Infosys Springboard Internship 5.0

Project Report

“API Development of SweetSpot – Delivering Delight to Your Doorstep”

Submitted by

Avishkar Patil

Under the Guidance of

Mentor: Sri Lalitha
Coordinator: Udayakumar

Submitted to

Infosys Springboard Team

◆ Introduction:

In today's fast-paced and digitally connected world, **SweetSpot** is designed to align with modern lifestyles, offering a hassle-free, personalized approach to celebrating life's special moments. With increasingly busy schedules, many people find it difficult to make time for elaborate cake preparations or bakery visits. **SweetSpot** eliminates these challenges, providing an efficient online platform for browsing, customizing, and ordering cakes. With just a few clicks, customers can have their dream cakes delivered right to their doorstep, saving time and effort.

SweetSpot goes beyond convenience by empowering customers with extensive customization options and access to unique creations from skilled local bakers. It caters to specific dietary preferences, offering allergy-friendly and specialty cakes that traditional bakeries may not provide. By connecting customers to local artisans, **SweetSpot** fosters a sense of community while supporting small businesses.

The platform also addresses the evolving nature of social gatherings, enabling users to celebrate remotely with loved ones through surprise cake deliveries. Whether it's a virtual party or a spontaneous celebration, **SweetSpot** makes it easy to share joy and create lasting memories.

This innovative online cake delivery system benefits everyone involved: customers enjoy the convenience and personalization of online shopping, while local bakers expand their reach and showcase their creativity. **SweetSpot** supports a vibrant local economy by bridging the gap between consumers and artisans, encouraging growth and connection.

In an era where convenience and personalization are paramount, **SweetSpot** delivers a seamless experience for cake enthusiasts while fostering a sense of community. So, the next time a celebration calls for a delicious treat, skip the stress and let **SweetSpot** bring the joy of cake to your doorstep.

◆ Objectives:

1. **Simplify Online Cake Ordering**:
   Provide a seamless platform where users can easily browse, customize, and order cakes with just a few clicks

2. **Empower Customization**:
   Offer users a wide range of options to personalize cakes, including flavors, sizes, decorations, and messages, catering to diverse occasions and dietary needs.

3. **Enhance User Experience**:
   Develop a user-friendly interface with features like real-time delivery tracking, automated notifications, and secure payment integration.

4. **Support Local Bakers**:
   Create opportunities for small bakeries and local artisans to showcase their creativity and connect with a wider audience, fostering economic growth within communities.

5. **Promote Efficiency and Scalability**:
   Streamline operations such as inventory management and order processing to improve efficiency and enable the system to scale with increased demand.

6. **Encourage Inclusivity**:
   Cater to dietary preferences and restrictions, offering allergy-friendly and specialty cakes to ensure everyone can enjoy the experience.

7. **Build Strong Customer Relationships**:
   Foster customer satisfaction and loyalty by ensuring timely delivery, transparency, and a delightful ordering experience.

8. **Establish a Market Presence**:

   Position SweetSpot as a leading platform in the cake delivery industry, driven by innovation and a commitment to customer delight.

◆ Project Scope:

The **SweetSpot** project scope outlines the functionalities and limitations of the platform, setting clear boundaries for its development. The platform aims to provide a seamless experience for customers, allowing them to easily browse and customize cakes from local stores. Customers can register, log in, personalize their cakes with various flavors, sizes, and decorations, and manage their orders through a virtual shopping cart. Secure online payment processing and automated email notifications will keep them informed about their orders, while real-time tracking will provide peace of mind. For store owners, SweetSpot offers a management dashboard where they can upload cake details, manage customization options, and update order statuses to keep customers informed. Admins will have the ability to manage user accounts and perform CRUD operations for stores, cakes, and orders. The platform may also provide access to analytics for better decision-making.

While SweetSpot includes essential features like cake customization, secure payments, and order management, some advanced features are excluded from the initial scope. These include highly intricate cake designs requiring extensive communication between customers and stores, third-party delivery service integrations, and complex search filters or recommendation algorithms. Additionally, limitations like region-specific payment gateway support and scalability constraints will be addressed in future updates as the platform grows and evolves. The project’s initial focus is on delivering core functionalities, with plans for future enhancements based on user feedback and business needs.

◆ Requirements:

➢ Functional Requirements

For Customers:

1. **User Registration and Login:** Secure account creation and login functionality.
2. **Cake Browsing:** Search and filter cakes by categories or dietary preferences
3. **Cake Customization:** Personalize cakes with Flavors, sizes, toppings, and inscriptions.
4. **Shopping Cart Management:** Add, remove, or modify items in a virtual cart before checkout.
5. **Secure Online Payment:** Process payments safely via a secure payment gateway.
6. **Automated Notifications:** Send email updates for order confirmation, payment, and delivery status.
7. **Order Tracking:** Enable customers to monitor order status (e.g., "placed," "in progress," "delivered").

For Store Owners:

1. **Store Registration:** Create and manage store profiles with essential details.
2. **Cake Information Upload:** Add cake descriptions, images, prices, and customization options.
3. **Customization Options Management:** Configure available customization options.
4. **Order Management Dashboard:** Review, accept, and update customer orders through a centralized interface.
5. **Order Status Updates:** Notify customers of their order progress (e.g., "baking," "out for delivery").

For Admins:

1. **Comprehensive User Management:** Manage accounts for customers and stores
2. **CRUD Operations:** Perform Create, Read, Update, and Delete operations for stores, cakes, and customer orders.
3. **Analytics Access:** View platform metrics, such as sales trends and user demographics, to guide decision-making.

➢ Non-Functional Requirements

1. **Performance:** Pages must load quickly and respond efficiently to user actions.
2. **Scalability:** The system should handle growing numbers of users and orders without performance degradation.
3. **Security:** Ensure data protection, secure payment processing, and guard against vulnerabilities.
4. **Availability:** Guarantee high uptime with minimal disruptions to service.
5. **Usability:** Maintain an intuitive interface for customers, stores, and admins.
6. **Maintainability:** Use a well-structured, documented codebase for future upgrades and bug fixes.
7. **Reliability:** Ensure consistent and error-free operations.

Additional Considerations:

- **Mobile Responsiveness:** Optimize the platform for use on mobile devices.
- **Accessibility:** Adhere to accessibility standards for users with disabilities.
- **Compliance:** Follow data privacy regulations and industry security standards.

➢ User Stories Behind Development of SweetSpot:

For Customers:

1. **Enhanced Discovery:**

"As a customer, I want to browse cakes from nearby stores so I can find unique options quickly." Solution: Location-based search and partnerships with local bakeries.

2. **Customization for All Occasions:**

"As a customer, I want to customize cakes to suit different occasions and dietary preferences." Solution: Intuitive customization interface for flavors, sizes, and toppings.

3. **Transparency through Tracking:**

"As a customer, I want to track my order status so I know when my cake will arrive." Solution: Robust order tracking system with automated email notifications.

4. **Seamless Payment Integration:**

"As a customer, I want secure and simple payment options for a smooth checkout experience." Solution: Integration with secure payment gateways.

For Store Owners:

1. **Showcasing Expertise:**

"As a store owner, I want to upload detailed cake information to attract customers." Solution: Store dashboard for managing cake details, images, and prices.

2. **Efficient Order Management:**

"As a store owner, I need a simple way to manage incoming orders." Solution: Centralized dashboard for receiving, reviewing, and updating orders.

3. **Customer Communication:**

"As a store owner, I want to update customers about their order status to manage expectations." Solution: Order status update functionality to ensure transparency.

◆ Technical Stack:

Backend Technologies

- **Python**: The core programming language used for application development, known for its simplicity and versatility.
- **Django**: A high-level web framework that simplifies backend development with features like ORM, user authentication, and admin interfaces.
- **Django REST Framework (DRF)**: A toolkit for creating robust and scalable RESTful APIs.
- **PostgreSQL**: A powerful open-source relational database system for storing and managing structured data.

Frontend Technologies

- **HTML/CSS**: Used for designing and styling the structure and layout of web pages.
- **Streamlit**: A lightweight framework for creating interactive user interfaces and dashboards.

APIs and Integrations

- **Google Maps API**: Enables embedding maps and providing location-based services.
- **Formspree**: API for handling form submissions efficiently.
- **Swagger/OpenAPI**: Utilized for documenting and testing APIs, ensuring clear communication with developers.
- **SMTP (Gmail)**: For sending automated email notifications and facilitating user communication.

Tools and Platforms

- **Visual Studio Code**: The primary IDE for coding, offering features like debugging, extensions, and version control.
- **Postman**: For testing and debugging API endpoints during development.
- **GitHub**: For version control, collaboration, and maintaining the project's codebase.

◆ Architecture/Design:

Overview of the System Architecture

The architecture of SweetSpot is designed for scalability, reliability, and efficiency, ensuring a seamless experience for customers, store owners, and administrators. It integrates various components to provide robust functionality while maintaining a user-friendly interface.

Backend

1. **Django Web Framework**:
    - Core of the backend, handling tasks such as user authentication, database interactions, and business logic.
    - Acts as the central hub, processing requests and generating responses.
2. **Django REST Framework (DRF)**:
    - Facilitates the creation of RESTful APIs that serve as communication channels between the frontend and backend.
    - Supports easy integration with external systems.
3. **PostgreSQL Database**:
    - A highly reliable relational database storing crucial data such as user accounts, cake details, orders, and store information.
    - Ensures data integrity and scalability for increasing user demands.

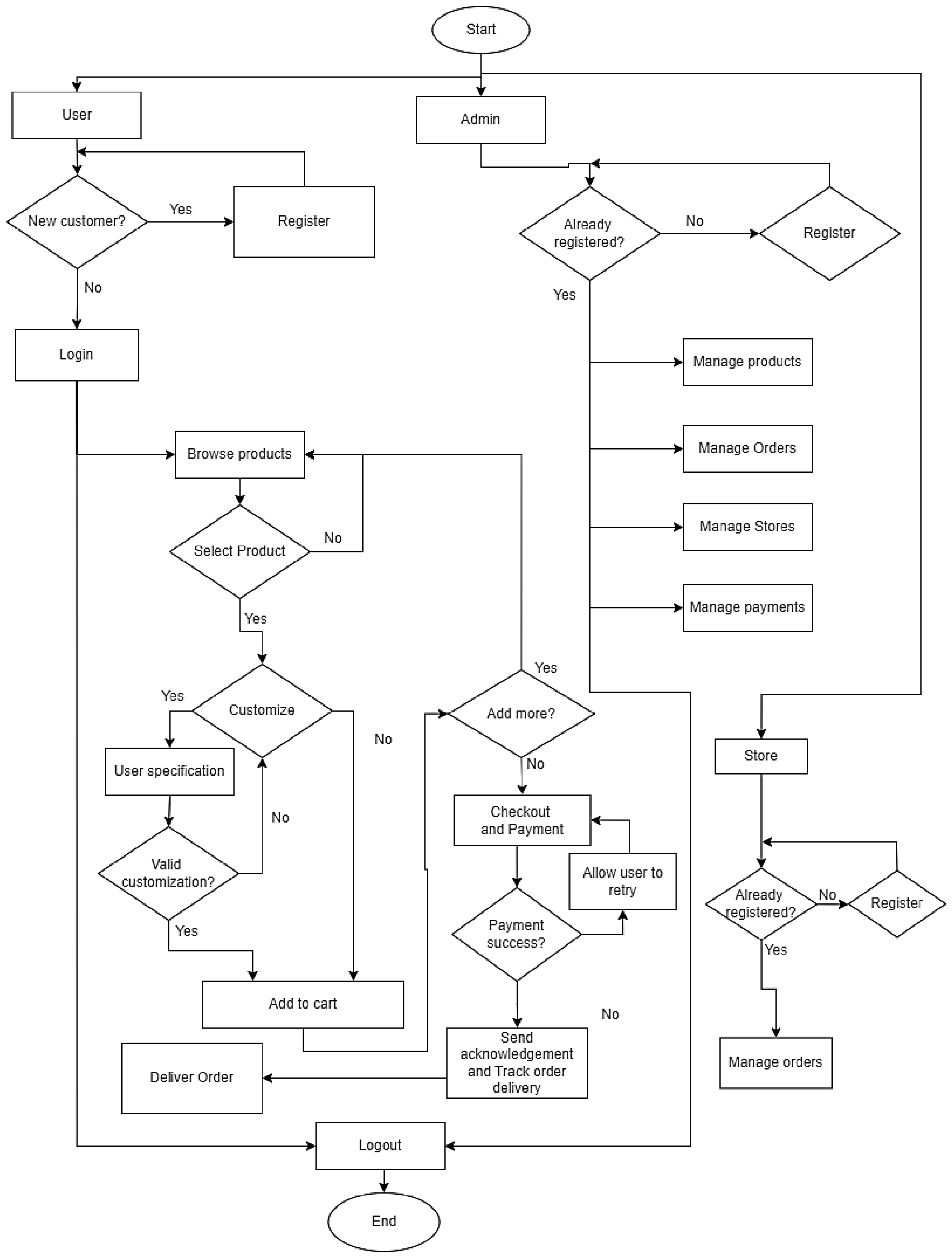
External Services

1. **Payment Gateway**:
    - PayPal integration enables secure and efficient payment processing for online orders.
    - Django interacts with the gateway to manage transactions securely.
2. **Email Service**:
    - SMTP (e.g., Gmail) facilitates automated email notifications for key events like order confirmation and delivery updates.
    - Triggered by Django for seamless communication.
3. **Google Maps Platform (Optional)**:
    - Utilizes Distance Matrix API to calculate delivery distances and estimated times.
    - Enhances transparency and improves customer satisfaction by providing accurate delivery details.

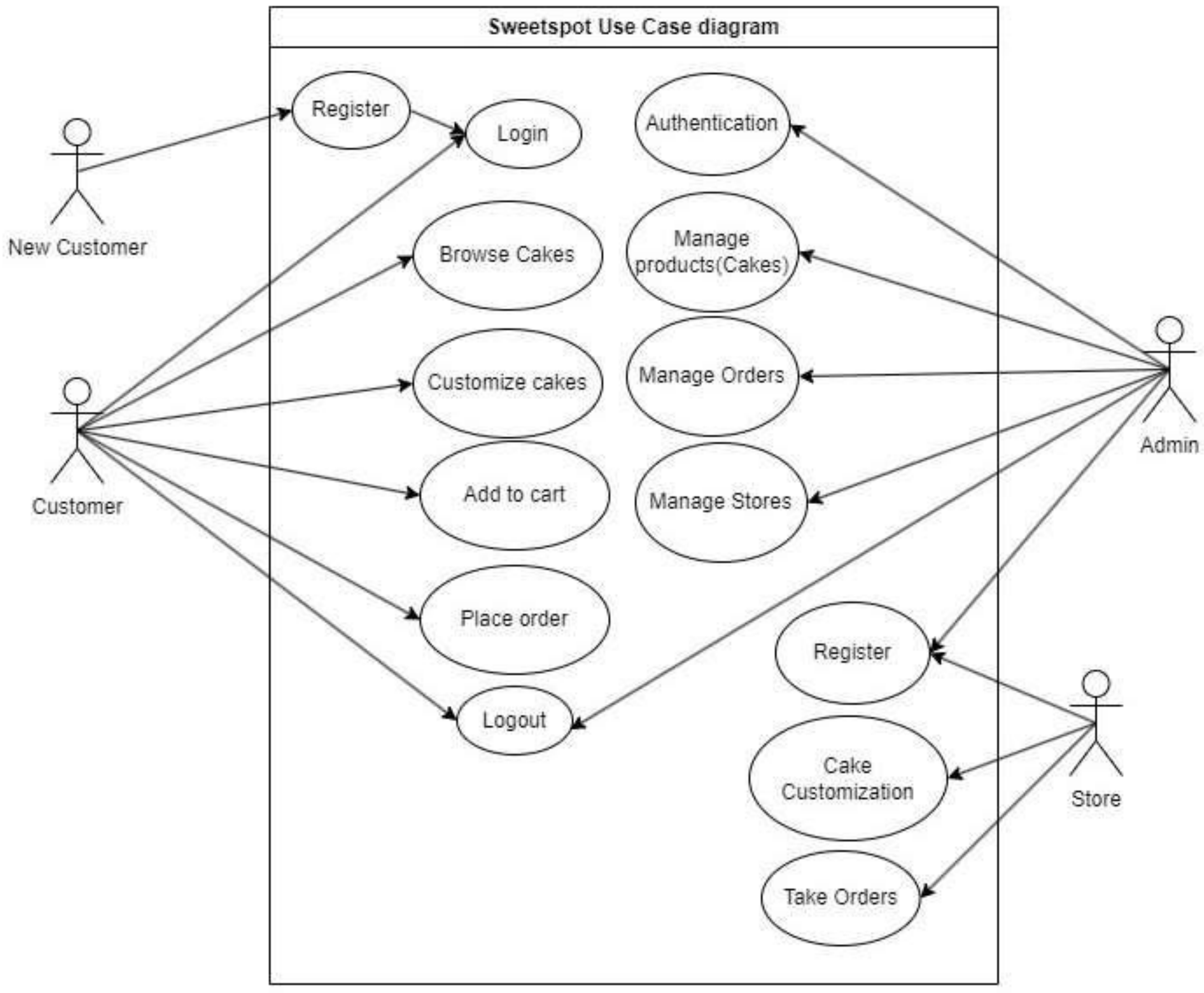
Communication Flow

1. **API Requests**:
   - The frontend sends API requests to endpoints in the Django backend for actions like login, order placement, or cake browsing.
2. **Django Processing**:
   - The backend processes requests, validates user credentials, retrieves or updates data in PostgreSQL, and interacts with external services when required.
3. **External Services Interaction**:
   - Django communicates with services like the payment gateway to handle payments or with email servers for notifications.
4. **API Response**:
   - After processing, Django sends a structured response back to the frontend with relevant data or status updates.
5. **Frontend Update**:
   - The UI dynamically updates based on the backend response, providing users with real-time information and feedback.

Design Flowchart:



UML Diagrams



The Use Case Diagram for the Sweetspot platform outlines the roles and interactions of various actors, including new customers, customers, store owners, and the admin, with the system's functionalities. New customers can register and log in to access features like browsing cakes, customizing orders, managing their cart, and placing orders. Existing customers further interact by viewing cake options, personalizing selections, and completing transactions. Store owners manage their profiles, upload products, customize cake options, and handle order updates. The admin oversees critical platform functionalities, including user authentication, managing stores, monitoring products, and handling orders. This diagram highlights the seamless connectivity and functionality of the platform for all stakeholders, emphasizing Sweetspot's goal to provide a user-friendly experience.

◆ Development:

The development of the SweetSpot project relied on modern technologies and frameworks to create a robust, scalable, and secure online cake delivery system. This section details the technologies used, the coding practices followed, and the challenges addressed during the implementation process.

Technologies and Frameworks Used

1. Programming Language: Python (version 3.9)

- Python was chosen for its simplicity, readability, and extensive standard library, which accelerated development. It also supports multiple programming paradigms (procedural, object-oriented, and functional), offering flexibility for different use case
- Its concise syntax enabled faster development, while its vast ecosystem of third-party libraries helped in solving complex problems efficiently.

2. Web Framework: Django (version 3.2)

- Django, known for its rapid development capabilities, was selected for its comprehensive set of tools, including ORM, authentication, routing, and admin interface
- The framework adheres to the **DRY (Don't Repeat Yourself)** principle, reducing code redundancy and increasing maintainability.
- Its scalable architecture makes it suitable for applications with growing traffic and data.

4. API Framework: Django REST Framework (DRF) (version 3.14.0)

- DRF was used to build RESTful APIs seamlessly integrated with Django models and views.
- Features such as serializers, authentication, and permission classes ensured secure and efficient API management, while its interactive API browser enhanced testing and debugging.

4. Database: PostgreSQL

- PostgreSQL, an open-source relational database, was chosen for its advanced features, including full-text search and JSON support.
- Its ACID compliance ensures data integrity, making it ideal for handling complex queries and transactional operations in the SweetSpot system.

5. Development Tools

- **Visual Studio Code (IDE):** Used for its integrated terminal, debugging tools, and support for extensions, which enhanced productivity and streamlined development.
- **Postman:** Employed for API testing, ensuring endpoints function as intended through collections of automated requests.

Coding Standards and Best Practices

The development team adhered to several coding standards to ensure scalability, maintainability, and security:

1. **Project Structure:**
    - Followed Django's modular structure with separate apps for functionalities like customers, orders, and cakes.
    - Adopted the MVT (Model-View-Template) pattern for a clear separation of concerns, simplifying the codebase.
2. **Version Control:**
    - Used Git for version tracking, employing a branching strategy (feature, development, and main branches) for collaborative work.
    - Maintained clear and descriptive commit messages for ease of understanding project history.
3. **Testing:**
    - Conducted unit testing with Django's built-in framework to validate individual components like models and views.
    - Performed integration testing using Postman to ensure seamless interaction between system components.
    - Automated tests with continuous integration tools (e.g., GitHub Actions) to prevent regressions.
4. **Security Measures:**
    - Leveraged Django's built-in authentication and DRF's permission classes for secure API endpoints.
    - Used environment variables for managing sensitive data like database credentials and secret keys.
    - Implemented protection against common vulnerabilities, including SQL injection, XSS, and CSRF.
5. **Documentation:**
    - Utilized Swagger and DRF's built-in tools to maintain up-to-date API documentation.
    - Added docstrings and comments to explain complex logic, aiding future developers.

Challenges Encountered and Solutions

1. **Database Migrations:**
   - Challenge: Ensuring smooth database migrations during schema changes.
   - Solution: Tested migrations in a development environment before deployment. Used Django's migration tools to handle incremental changes with rollback strategies for errors.
2. **API Versioning:**
   - Challenge: Managing API changes without breaking existing clients.
   - Solution: Implemented versioning through URL patterns (e.g., /api/v1/) to allow multiple API versions to coexist, giving clients flexibility to migrate.
3. **Cake Customizations:**
   - Challenge: Developing a flexible system to handle diverse customization requests.
   - Solution: Created a robust data model linking customizations to cakes and validated customization data using serializers for seamless integration into the order process.
4. **Email Notifications:**
   - Challenge: Ensuring reliable email delivery for notifications such as order confirmations.
   - Solution: Integrated Django's email backend with SMTP services and implemented asynchronous email sending using Celery, reducing load on the main thread.

5. Performance Optimization:

- Challenge: Optimizing system performance to handle high traffic and large data volumes.
- Solution: Employed caching with Redis, optimized database queries using indexing, and reduced redundant queries with Django's select_related and prefetch_related. Load testing and profiling were performed to identify and resolve bottlenecks.

◆ Testing:

The development of *Sweetspot* emphasized not only feature creation but also delivering a seamless, bug-free user experience. To ensure this, a **multi-layered testing strategy** was adopted, incorporating **unit tests, integration tests, and system tests**. Each layer validated different aspects of the application, from individual components to end-to-end workflows, ensuring comprehensive reliability and stability.

1. Unit Testing: Building Blocks of Reliability

Unit tests were implemented to verify the functionality of isolated components, such as models, views, and utility functions. This foundational testing layer ensured that each building block of Sweetspot worked correctly.

- **Purpose:** To validate individual components in isolation, ensuring accuracy and reliability.
- **Tools Used:**
    1. Django's built-in testing framework (django.test)
    2. Python's unittest module
- **Scope:**
    1. **Models:** Ensured database schema integrity and proper relationships.
    2. **Views:** Verified expected HTTP status codes and responses.
    3. **Serializers:** Tested data serialization/deserialization processes.
    4. **Utilities:** Validated helper functions and custom logic for correctness.

2. Integration Testing: Bridging the Gaps

Integration tests focused on verifying interactions between components and ensuring proper data flow across the system. This layer addressed scenarios where modules worked together, such as the communication between the user interface, backend, and database.

- **Purpose:** To confirm that interconnected components work harmoniously.
- **Tools Used:**Django's testing framework
    1. Postman for REST API validation
    2. pytest for advanced integration testing
- **Scope:**
    1. **API Endpoints:** End-to-end testing of REST APIs, including authentication and CRUD operations.
    2. **User Scenarios:** Real-world flows like placing an order, customizing cakes, and tracking delivery.

3. System Testing: Simulating the Real World

System tests evaluated the overall functionality from an end-user's perspective, simulating real-world workflows like searching for cakes, placing orders, and processing payments. This layer was critical in validating the complete system's usability and performance.

- **Purpose:** To ensure the application meets user requirements in a production-like environment.
- **Tools Used:**
    1. Selenium for browser automation and UI testing
    2. Manual testing for hands-on verification
- **Scope:**
    1. **End-to-End Scenarios:** Full workflows, including user registration, cart management, and order tracking.
    2. **UI/UX Testing:** Checked responsiveness and usability across devices and browsers.

Results of the Testing Phase

The thorough multi-layered testing approach led to the discovery and resolution of several critical bugs and optimization needs. Below are the key findings and resolutions:

Key Issues and Fixes

1. **Incorrect Order Total Calculation**
    - **Issue:** Floating-point errors caused incorrect price calculations.
    - **Resolution:** Switched to Django's DecimalField and the decimal module for precise monetary operations.

2. **API Endpoint Authentication**
    - **Issue:** Some endpoints were accessible without authentication.
    - **Resolution:** Applied DRF's IsAuthenticated and custom permission classes to secure sensitive endpoints.

3. **Slow Response Times for Complex Queries**
    - **Issue:** Complex database queries led to delays
    - **Resolution:** Optimized queries using select_related and prefetch_related. Caching was implemented for frequently accessed data.

4. **Email Delivery Failures**
    - **Issue:** Emails for order confirmations and delivery tracking were unreliable.
    - **Resolution:** Integrated a reliable email service with **Celery** and **Redis** for asynchronous email processing.

5. **Order Status Not Updating**

    - **Issue:** Payment or delivery updates weren’t reflected in the order status.
    - **Resolution:** Fixed workflow logic to ensure proper status updates based on events.

Deployment:

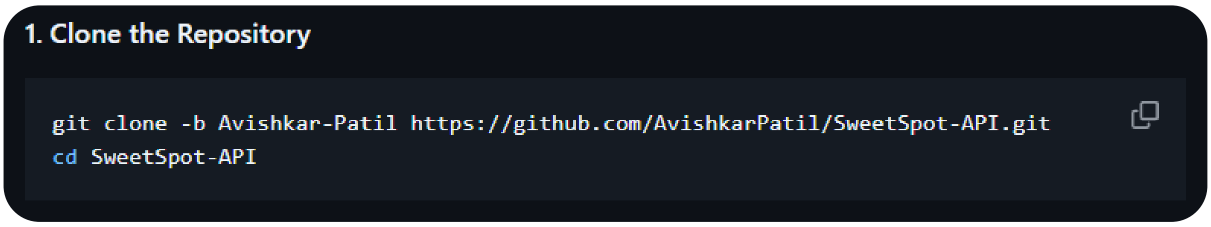
1. Deployment Process Overview

Steps Involved in Deployment

1) **Code Preparation:**
   - Merge code changes into the master branch.
   - Run unit and integration tests to ensure functionality.
   - Perform static code analysis to detect potential issues.
2) **Building for Deployment:**
   - Use deployment scripts (e.g., Python or Bash) to automate tasks such as:
   - Collecting static files.
   - Applying database migrations.
   - Creating a production-ready application build.
3) **Environment Selection:**
   - Deployment script allows selection of the target environment: Development, Staging, or Production.
   - Load environment-specific configurations (database credentials, API keys).
4) **Server Deployment:**
   - Transfer files securely using tools like SFTP or resync.
   - Deploy the build and configuration files to the target server.
5) **Service Restart and Verification:**
   - Restart the web server
   - Perform health checks to ensure application functionality.

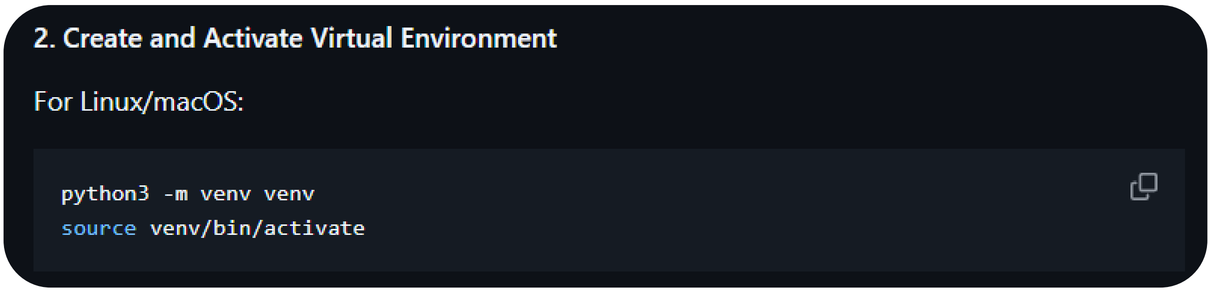
2. Environment Setup Instructions

Setting Up the Server



1. Clone the Repository

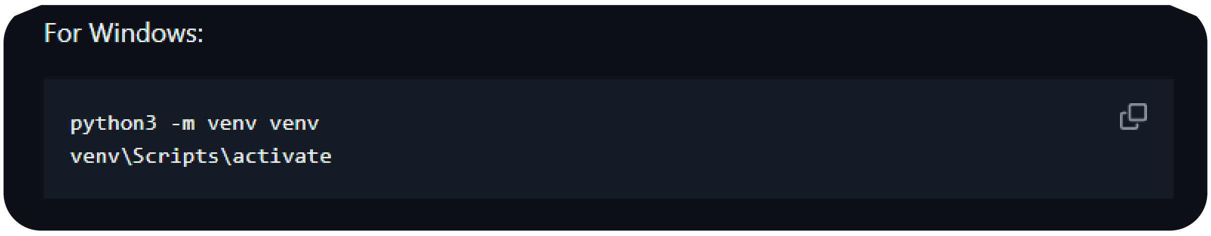
```
git clone -b Avishkar-Patil https://github.com/AvishkarPatil/SweetSpot-API.git
cd SweetSpot-API
```



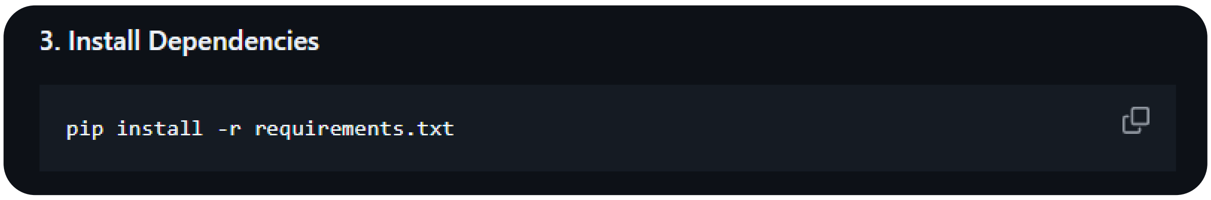
2. Create and Activate Virtual Environment

For Linux/macOS:

```
python3 -m venv venv
source venv/bin/activate
```


For Windows:

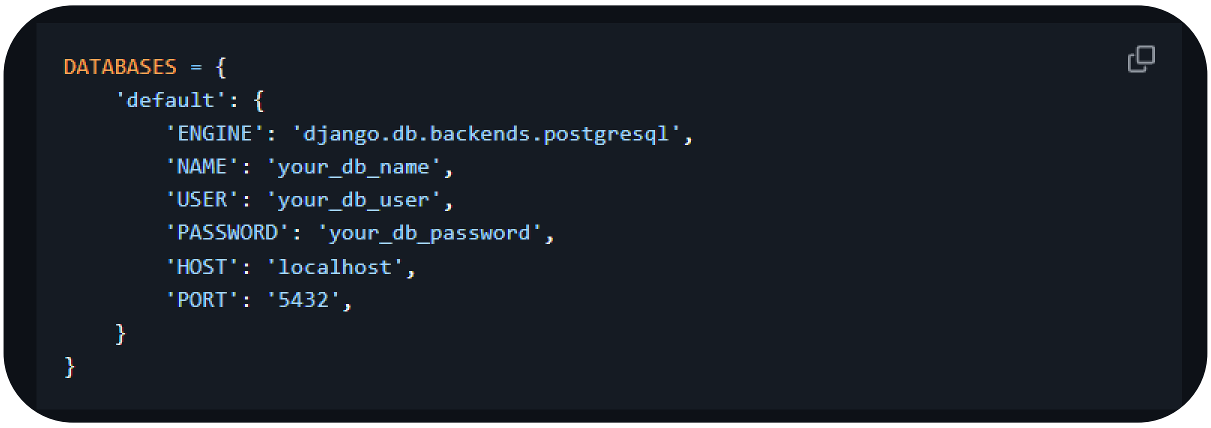
```
python3 -m venv venv
venv\Scripts\activate
```


3. Install Dependencies

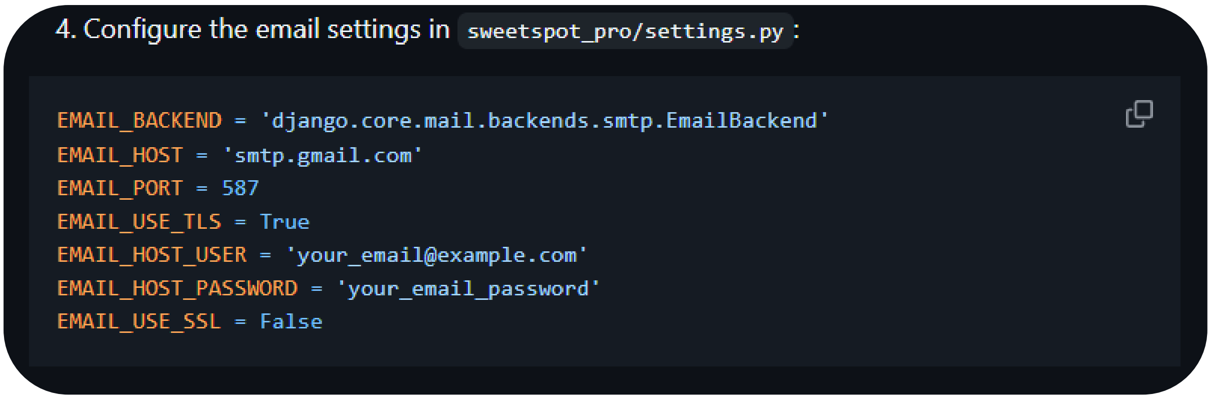
```
pip install -r requirements.txt
```

4. Database and Email Configuration

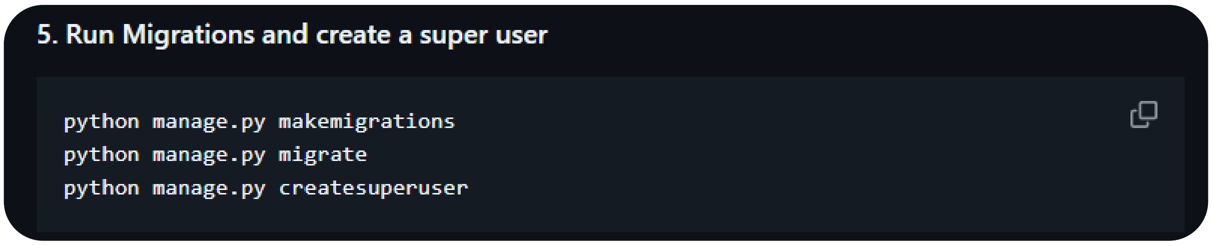
1. Install and start PostgreSQL server
2. Create a new PostgreSQL database
3. Update the database configuration in sweetspot_pro/settings.py:



```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'your_db_name',
        'USER': 'your_db_user',
        'PASSWORD': 'your_db_password',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}
```

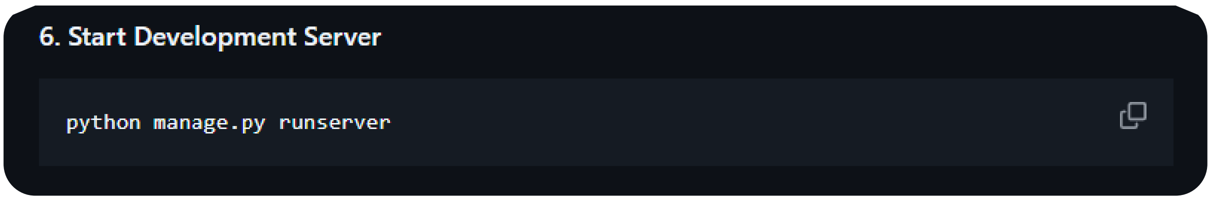

4. Configure the email settings in `sweetspot_pro/settings.py`:

```
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = 587
EMAIL_USE_TLS = True
EMAIL_HOST_USER = 'your_email@example.com'
EMAIL_HOST_PASSWORD = 'your_email_password'
EMAIL_USE_SSL = False
```


5. Run Migrations and create a super user

```
python manage.py makemigrations
python manage.py migrate
python manage.py createsuperuser
```

You can access the admin panel at http://localhost:8000/admin to add sample data.

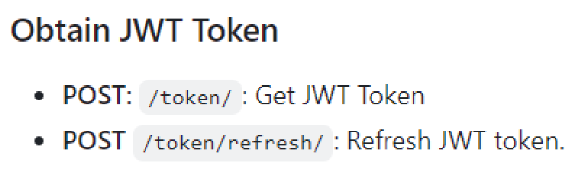

6. Start Development Server

```
python manage.py runserver
```

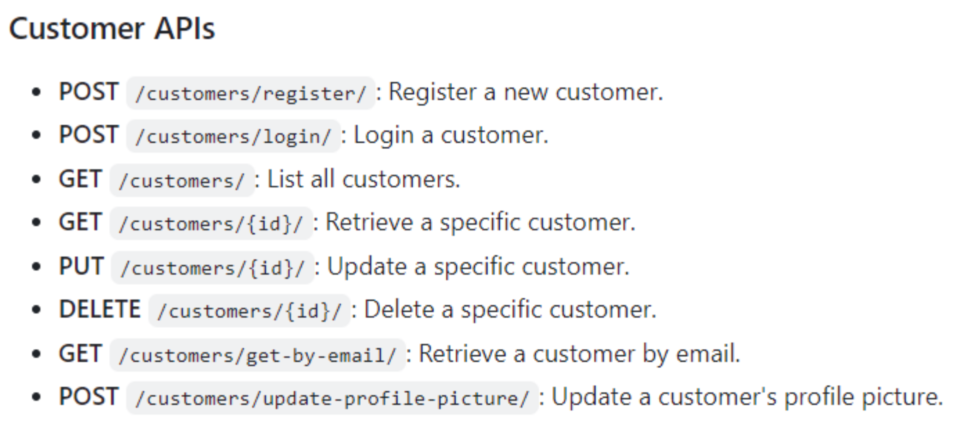
3. Environment-Specific Guidelines

1) **Development Environment:**
    - Hosted locally or on a VM for the team.
    - Skip migrations if using a test database
2) **Staging Environment:**
    - Mirrors production for final testing.
    - Follow the production deployment process with staging-specific configs.
3) **Production Environment:**
    - Public-facing environment requiring maximum security:
        - Enable HTTPS.
        - Use environment variables for sensitive data.
        - Schedule regular database and codebase backups.

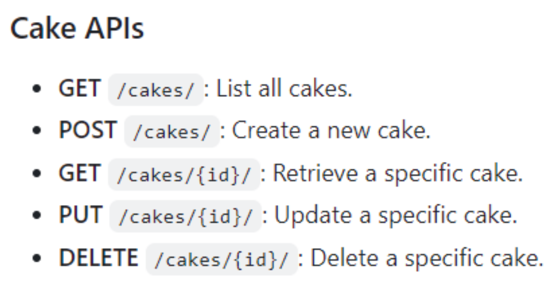
4. SweetSpot API Usage


Obtain JWT Token

- POST: `/token/` : Get JWT Token
- POST `/token/refresh/` : Refresh JWT token.


Customer APIs

- POST `/customers/register/` : Register a new customer.
- POST `/customers/login/` : Login a customer.
- GET `/customers/` : List all customers.
- GET `/customers/{id}/` : Retrieve a specific customer.
- PUT `/customers/{id}/` : Update a specific customer.
- DELETE `/customers/{id}/` : Delete a specific customer.
- GET `/customers/get-by-email/` : Retrieve a customer by email.
- POST `/customers/update-profile-picture/` : Update a customer's profile picture.


Cake APIs

- GET `/cakes/` : List all cakes.
- POST `/cakes/` : Create a new cake.
- GET `/cakes/{id}/` : Retrieve a specific cake.
- PUT `/cakes/{id}/` : Update a specific cake.
- DELETE `/cakes/{id}/` : Delete a specific cake.

Cake Customization APIs

- GET `/cake-customizations/` : List all cake customizations.
- POST `/cake-customizations/` : Create a new cake customization.
- GET `/cake-customizations/{id}/` : Retrieve a specific cake customization.
- PUT `/cake-customizations/{id}/` : Update a specific cake customization.
- DELETE `/cake-customizations/{id}/` : Delete a specific cake customization.

Cart APIs

- GET `/carts/` : List all cart items.
- POST `/add-cake-to-cart/` : Add a cake to the cart.
- GET `/carts/{id}/` : Retrieve a specific cart item.
- PUT `/carts/{id}/` : Update a specific cart item.
- DELETE `/carts/{id}/` : Delete a specific cart item.

Order APIs

- GET `/orders/` : List all orders.
- POST `/orders/` : Create a new order.
- GET `/orders/{id}/` : Retrieve a specific order.
- PUT `/orders/{id}/` : Update a specific order.
- DELETE `/orders/{id}/` : Delete a specific order.
- GET `/orders/by-user/` : Retrieve orders by user ID.
- GET `/orders/delivery-tracking/{id}/` : Track delivery of a specific order.

Store APIs

- GET `/stores/` : List all stores.
- POST `/stores/` : Create a new store.
- GET `/stores/{id}/` : Retrieve a specific store.
- PUT `/stores/{id}/` : Update a specific store.
- DELETE `/stores/{id}/` : Delete a specific store.
- GET `/stores/{id}/cakes/` : List all cakes in a specific store.
- GET `/stores/search/` : Search stores by name or city.
- GET `/stores/filter/` : Filter stores by city.
- GET `/stores/{id}/store-has-cakes/` : Check if a store has cakes.

Payment APIs

- POST `/payment/` : Process a payment.

Admin API Documentation

Admin Customer APIs

- GET `/admin/customers/` : List all customers.
- POST `/admin/customers/` : Create a new customer.
- GET `/admin/customers/{id}/` : Retrieve a specific customer.
- PUT `/admin/customers/{id}/` : Update a specific customer.
- DELETE `/admin/customers/{id}/` : Delete a specific customer.

Admin Cake APIs

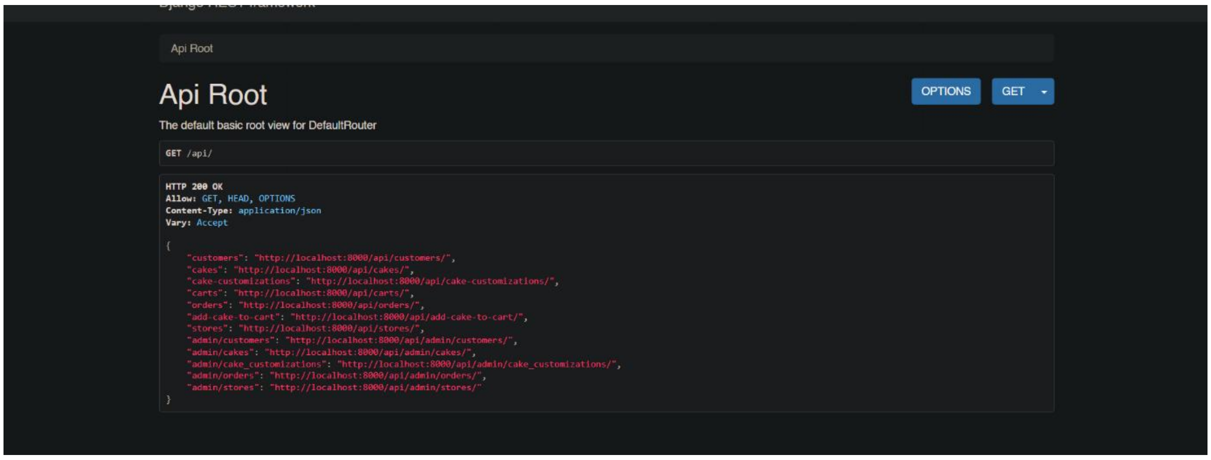
- GET `/admin/cakes/` : List all cakes.
- POST `/admin/cakes/` : Create a new cake.
- GET `/admin/cakes/{id}/` : Retrieve a specific cake.
- PUT `/admin/cakes/{id}/` : Update a specific cake.
- DELETE `/admin/cakes/{id}/` : Delete a specific cake.

Admin Order APIs

- GET `/admin/orders/` : List all orders.
- POST `/admin/orders/` : Create a new order.
- GET `/admin/orders/{id}/` : Retrieve a specific order.
- PUT `/admin/orders/{id}/` : Update a specific order.
- DELETE `/admin/orders/{id}/` : Delete a specific order.

Admin Store APIs

- GET `/admin/stores/` : List all stores.
- POST `/admin/stores/` : Create a new store.
- GET `/admin/stores/{id}/` : Retrieve a specific store.
- PUT `/admin/stores/{id}/` : Update a specific store.
- DELETE `/admin/stores/{id}/` : Delete a specific store.
- GET `/admin/stores/{id}/store-has-cakes/` : List cakes available in a specific store.

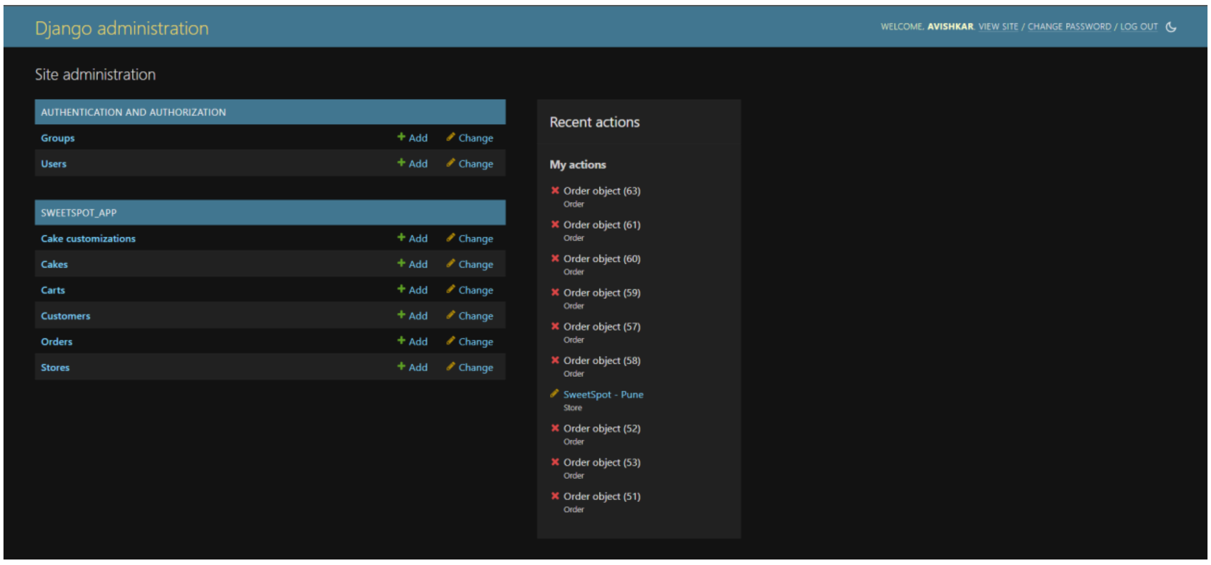


Api Root

The default basic root view for DefaultRouter

```
GET /api/
```

```
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "customers": "http://localhost:8000/api/customers/",
    "cakes": "http://localhost:8000/api/cakes/",
    "cake-customizations": "http://localhost:8000/api/cake-customizations/",
    "carts": "http://localhost:8000/api/carts/",
    "orders": "http://localhost:8000/api/orders/",
    "add-cake-to-cart": "http://localhost:8000/api/add-cake-to-cart/",
    "stores": "http://localhost:8000/api/stores/",
    "admin/customers": "http://localhost:8000/api/admin/customers/",
    "admin/cakes": "http://localhost:8000/api/admin/cakes/",
    "admin/cake_customizations": "http://localhost:8000/api/admin/cake_customizations/",
    "admin/orders": "http://localhost:8000/api/admin/orders/",
    "admin/stores": "http://localhost:8000/api/admin/stores/"
}
```



Site administration

AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	Change
Users	+ Add	Change

SWEETSPOT_APP		
Cake customizations	+ Add	Change
Cakes	+ Add	Change
Carts	+ Add	Change
Customers	+ Add	Change
Orders	+ Add	Change
Stores	+ Add	Change

Recent actions

My actions

- ✗ Order object (63) Order
- ✗ Order object (61) Order
- ✗ Order object (60) Order
- ✗ Order object (59) Order
- ✗ Order object (57) Order
- ✗ Order object (58) Order
- SweetSpot - Pune Store
- ✗ Order object (52) Order
- ✗ Order object (53) Order
- ✗ Order object (51) Order

- ❑ **Installation:** https://github.com/AvishkarPatil/SweetSpot-API/blob/main/Documentation.md

- ❑ **API Documentation:** https://github.com/AvishkarPatil/SweetSpot-API/blob/main/APIs.md

- ❑ **Screenshots:** https://github.com/AvishkarPatil/SweetSpot-API/blob/main/Screenshots.md

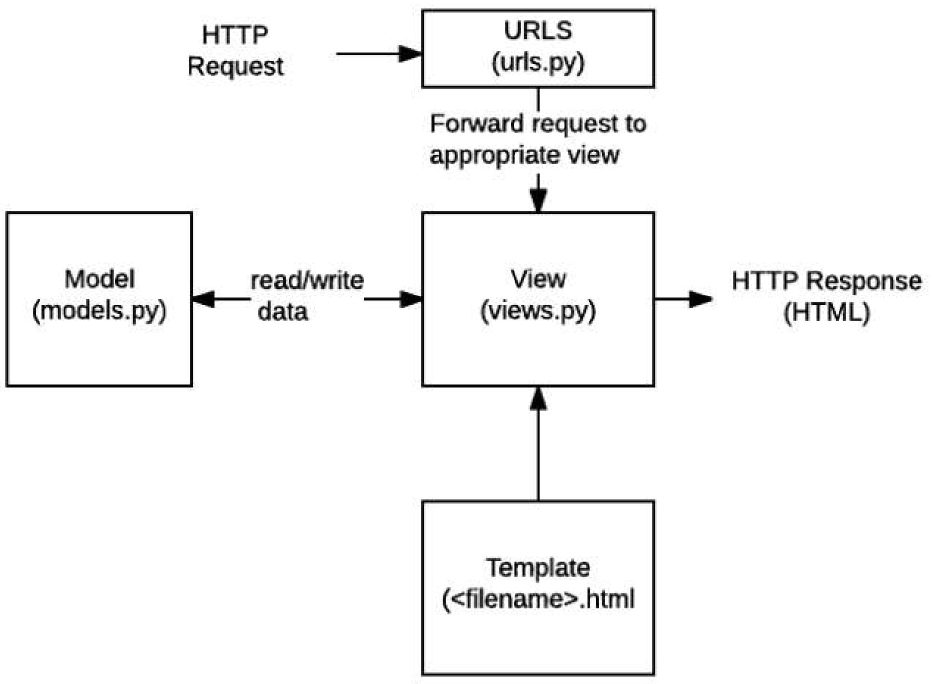
◆ Conclusion:

SweetSpot, the delightful online cake delivery platform, has successfully transitioned from development to launch, showcasing the collaborative efforts of a dedicated development team. Designed to bridge the gap between cake enthusiasts and bakeries, SweetSpot offers a user-centric platform that ensures a seamless and enjoyable experience for customers and bakery owners alike. Built on a robust backend powered by Python, Django, and PostgreSQL, the platform provides a secure and scalable foundation for growth. With a user-friendly interface, Sweetspot simplifies the process of browsing cakes, placing orders, and tracking deliveries, all while maintaining a strong focus on security and reliability through rigorous coding practices and comprehensive testing. Automated deployment scripts further streamline the transition from development to production, reducing human error and ensuring consistency.

Throughout the project, key lessons were learned, including the value of focusing on a Minimum Viable Product (MVP) to enable faster launches and gather valuable user feedback, as well as the importance of clear documentation for smoother development and onboarding. Looking ahead, the team plans to continuously monitor user feedback, performance, and industry trends to guide future improvements. Potential enhancements include developing a dedicated mobile app for on-the-go convenience, implementing real-time order tracking for better transparency, and integrating social login options to streamline the registration process and attract more customers. Sweetspot's launch marks the start of an exciting journey, with a commitment to ongoing innovation and an enhanced user experience.

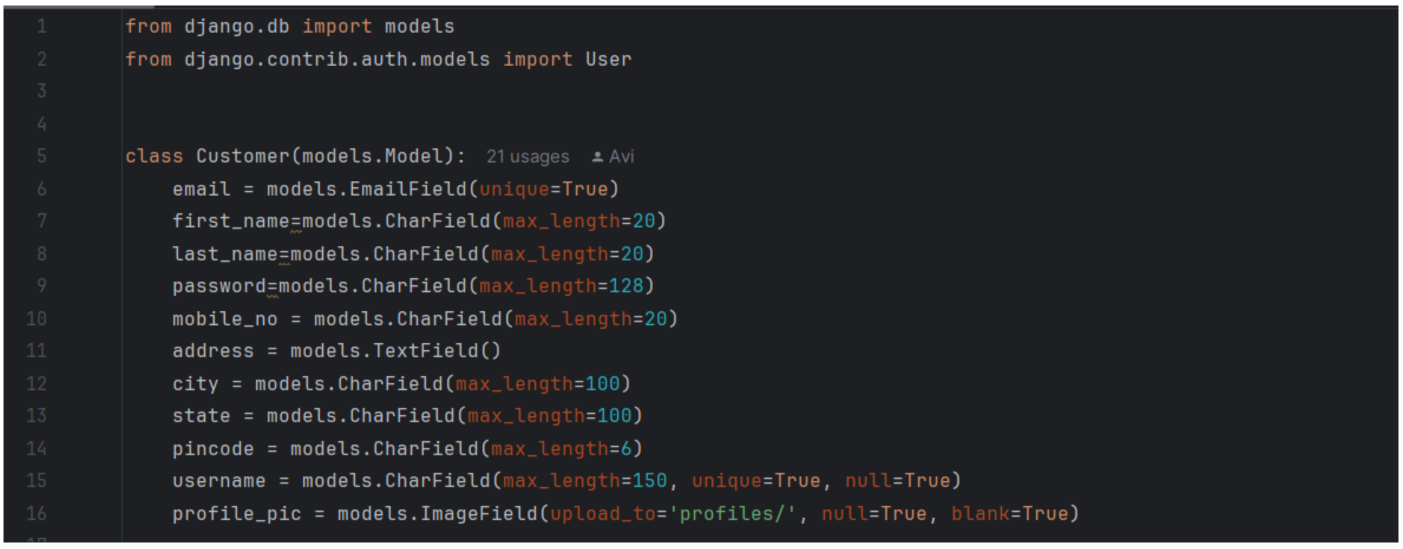
◆ Appendices:

Appendix A: Platform Architecture Diagram

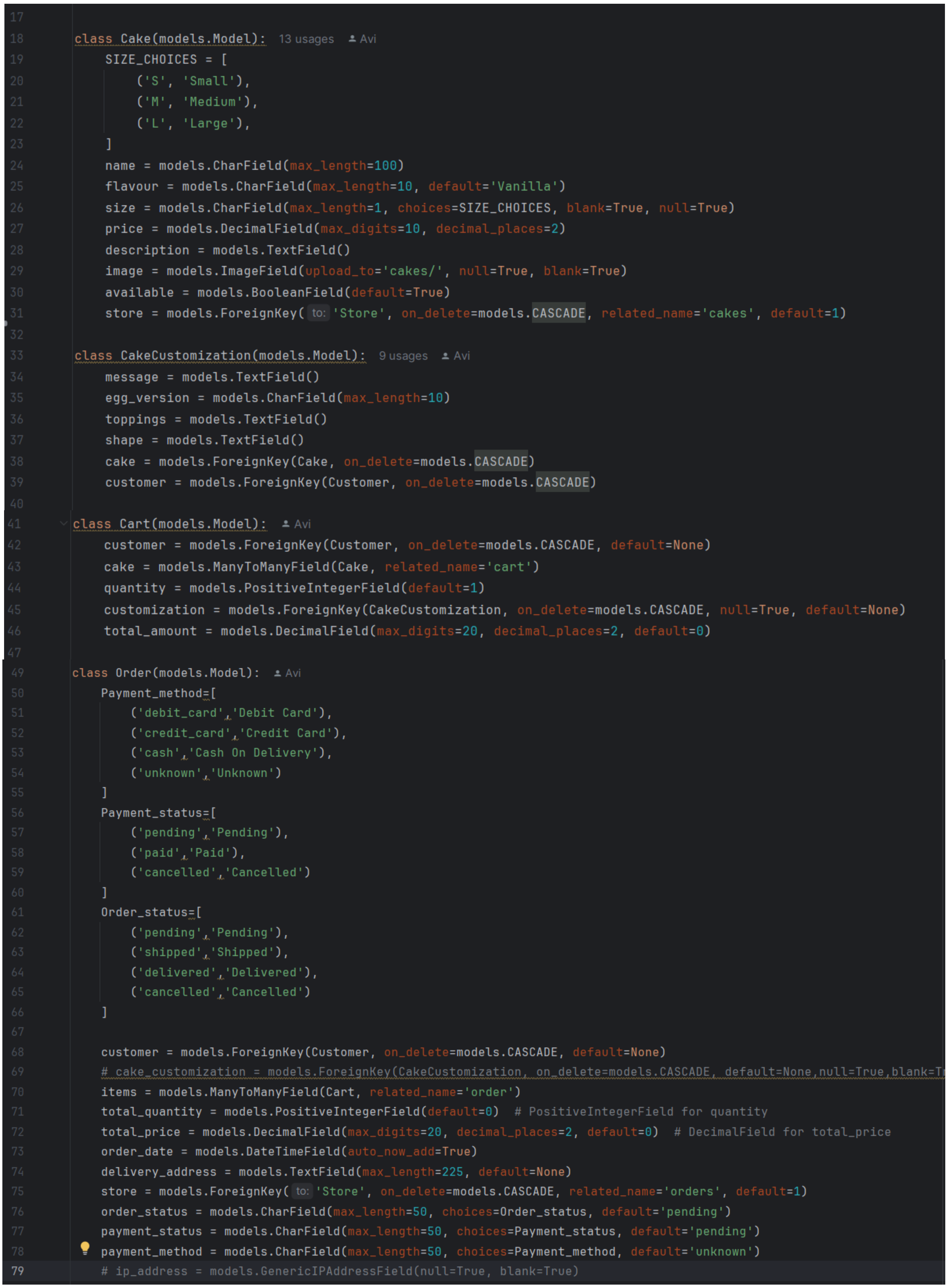


Working Flow of Django REST Framework Diagram

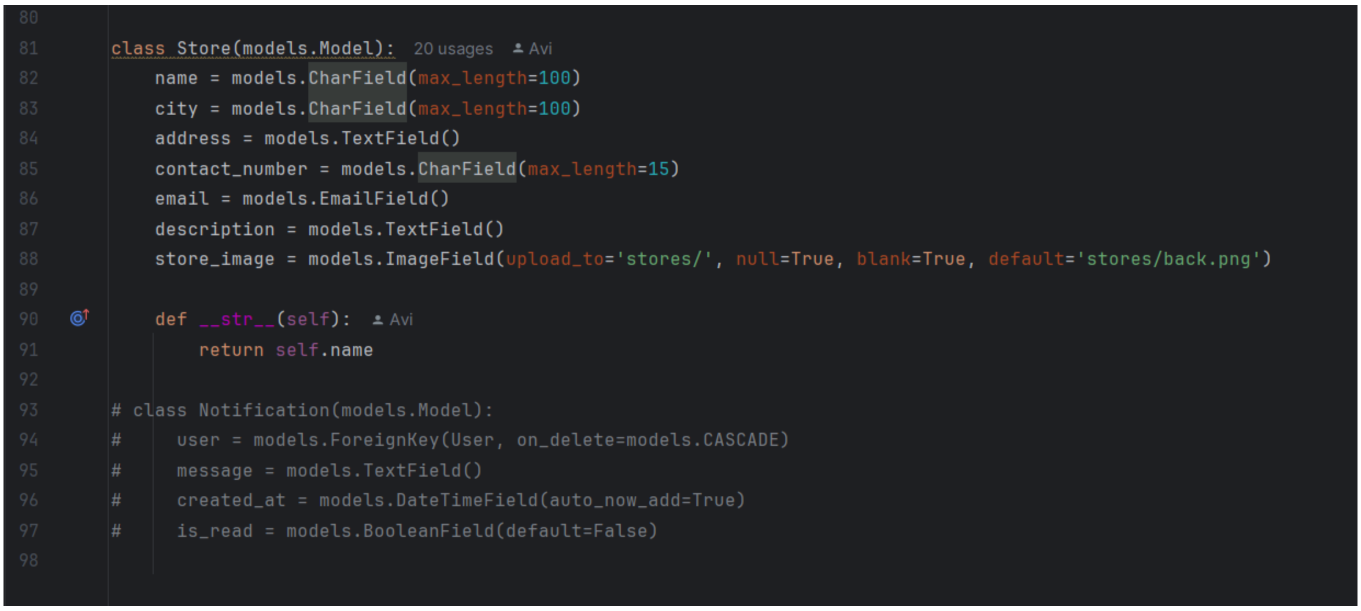
Appendix B: Sample Code Snippets



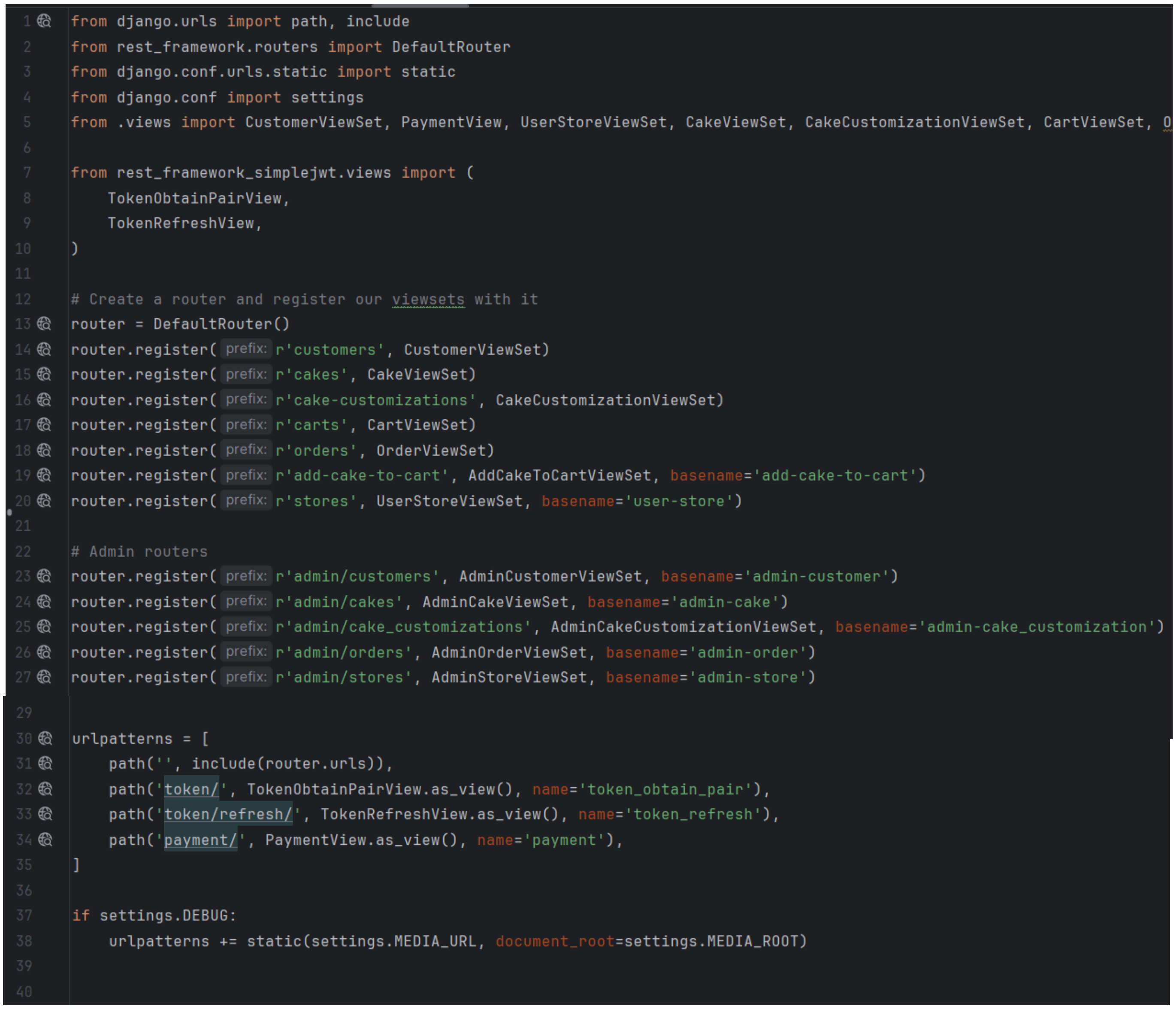
```
from django.db import models
from django.contrib.auth.models import User


class Customer(models.Model):  21 usages  Avi
    email = models.EmailField(unique=True)
    first_name=models.CharField(max_length=20)
    last_name=models.CharField(max_length=20)
    password=models.CharField(max_length=128)
    mobile_no = models.CharField(max_length=20)
    address = models.TextField()
    city = models.CharField(max_length=100)
    state = models.CharField(max_length=100)
    pincode = models.CharField(max_length=6)
    username = models.CharField(max_length=150, unique=True, null=True)
    profile_pic = models.ImageField(upload_to='profiles/', null=True, blank=True)
```



```python
class Cake(models.Model):  13 usages  Avi
    SIZE_CHOICES = [
        ('S', 'Small'),
        ('M', 'Medium'),
        ('L', 'Large'),
    ]
    name = models.CharField(max_length=100)
    flavour = models.CharField(max_length=10, default='Vanilla')
    size = models.CharField(max_length=1, choices=SIZE_CHOICES, blank=True, null=True)
    price = models.DecimalField(max_digits=10, decimal_places=2)
    description = models.TextField()
    image = models.ImageField(upload_to='cakes/', null=True, blank=True)
    available = models.BooleanField(default=True)
    store = models.ForeignKey( to: 'Store', on_delete=models.CASCADE, related_name='cakes', default=1)

class CakeCustomization(models.Model):  9 usages  Avi
    message = models.TextField()
    egg_version = models.CharField(max_length=10)
    toppings = models.TextField()
    shape = models.TextField()
    cake = models.ForeignKey(Cake, on_delete=models.CASCADE)
    customer = models.ForeignKey(Customer, on_delete=models.CASCADE)

class Cart(models.Model):  Avi
    customer = models.ForeignKey(Customer, on_delete=models.CASCADE, default=None)
    cake = models.ManyToManyField(Cake, related_name='cart')
    quantity = models.PositiveIntegerField(default=1)
    customization = models.ForeignKey(CakeCustomization, on_delete=models.CASCADE, null=True, default=None)
    total_amount = models.DecimalField(max_digits=20, decimal_places=2, default=0)

class Order(models.Model):  Avi
    Payment_method=[
        ('debit_card','Debit Card'),
        ('credit_card','Credit Card'),
        ('cash','Cash On Delivery'),
        ('unknown','Unknown')
    ]
    Payment_status=[
        ('pending','Pending'),
        ('paid','Paid'),
        ('cancelled','Cancelled')
    ]
    Order_status=[
        ('pending','Pending'),
        ('shipped','Shipped'),
        ('delivered','Delivered'),
        ('cancelled','Cancelled')
    ]

    customer = models.ForeignKey(Customer, on_delete=models.CASCADE, default=None)
    # cake_customization = models.ForeignKey(CakeCustomization, on_delete=models.CASCADE, default=None,null=True,blank=T
    items = models.ManyToManyField(Cart, related_name='order')
    total_quantity = models.PositiveIntegerField(default=0)  # PositiveIntegerField for quantity
    total_price = models.DecimalField(max_digits=20, decimal_places=2, default=0)  # DecimalField for total_price
    order_date = models.DateTimeField(auto_now_add=True)
    delivery_address = models.TextField(max_length=225, default=None)
    store = models.ForeignKey( to: 'Store', on_delete=models.CASCADE, related_name='orders', default=1)
    order_status = models.CharField(max_length=50, choices=Order_status, default='pending')
    payment_status = models.CharField(max_length=50, choices=Payment_status, default='pending')
    payment_method = models.CharField(max_length=50, choices=Payment_method, default='unknown')
    # ip_address = models.GenericIPAddressField(null=True, blank=True)
```



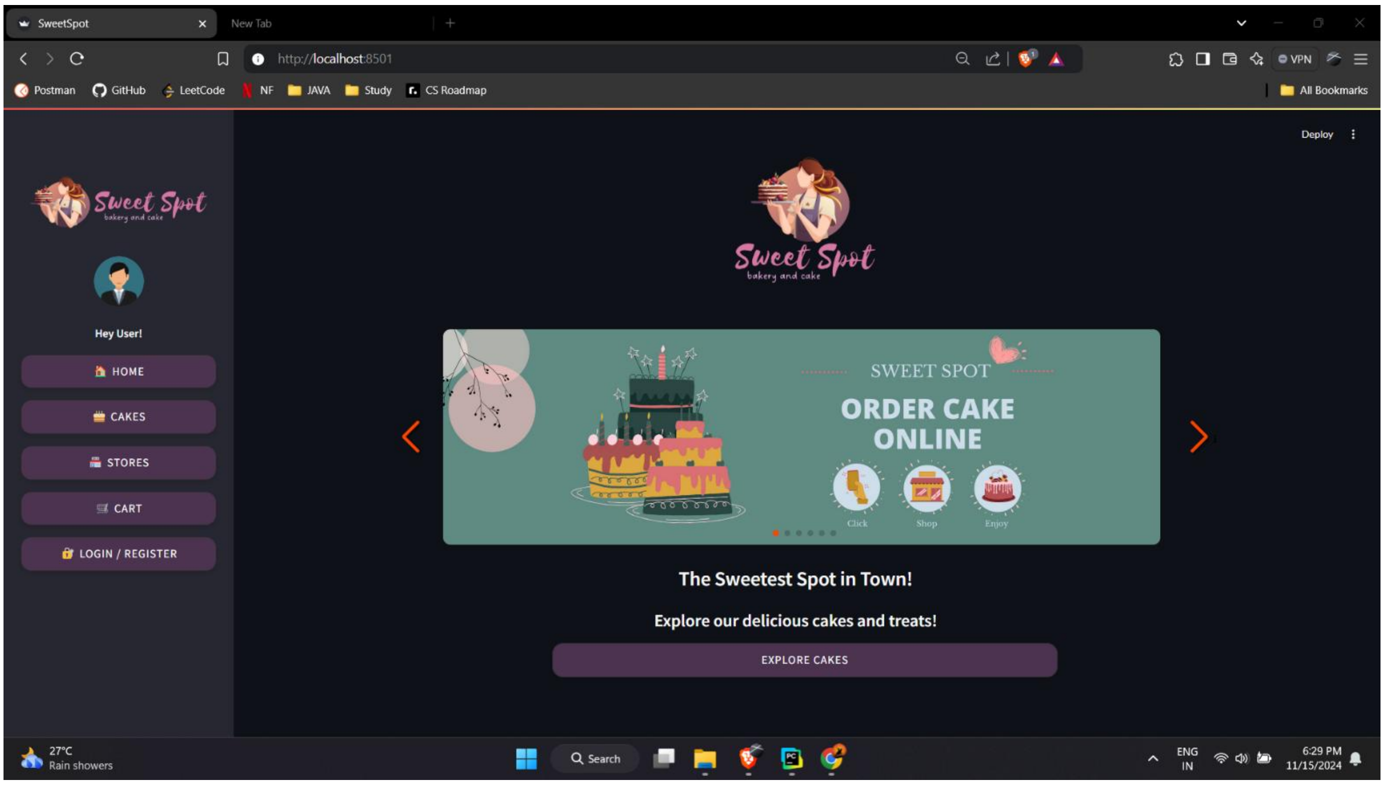
```
class Store(models.Model):  20 usages  Avi
    name = models.CharField(max_length=100)
    city = models.CharField(max_length=100)
    address = models.TextField()
    contact_number = models.CharField(max_length=15)
    email = models.EmailField()
    description = models.TextField()
    store_image = models.ImageField(upload_to='stores/', null=True, blank=True, default='stores/back.png')

    def __str__(self):  Avi
        return self.name

# class Notification(models.Model):
#     user = models.ForeignKey(User, on_delete=models.CASCADE)
#     message = models.TextField()
#     created_at = models.DateTimeField(auto_now_add=True)
#     is_read = models.BooleanField(default=False)

```



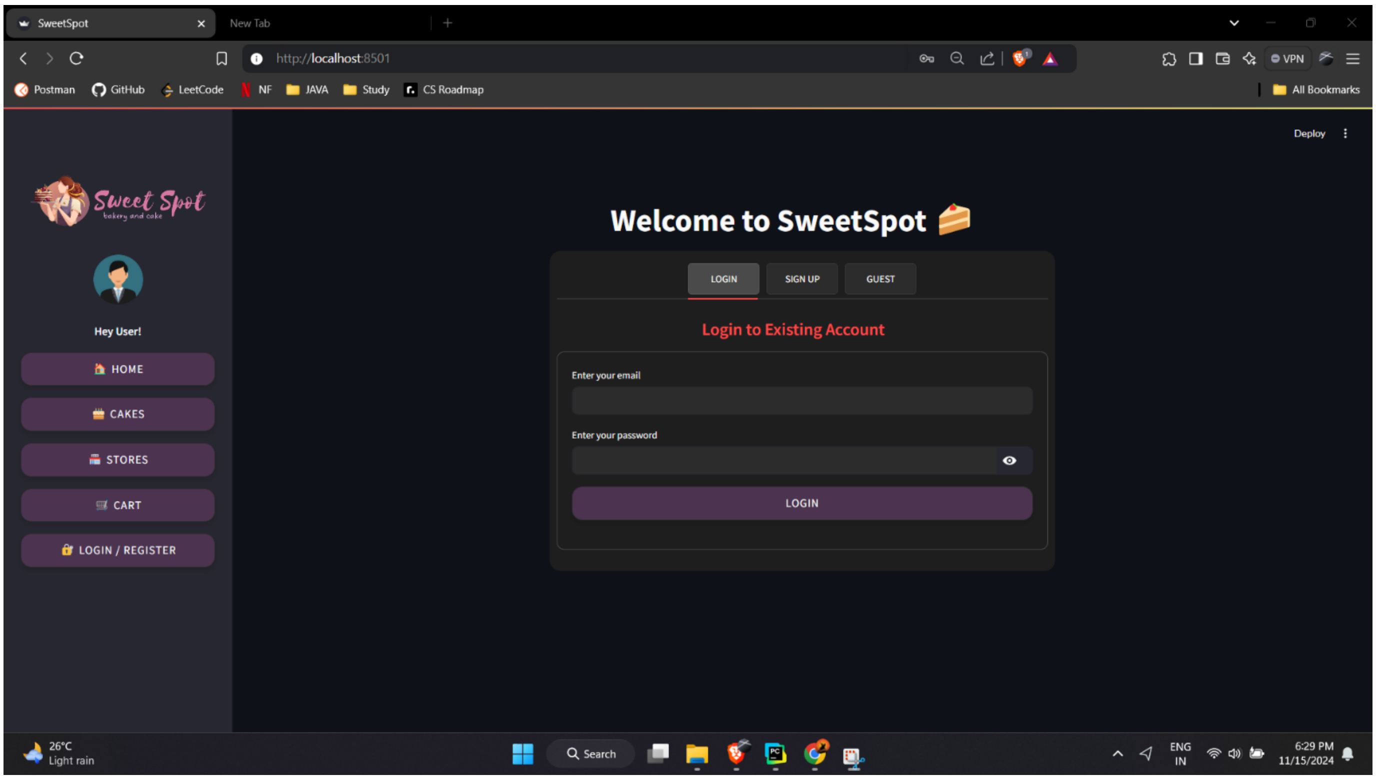
```
from django.urls import path, include
from rest_framework.routers import DefaultRouter
from django.conf.urls.static import static
from django.conf import settings
from .views import CustomerViewSet, PaymentView, UserStoreViewSet, CakeViewSet, CakeCustomizationViewSet, CartViewSet, O

from rest_framework_simplejwt.views import (
    TokenObtainPairView,
    TokenRefreshView,
)

# Create a router and register our viewsets with it
router = DefaultRouter()
router.register( prefix: r'customers', CustomerViewSet)
router.register( prefix: r'cakes', CakeViewSet)
router.register( prefix: r'cake-customizations', CakeCustomizationViewSet)
router.register( prefix: r'carts', CartViewSet)
router.register( prefix: r'orders', OrderViewSet)
router.register( prefix: r'add-cake-to-cart', AddCakeToCartViewSet, basename='add-cake-to-cart')
router.register( prefix: r'stores', UserStoreViewSet, basename='user-store')

# Admin routers
router.register( prefix: r'admin/customers', AdminCustomerViewSet, basename='admin-customer')
router.register( prefix: r'admin/cakes', AdminCakeViewSet, basename='admin-cake')
router.register( prefix: r'admin/cake_customizations', AdminCakeCustomizationViewSet, basename='admin-cake_customization')
router.register( prefix: r'admin/orders', AdminOrderViewSet, basename='admin-order')
router.register( prefix: r'admin/stores', AdminStoreViewSet, basename='admin-store')

urlpatterns = [
    path('', include(router.urls)),
    path('token/', TokenObtainPairView.as_view(), name='token_obtain_pair'),
    path('token/refresh/', TokenRefreshView.as_view(), name='token_refresh'),
    path('payment/', PaymentView.as_view(), name='payment'),
]

if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)

```

Appendix C: Research References

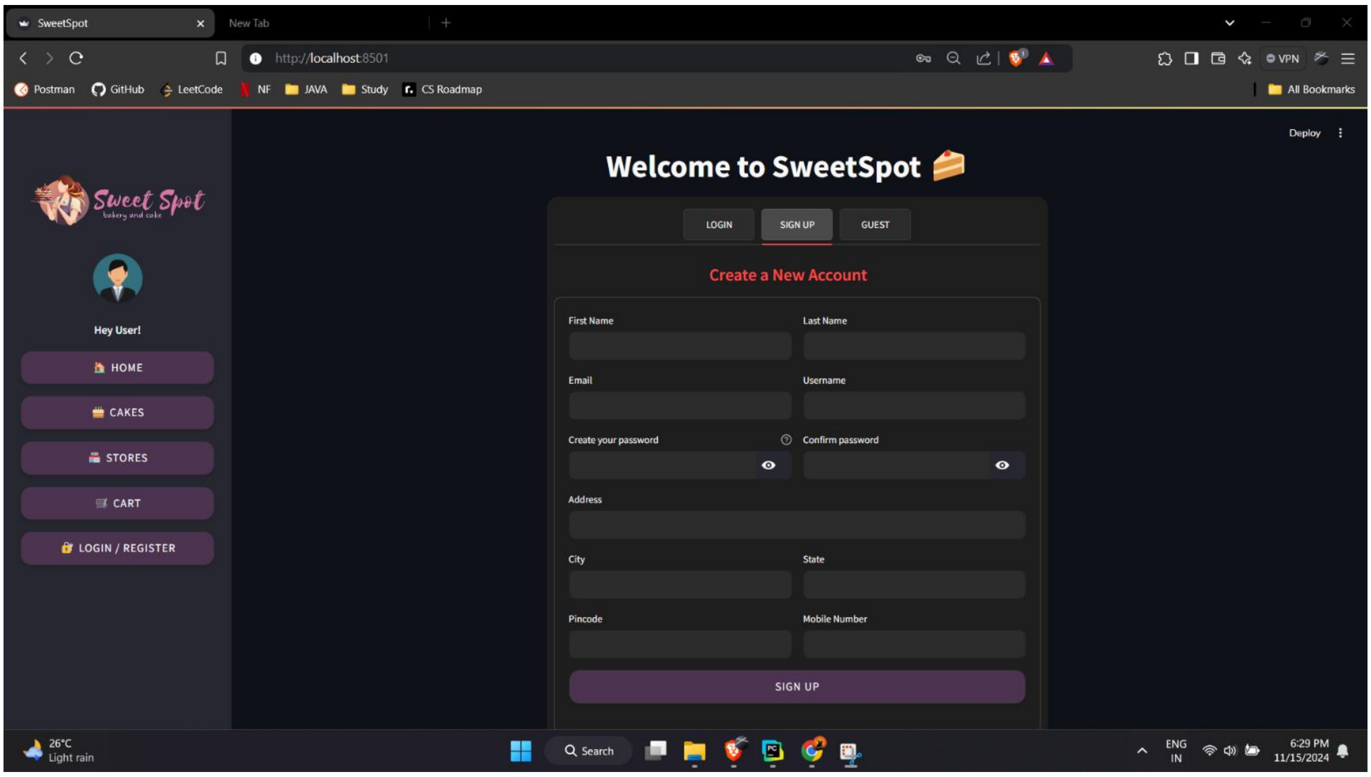
- ❑ **Official Django Documentation**: https://docs.djangoproject.com/
- ❑ **PostgreSQL Best Practices**: https://www.postgresql.org/docs/
- ❑ **UI/UX Design Principles**: Research on user-friendly design patterns for e-commerce platforms.
- ❑ **Online Cake Market Trends**: Industry reports detailing customer preferences and market growth for online cake delivery services.



Home Page



Login Page



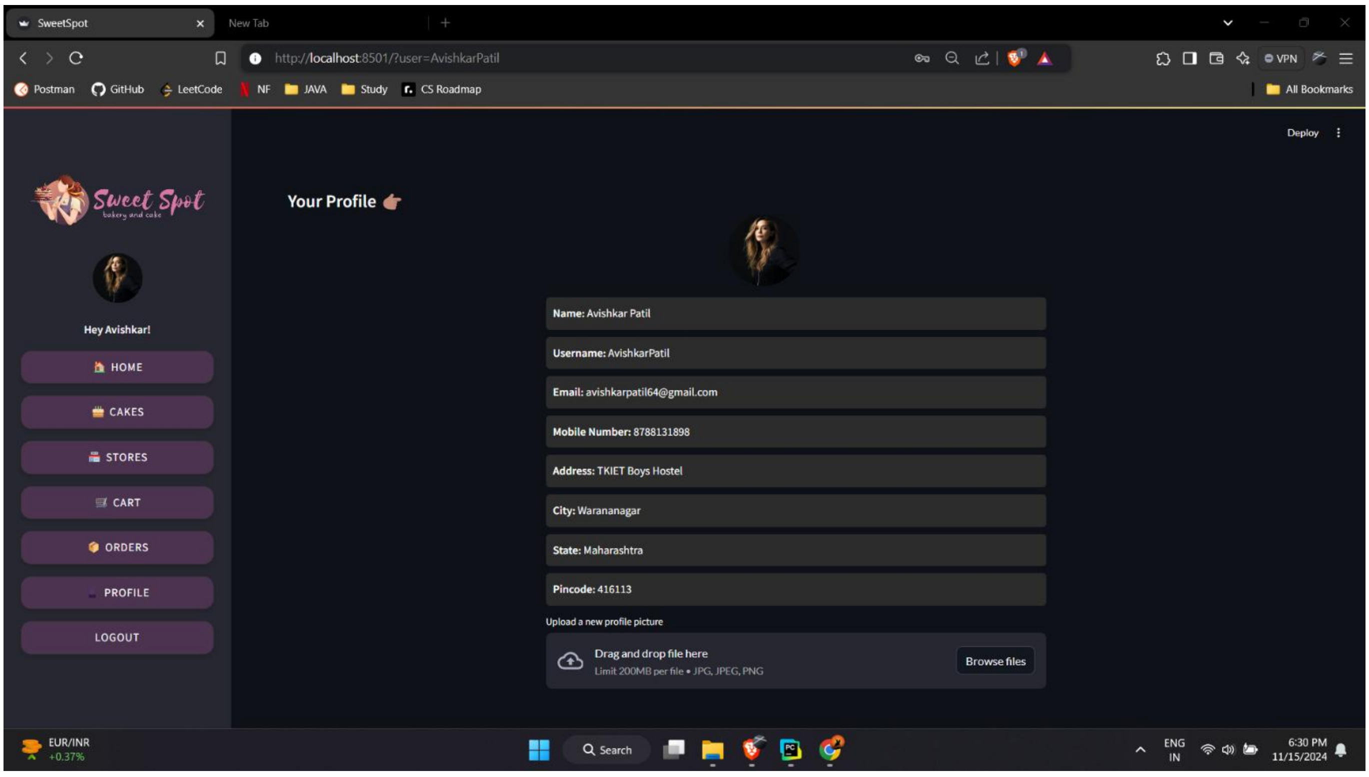
Welcome to SweetSpot 🍰

Create a New Account

First Name | Last Name
Email | Username
Create your password | Confirm password
Address
City | State
Pincode | Mobile Number

SIGN UP

Registration Page



Your Profile 👉

Name: Avishkar Patil

Username: AvishkarPatil

Email: avishkarpatil64@gmail.com

Mobile Number: 8788131898

Address: TKIET Boys Hostel

City: Warananagar

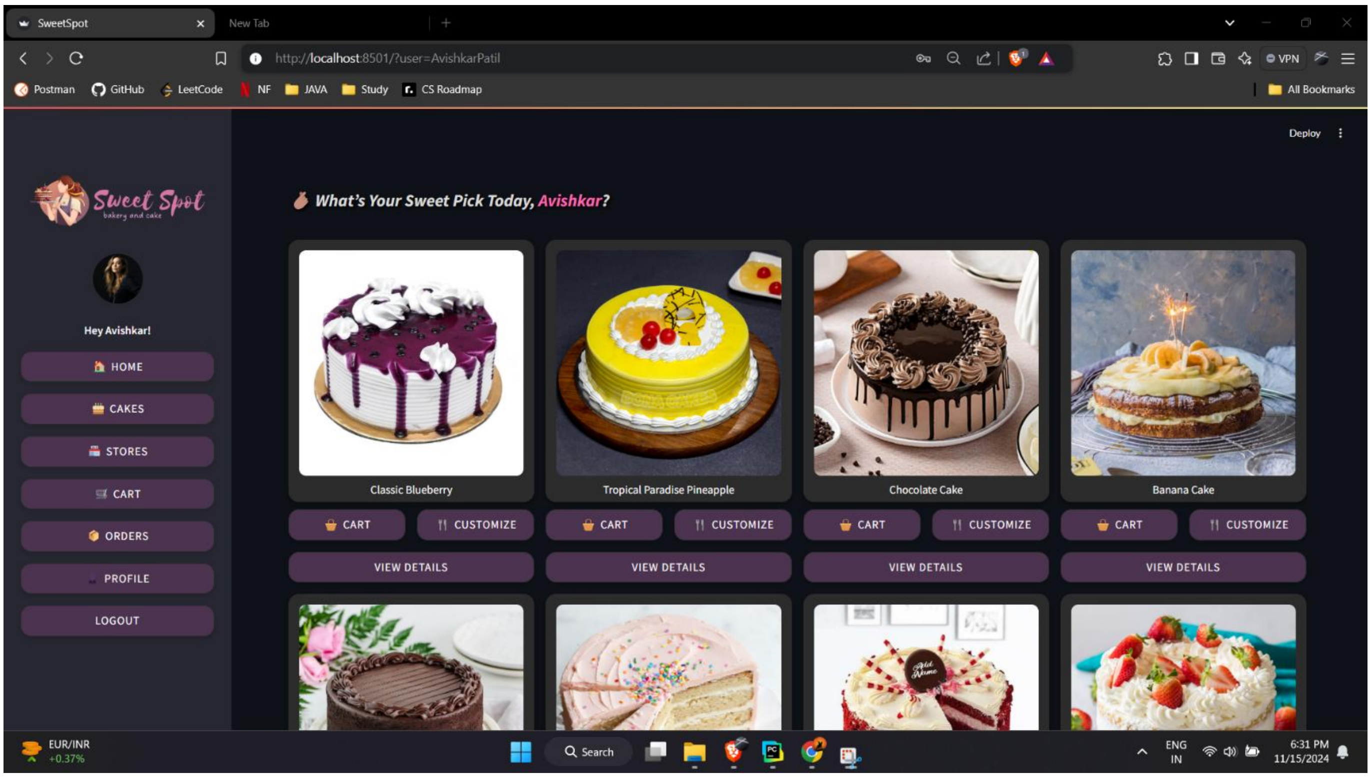
State: Maharashtra

Pincode: 416113

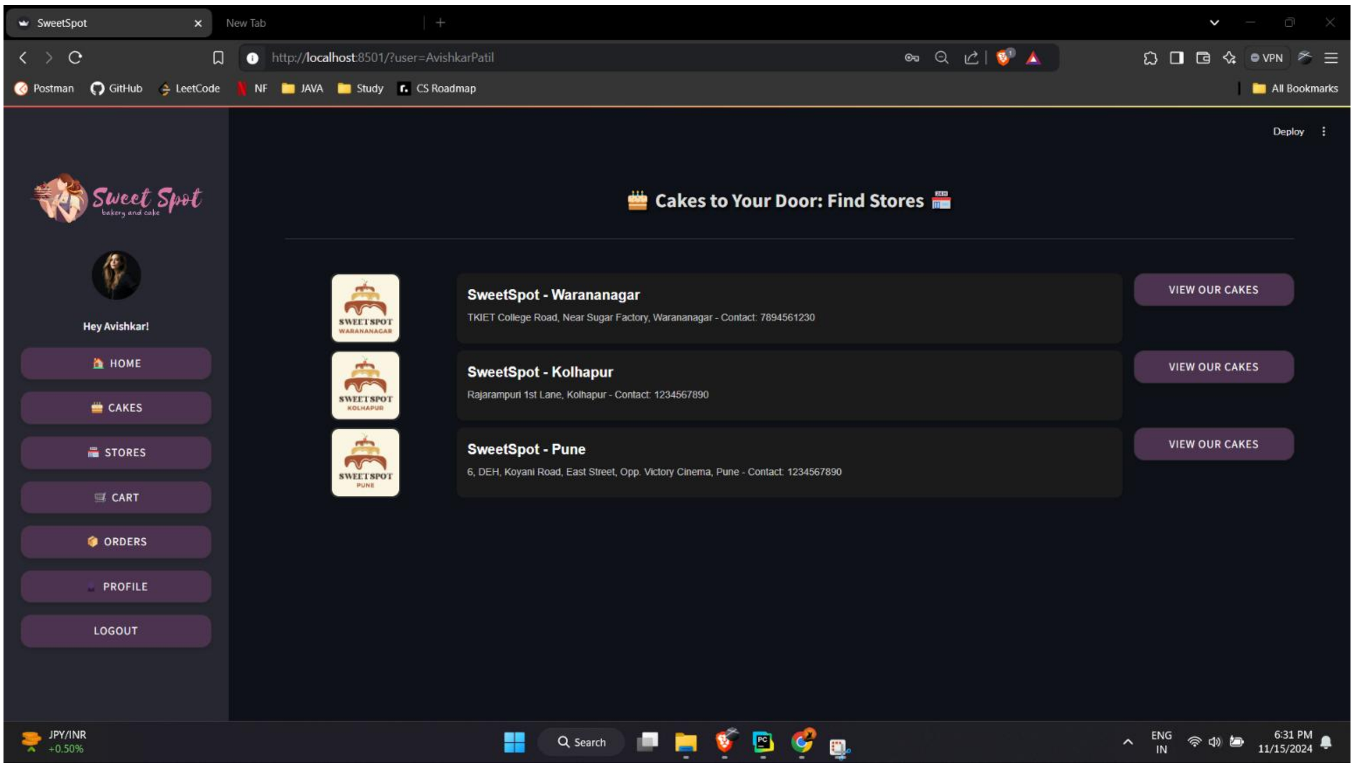
Upload a new profile picture

Drag and drop file here
Limit 200MB per file • JPG, JPEG, PNG
Browse files

Profile Page



Cakes List



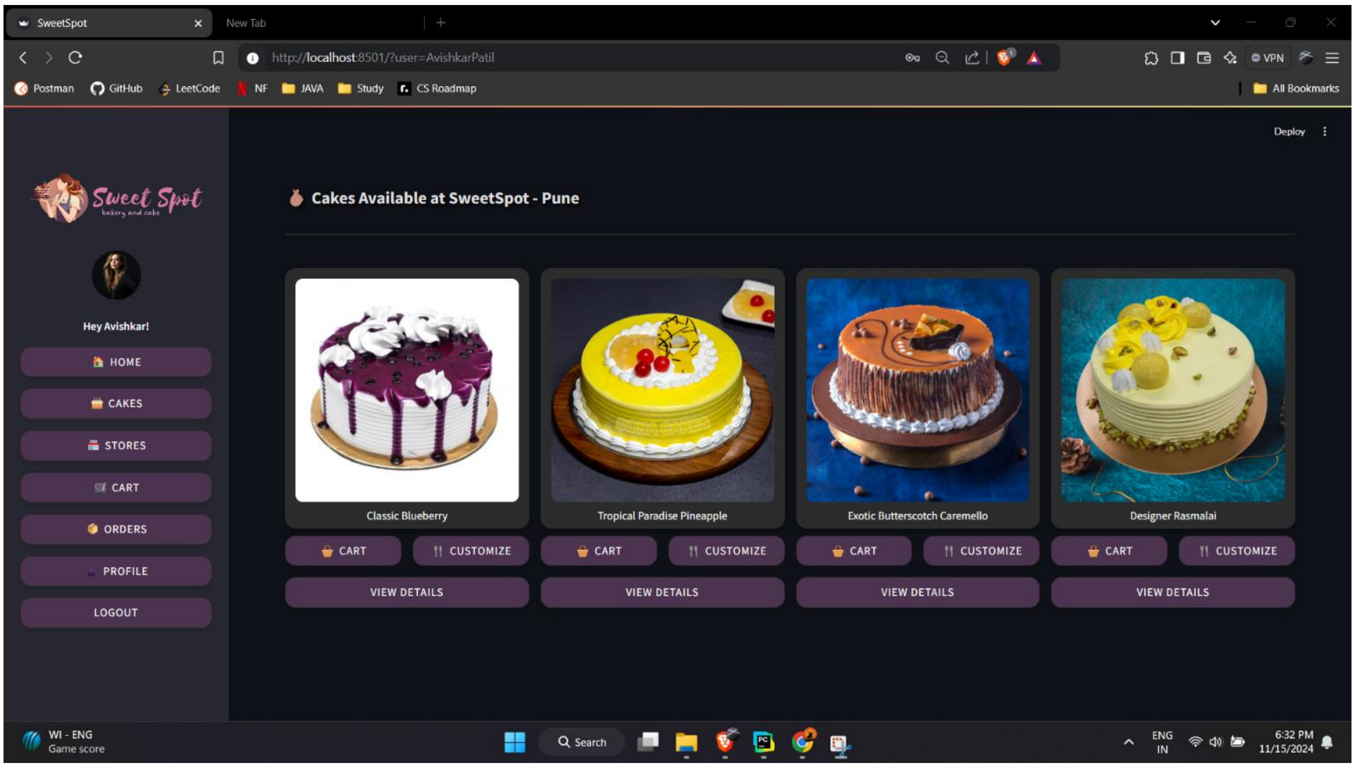
Cakes to Your Door: Find Stores

SweetSpot - Warananagar
TKIET College Road, Near Sugar Factory, Warananagar - Contact: 7894561230

SweetSpot - Kolhapur
Rajarampuri 1st Lane, Kolhapur - Contact: 1234567890

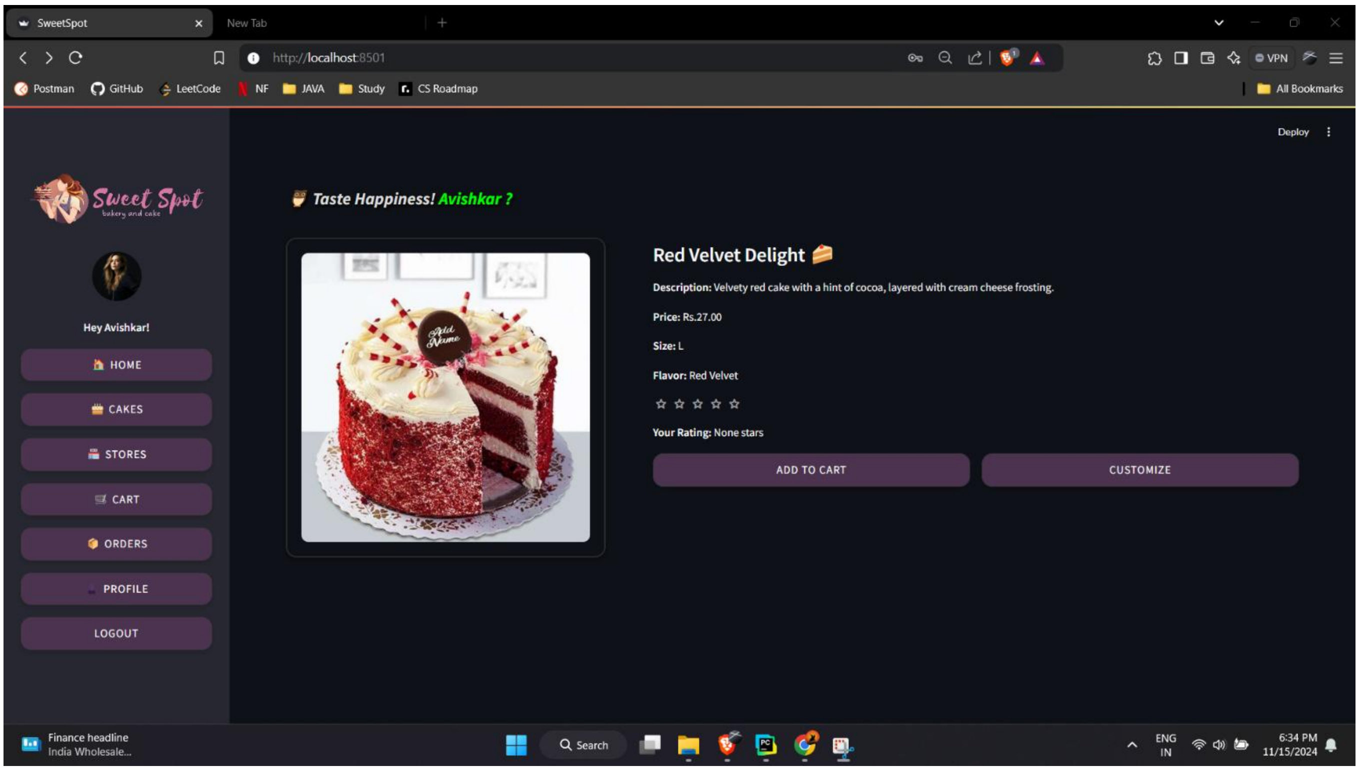
SweetSpot - Pune
6, DEH, Koyani Road, East Street, Opp. Victory Cinema, Pune - Contact: 1234567890

Stores List


SweetSpot
New Tab
http://localhost:8501/?user=AvishkarPatil
Postman GitHub LeetCode NF JAVA Study CS Roadmap All Bookmarks
Deploy
Sweet Spot
Hey Avishkar!
HOME
CAKES
STORES
CART
ORDERS
PROFILE
LOGOUT
Classic Blueberry
Tropical Paradise Pineapple
Exotic Butterscotch Caremello
Designer Rasmalai
CART
CUSTOMIZE
VIEW DETAILS

Cakes Available at SweetSpot - Pune

Cakes In Store



Taste Happiness! Avishkar ?

Red Velvet Delight

Description: Velvety red cake with a hint of cocoa, layered with cream cheese frosting.

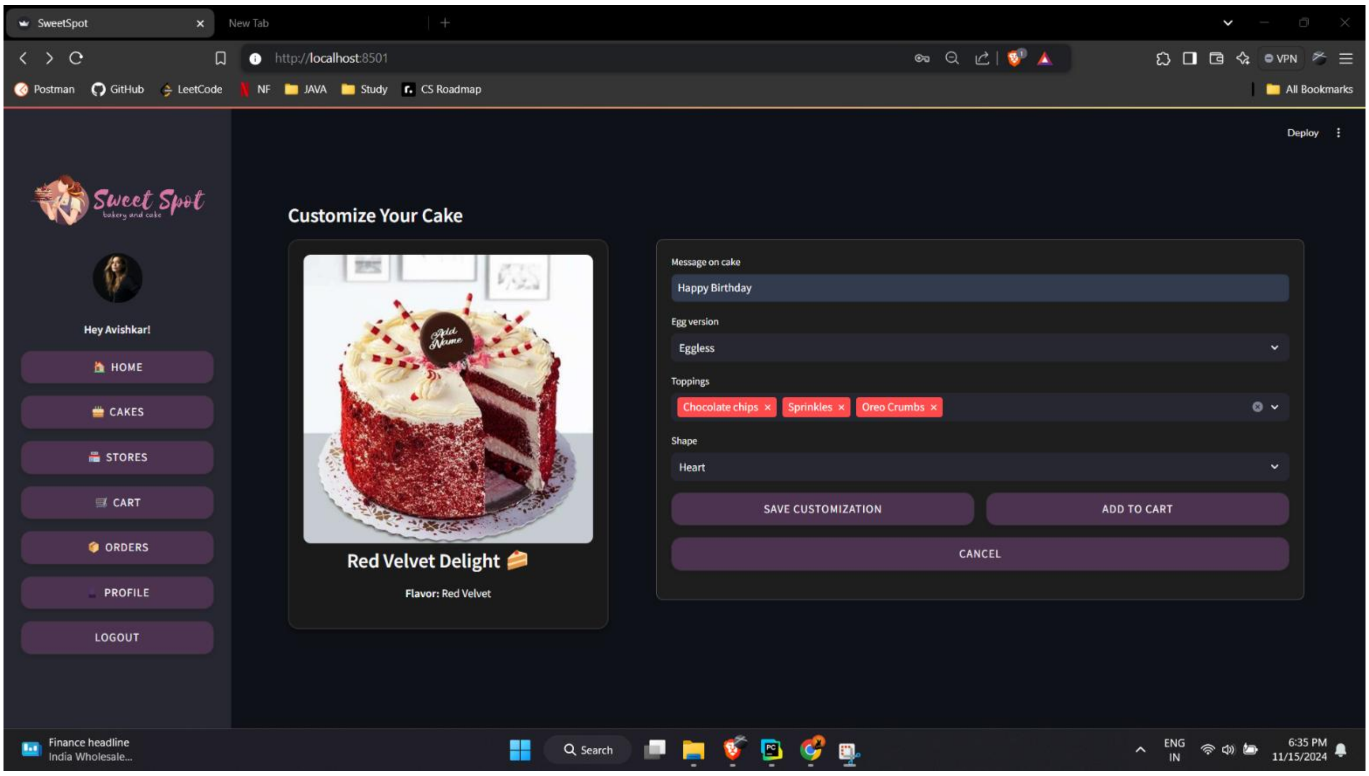
Price: Rs.27.00

Size: L

Flavor: Red Velvet

Your Rating: None stars

Cakes Details



Customize Your Cake

Red Velvet Delight 🍰

Flavor: Red Velvet

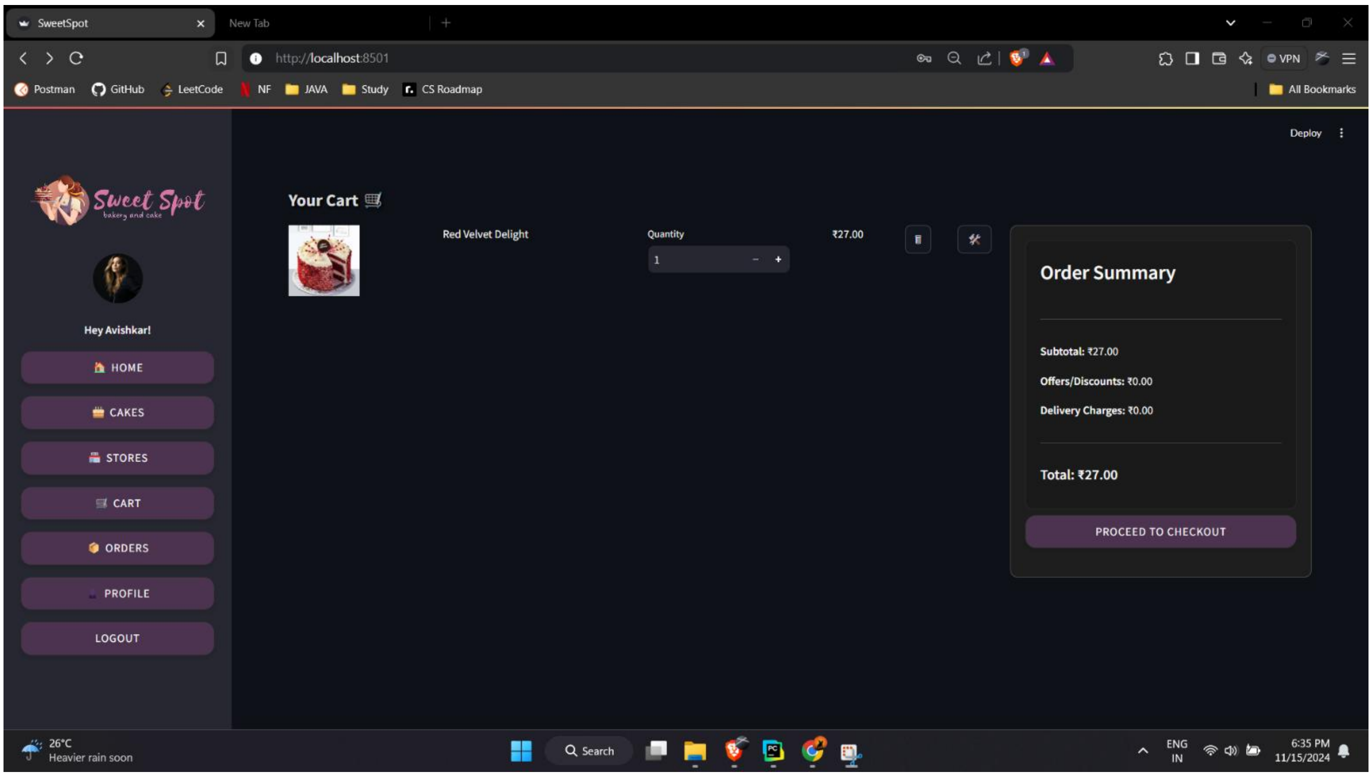
Message on cake: Happy Birthday

Egg version: Eggless

Toppings: Chocolate chips ×, Sprinkles ×, Oreo Crumbs ×

Shape: Heart

Cakes Customization



Your Cart 🛒

Red Velvet Delight | Quantity: 1 | ₹27.00

Order Summary

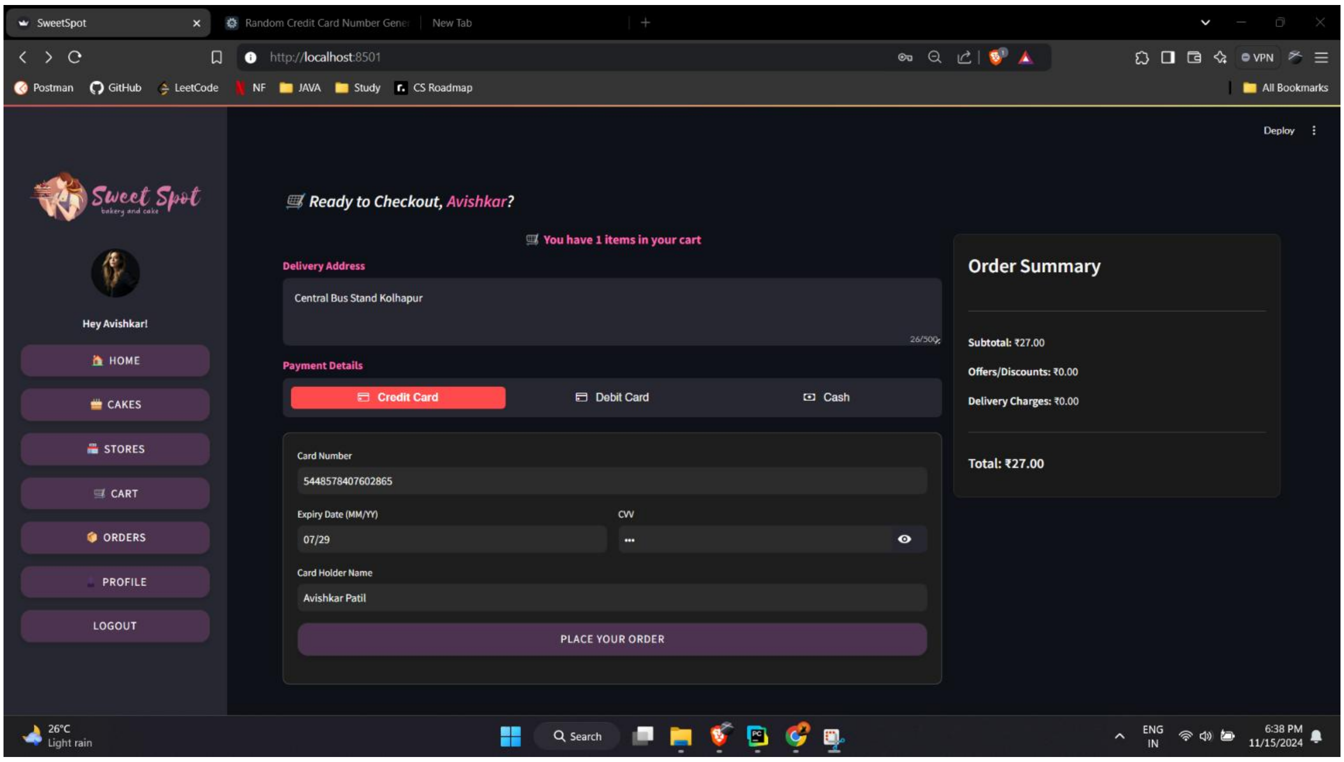
Subtotal: ₹27.00

Offers/Discounts: ₹0.00

Delivery Charges: ₹0.00

Total: ₹27.00

Cakes In Cart



Ready to Checkout, Avishkar?

You have 1 items in your cart

Delivery Address

Central Bus Stand Kolhapur

26/500

Payment Details

Credit Card | Debit Card | Cash

Card Number

5448578407602865

Expiry Date (MM/YY)

07/29

CVV

•••

Card Holder Name

Avishkar Patil

PLACE YOUR ORDER

Order Summary

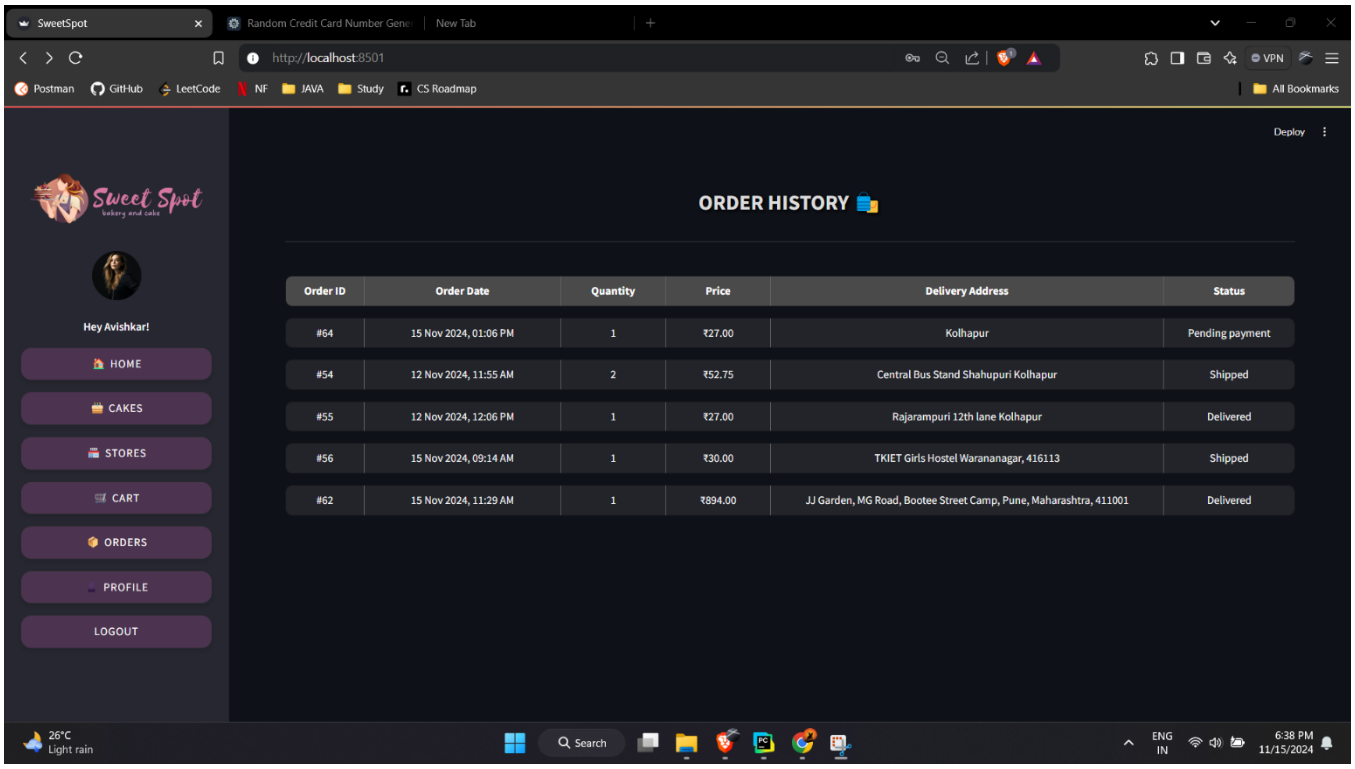
Subtotal: ₹27.00

Offers/Discounts: ₹0.00

Delivery Charges: ₹0.00

Total: ₹27.00

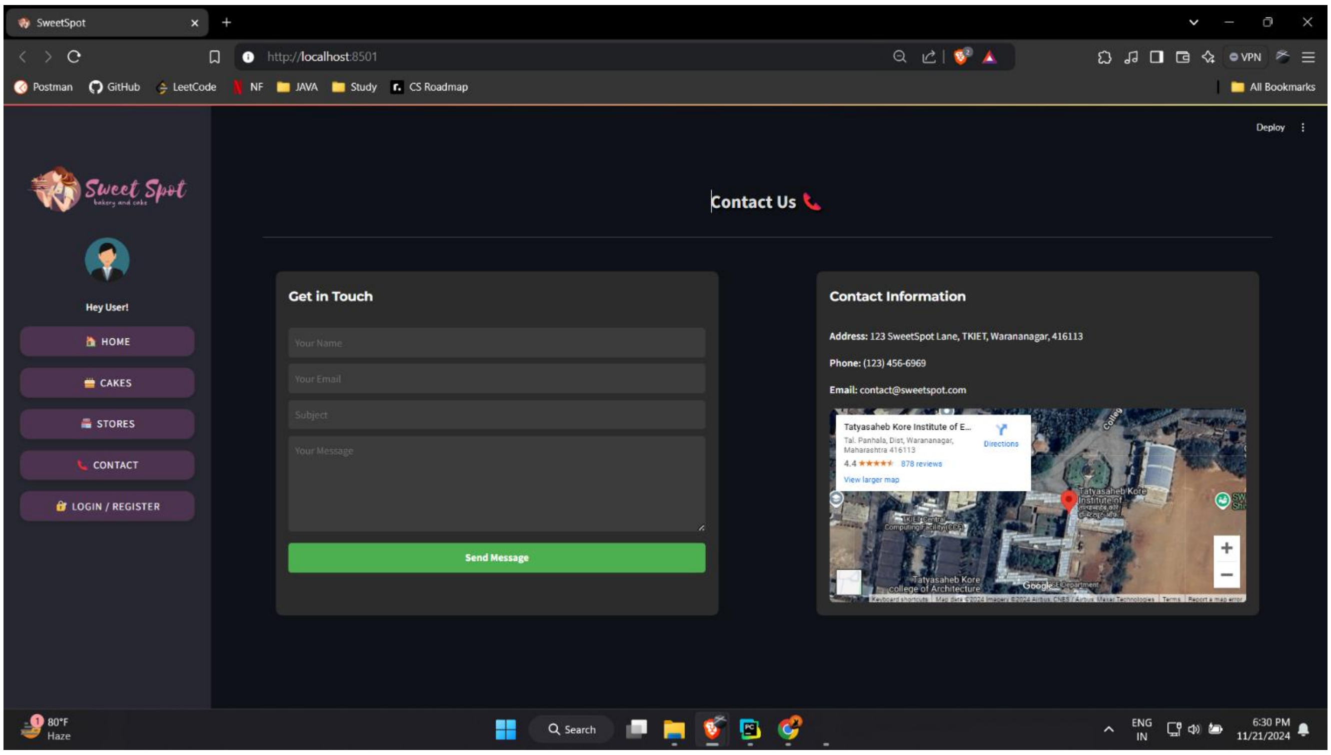
Checkout Page



ORDER HISTORY

Order ID	Order Date	Quantity	Price	Delivery Address	Status
#64	15 Nov 2024, 01:06 PM	1	₹27.00	Kolhapur	Pending payment
#54	12 Nov 2024, 11:55 AM	2	₹52.75	Central Bus Stand Shahupuri Kolhapur	Shipped
#55	12 Nov 2024, 12:06 PM	1	₹27.00	Rajarampuri 12th lane Kolhapur	Delivered
#56	15 Nov 2024, 09:14 AM	1	₹30.00	TKIET Girls Hostel Warananagar, 416113	Shipped
#62	15 Nov 2024, 11:29 AM	1	₹894.00	JJ Garden, MG Road, Bootee Street Camp, Pune, Maharashtra, 411001	Delivered

Orders History Page



Contact Us

Get in Touch

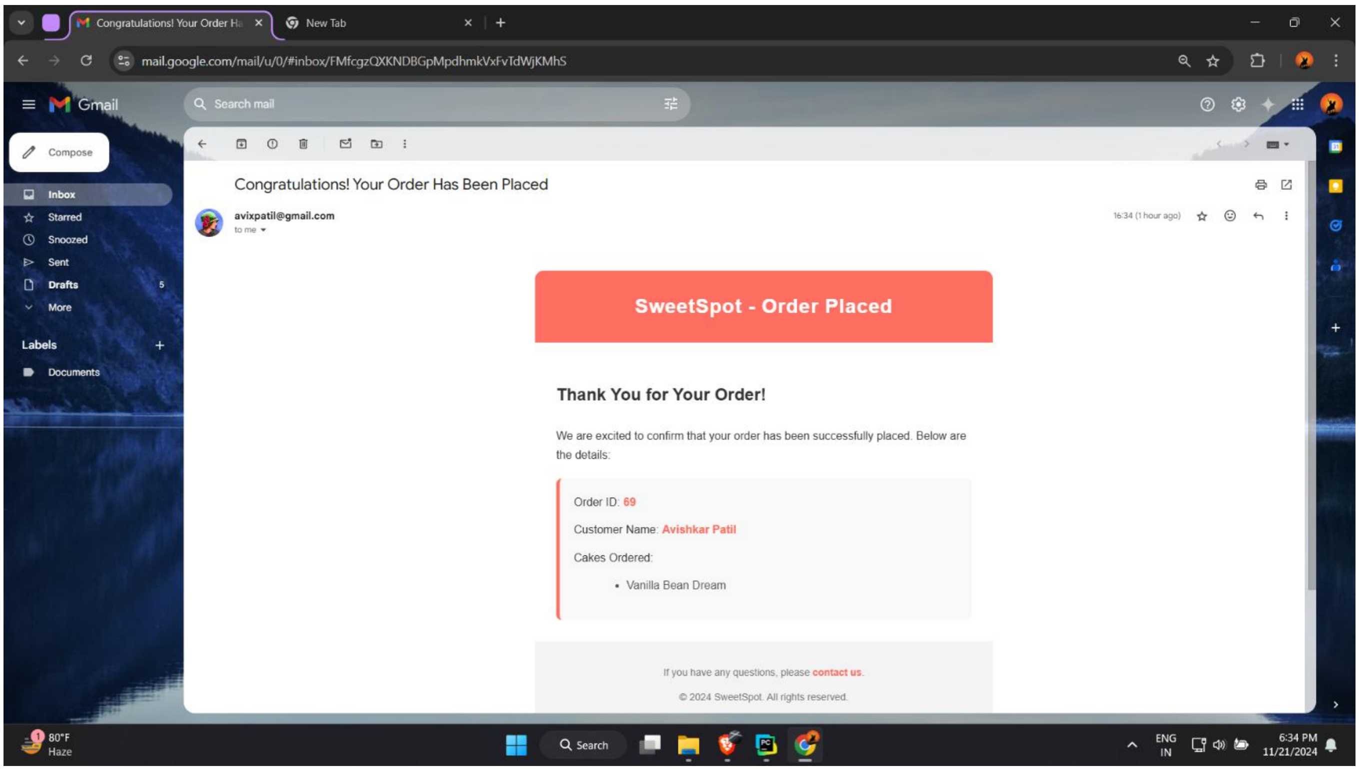
Contact Information

Address: 123 SweetSpot Lane, TKIET, Warananagar, 416113

Phone: (123) 456-6969

Email: contact@sweetspot.com

Contact Us Page



Congratulations! Your Order Has Been Placed

avixpatil@gmail.com
to me

16:34 (1 hour ago)

SweetSpot - Order Placed

Thank You for Your Order!

We are excited to confirm that your order has been successfully placed. Below are the details:

Order ID: 69

Customer Name: Avishkar Patil

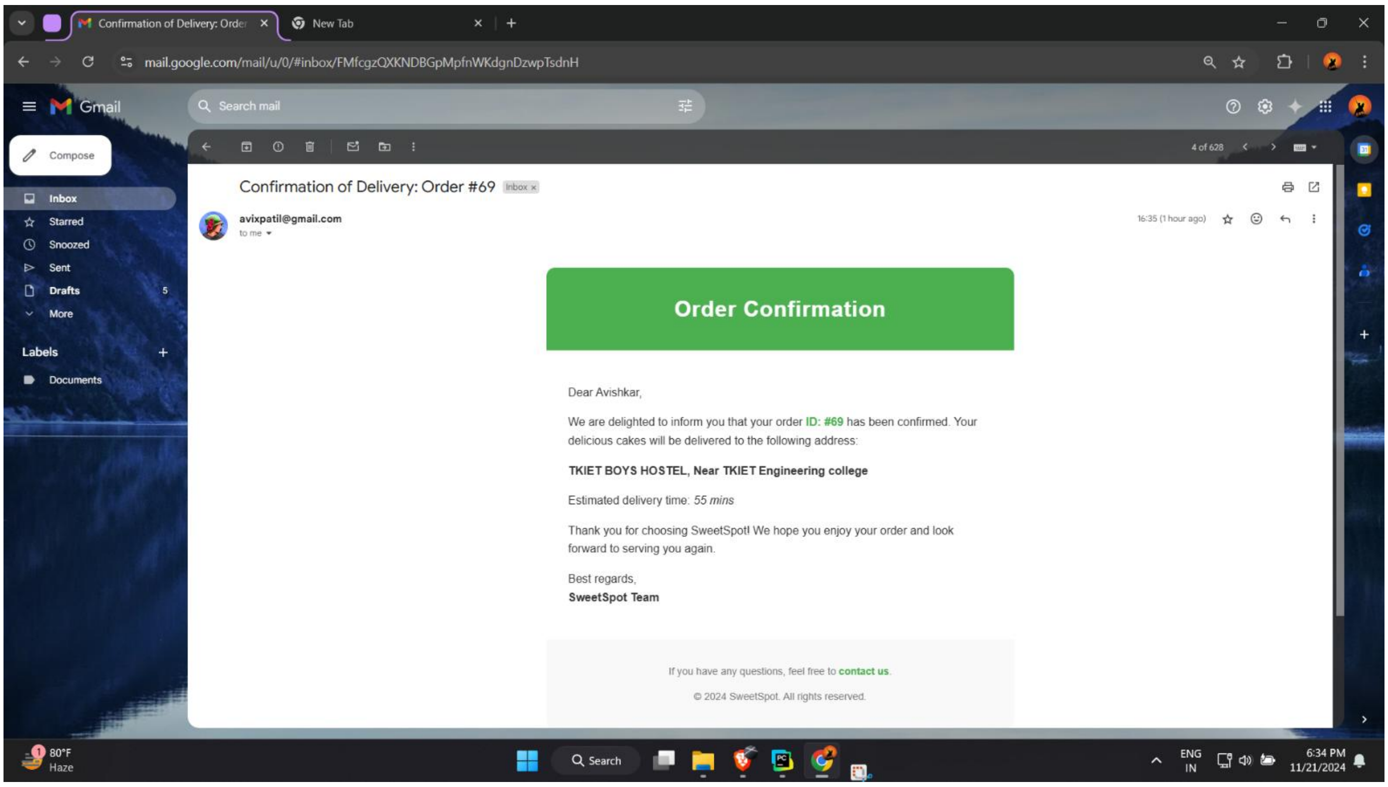
Cakes Ordered:

- Vanilla Bean Dream

If you have any questions, please contact us.

© 2024 SweetSpot. All rights reserved.

Order Placed Email Confirmation



Confirmation of Delivery: Order #69

avixpatil@gmail.com

to me

16:35 (1 hour ago)

Order Confirmation

Dear Avishkar,

We are delighted to inform you that your order ID: #69 has been confirmed. Your delicious cakes will be delivered to the following address:

TKIET BOYS HOSTEL, Near TKIET Engineering college

Estimated delivery time: *55 mins*

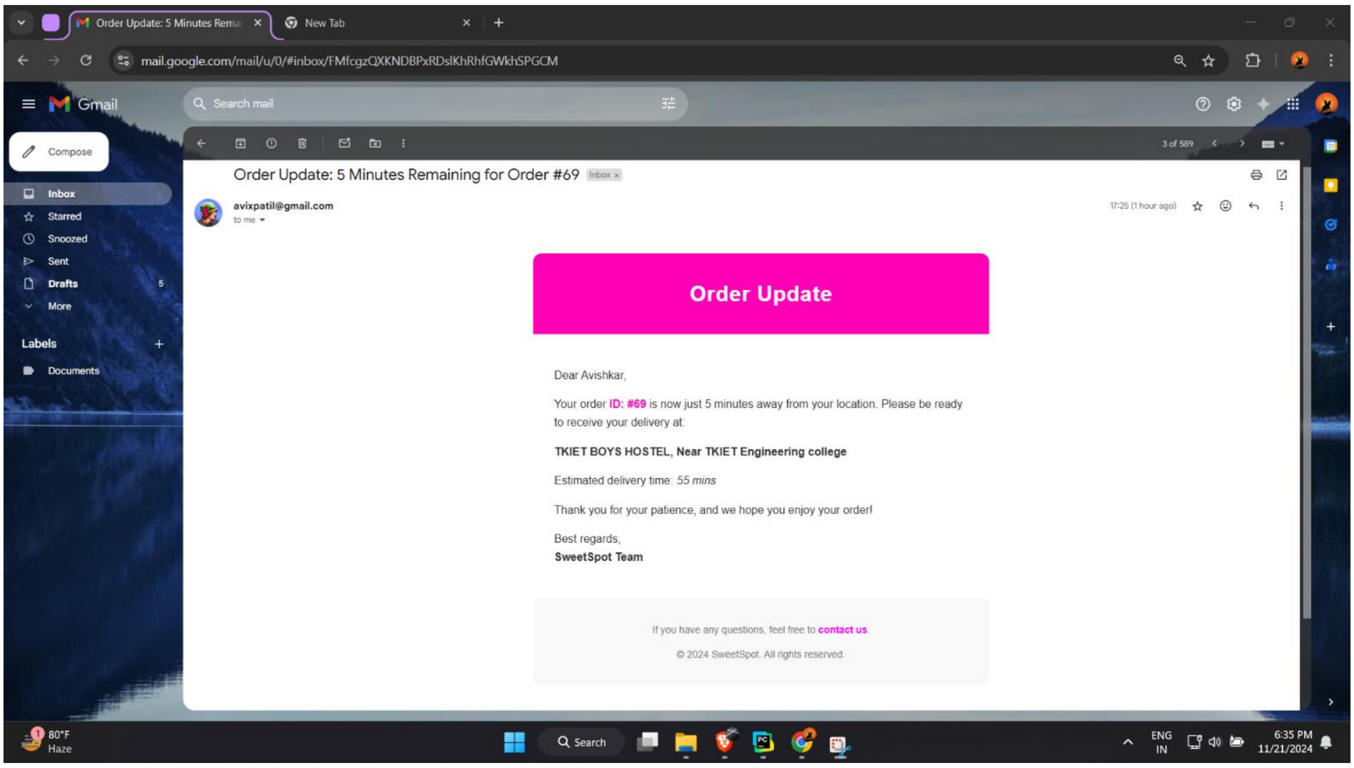
Thank you for choosing SweetSpot! We hope you enjoy your order and look forward to serving you again.

Best regards,
SweetSpot Team

If you have any questions, feel free to contact us.

© 2024 SweetSpot. All rights reserved.

Order Confirmed Email Received



Order Update: 5 Minutes Remaining for Order #69

avixpatil@gmail.com

to me

17:25 (1 hour ago)

Order Update

Dear Avishkar,

Your order ID: #69 is now just 5 minutes away from your location. Please be ready to receive your delivery at:

TKIET BOYS HOSTEL, Near TKIET Engineering college

Estimated delivery time: *55 mins*

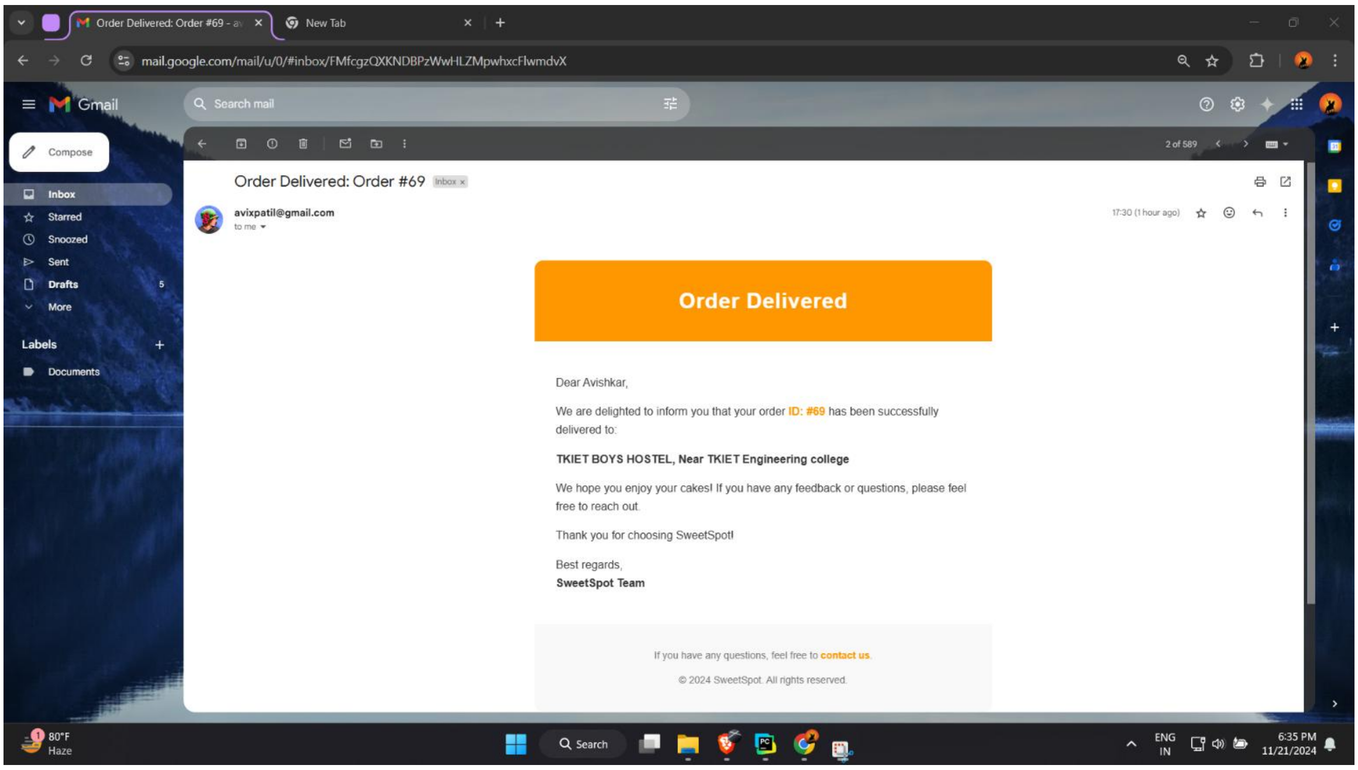
Thank you for your patience, and we hope you enjoy your order!

Best regards,
SweetSpot Team

If you have any questions, feel free to contact us.

© 2024 SweetSpot. All rights reserved.

Order 5min Away from Destination Email Update


Order Delivered: Order #69
New Tab
mail.google.com/mail/u/0/#inbox/FMfcgzQXKNDBPzWwHLZMpwhxcFlwmdvX
Gmail
Search mail
Compose
Inbox
Starred
Snoozed
Sent
Drafts
More
Labels
Documents
2 of 589

Order Delivered: Order #69 Inbox

avixpatil@gmail.com
to me

17:30 (1 hour ago)

Order Delivered

Dear Avishkar,

We are delighted to inform you that your order ID: #69 has been successfully delivered to:

TKIET BOYS HOSTEL, Near TKIET Engineering college

We hope you enjoy your cakes! If you have any feedback or questions, please feel free to reach out.

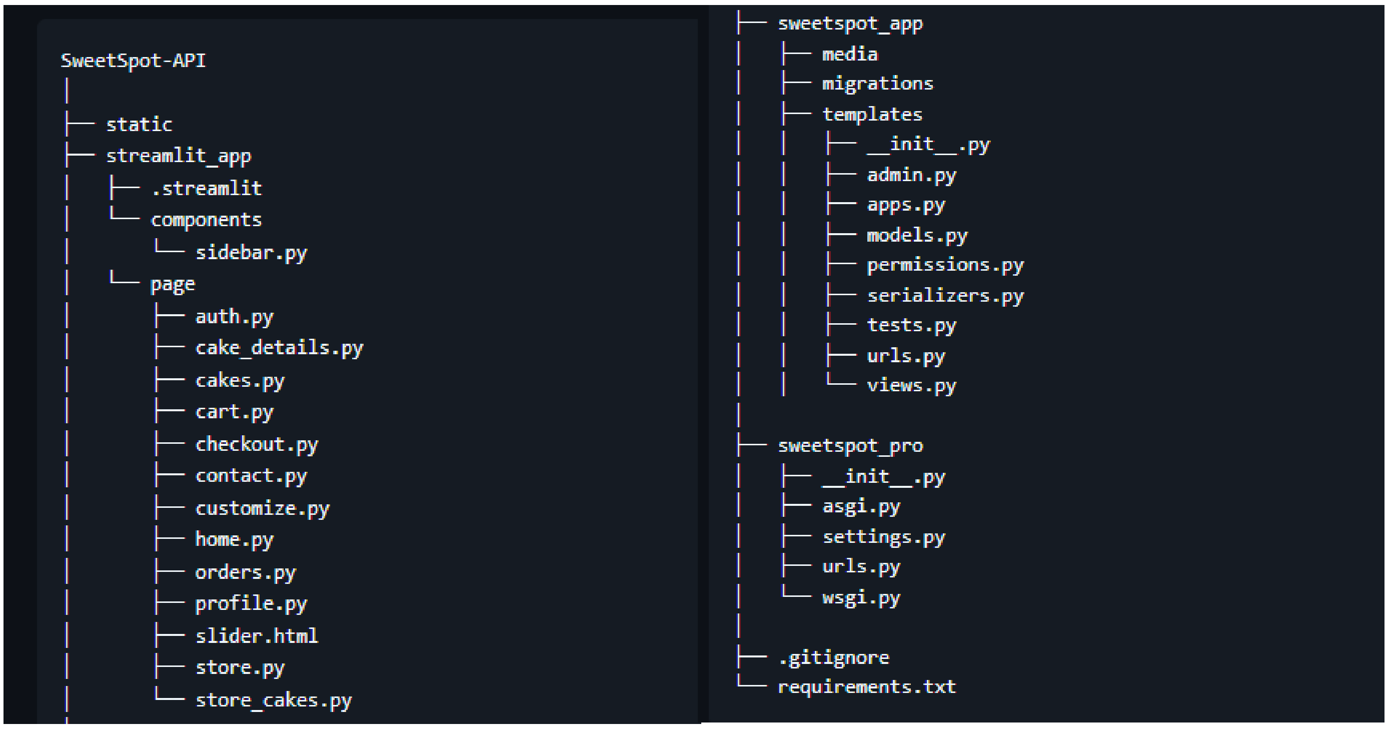
Thank you for choosing SweetSpot!

Best regards,
SweetSpot Team

If you have any questions, feel free to contact us.

© 2024 SweetSpot. All rights reserved.

Order (Cake) is Delivered at the Destination Email Received



```
SweetSpot-API
|
├── static
├── streamlit_app
│   ├── .streamlit
│   └── components
│       └── sidebar.py
│   └── page
│       ├── auth.py
│       ├── cake_details.py
│       ├── cakes.py
│       ├── cart.py
│       ├── checkout.py
│       ├── contact.py
│       ├── customize.py
│       ├── home.py
│       ├── orders.py
│       ├── profile.py
│       ├── slider.html
│       ├── store.py
│       └── store_cakes.py
│
├── sweetspot_app
│   ├── media
│   ├── migrations
│   ├── templates
│   │   ├── __init__.py
│   │   ├── admin.py
│   │   ├── apps.py
│   │   ├── models.py
│   │   ├── permissions.py
│   │   ├── serializers.py
│   │   ├── tests.py
│   │   ├── urls.py
│   │   └── views.py
│
├── sweetspot_pro
│   ├── __init__.py
│   ├── asgi.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
│
├── .gitignore
└── requirements.txt
```

Project File Structure

- Avishkar Patil