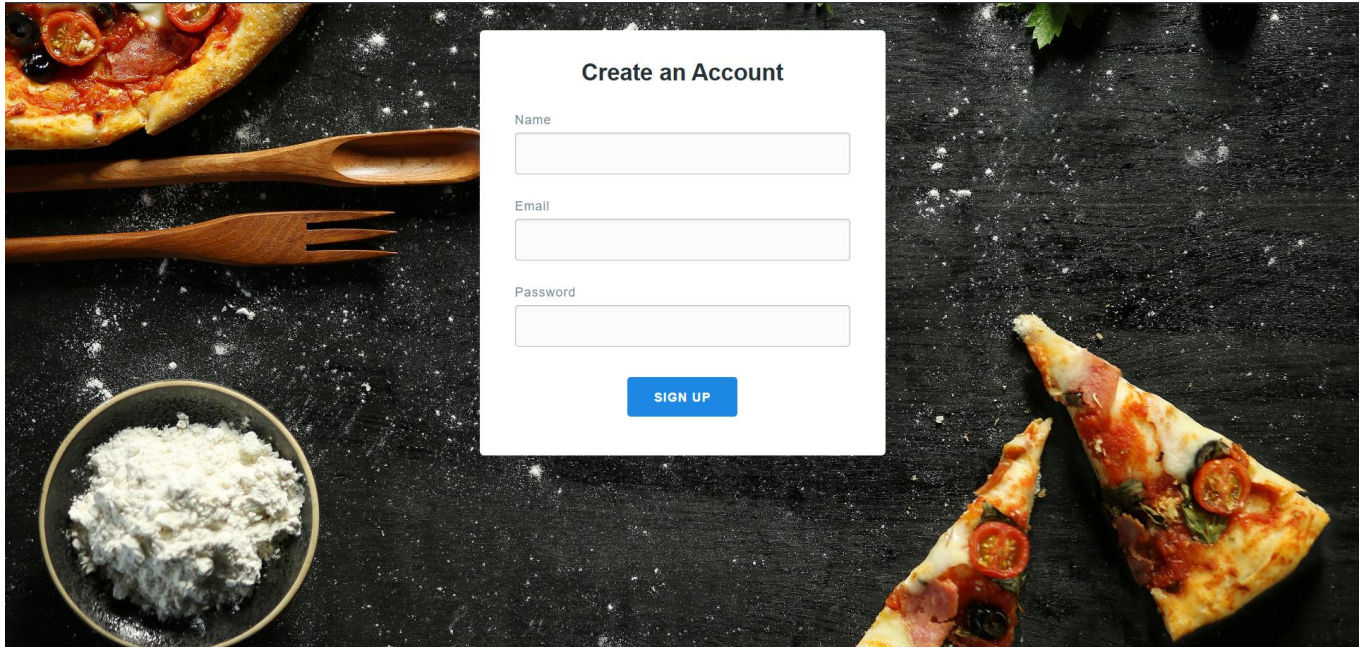


2. Login Page : Allows registered users to log in and access the admin dashboard with google login.





### 3. Signup Page : Enables new users to register.

A white rectangular form titled "Create an Account" is centered over a dark, textured background featuring a pizza, a wooden spoon, a wooden fork, and a bowl of white powder. The form contains three input fields labeled "Name", "Email", and "Password", followed by a blue "SIGN UP" button.

Create an Account

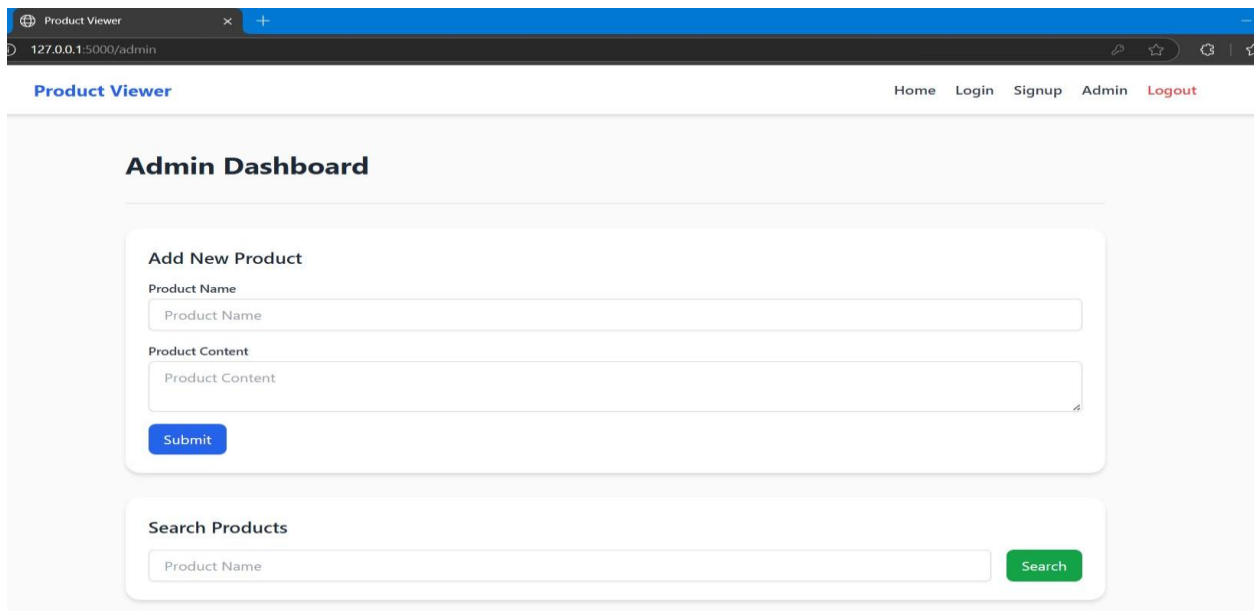
Name

Email

Password

SIGN UP

### 4. Admin Dashboard : Provides options to create, edit, and delete blog posts.

A screenshot of a web browser showing the "Admin Dashboard" of a "Product Viewer" application. The browser address bar shows "127.0.0.1:5000/admin". The dashboard has a navigation bar with links: Home, Login, Signup, Admin, and Logout. The main content area is titled "Admin Dashboard" and contains two sections: "Add New Product" with input fields for "Product Name" and "Product Content", a "Submit" button, and "Search Products" with a "Product Name" input field and a "Search" button.

Product Viewer

127.0.0.1:5000/admin

Product Viewer

Home Login Signup Admin Logout

### Admin Dashboard

#### Add New Product

Product Name

Product Content

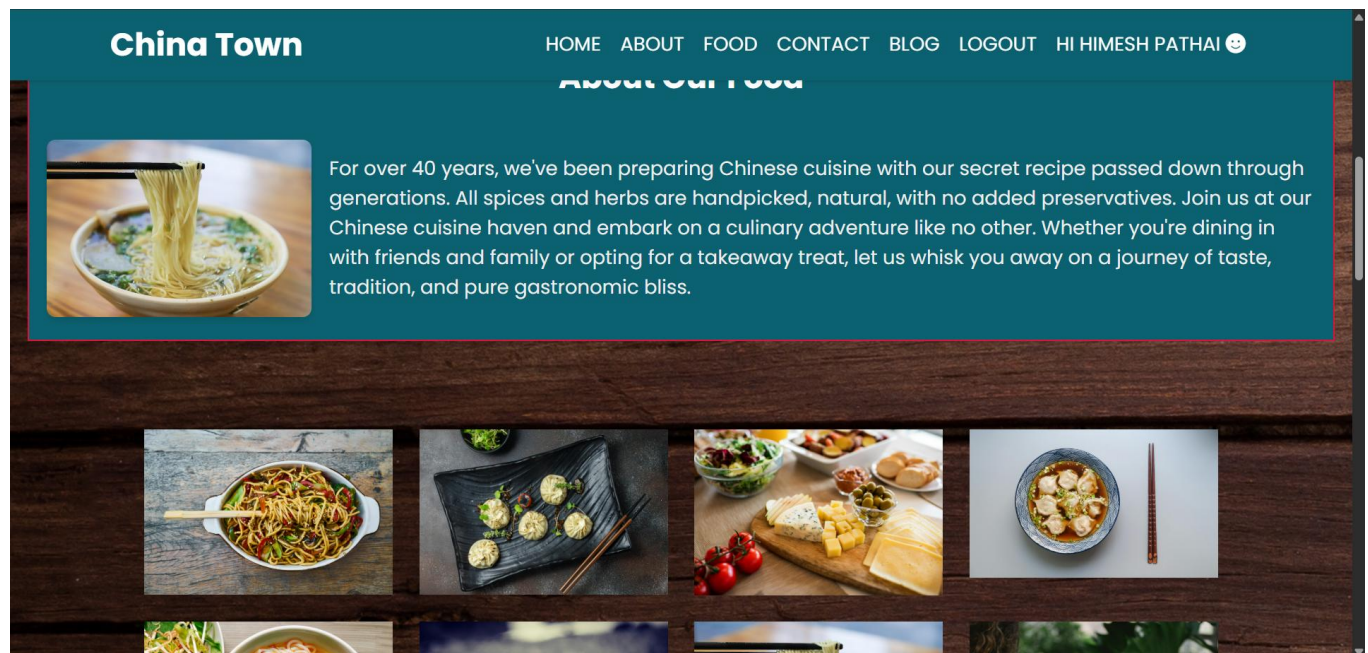
Submit

#### Search Products

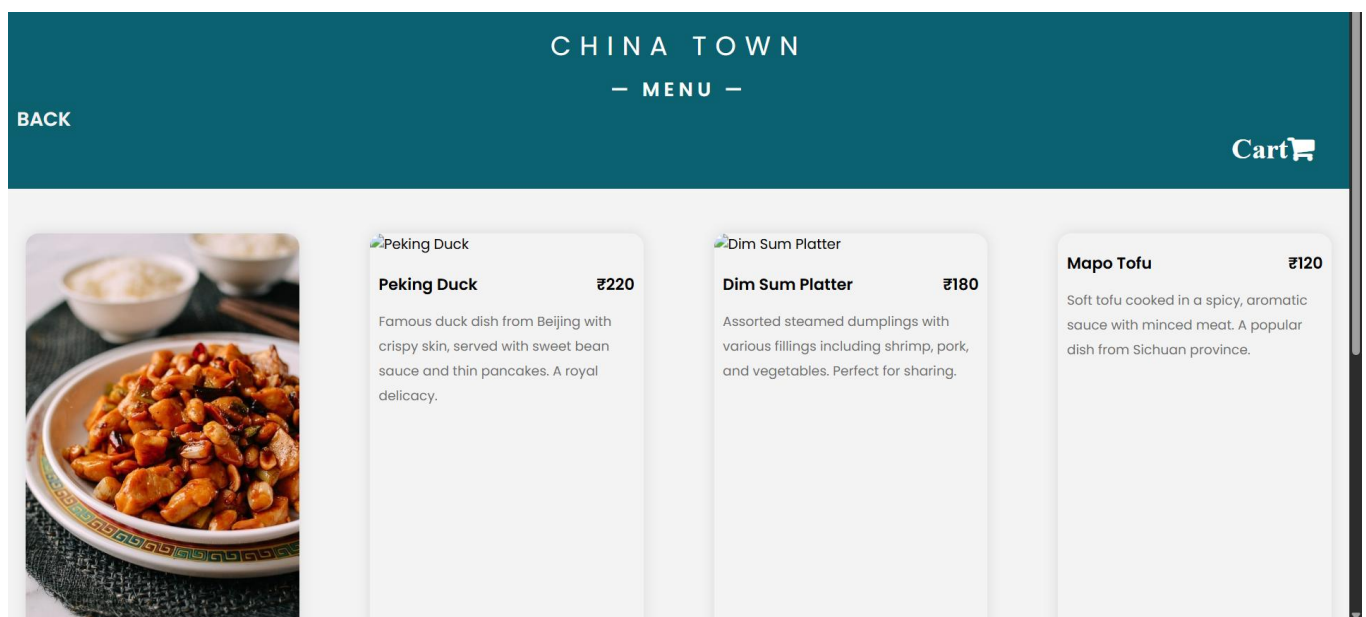
Product Name

Search

## 5. About Us Page



## 6. Food Menu Page – To order food and add to cart



## 10. Sample Code Snippets

**MongoDB Connection (credentials.py)** from  
pymongo import MongoClient  
client = MongoClient("mongodb://localhost:27017/") db =  
client['blog\_app']

### Flask Route for Creating Post (app.py)

```
@app.route('/create', methods=['GET', 'POST']) def  
create_post():    if request.method == 'POST':  
title = request.form['title']    content =  
request.form['content']  
        db.posts.insert_one({'title': title, 'content': content})  
return redirect(url_for('admin'))    return  
render_template('create_post.html')
```

## 11. Challenges Faced

- Connecting Flask to MongoDB using Pymongo for the first time.
- Managing session-based login system securely.
- Displaying dynamic content on HTML pages using Jinja templates.
- Handling CRUD operations correctly with form validations.

## 12. Conclusion

This project provided valuable experience in full-stack web development using Python. Through the development of a blog application, I gained practical knowledge in building dynamic web interfaces with Flask, integrating and managing data with MongoDB, and implementing user authentication to secure the admin dashboard. Overall, it strengthened my understanding of developing real-world web applications and the principles of secure and scalable design.

Through this project, I gained hands-on experience with:

- Flask routing and templates
- MongoDB document storage
- Session-based login and logout
- Full web development workflow

### **13. Future Improvements**

- Add a comments section under each blog post to allow user interaction.
- Implement image upload functionality to enrich blog content.
- Enhance the user interface using Bootstrap or Tailwind CSS for a modern design.
- Integrate email verification for new user registrations to improve account security.
- Use JWT (JSON Web Tokens) for secure and scalable authentication.

### **14. Reference:**

- [MongoDB Documentation](#)
- [W3Schools HTML/CSS Reference](#)

### **15. [GitHub Link](#)**