Food Ordering Application Documentation

-Mokshi Dharmesh Aya

Overview

This document provides an overview of the Food Ordering Application, including its design, implementation details, instructions on how to run the application, and potential improvements.

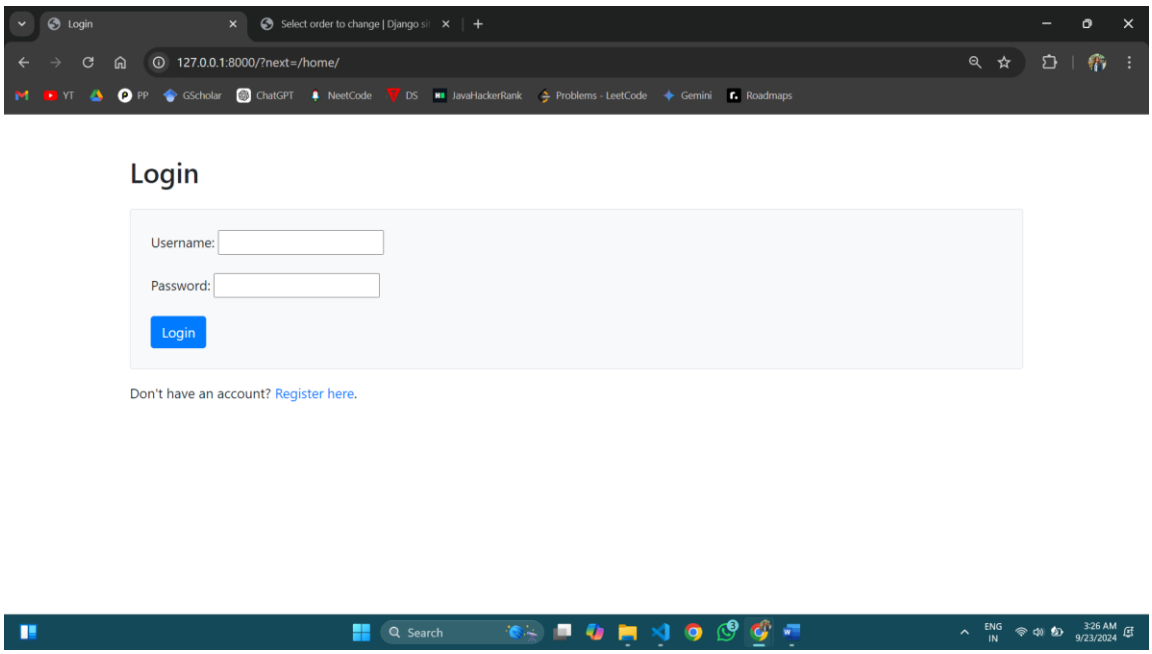
Design

Design Overview

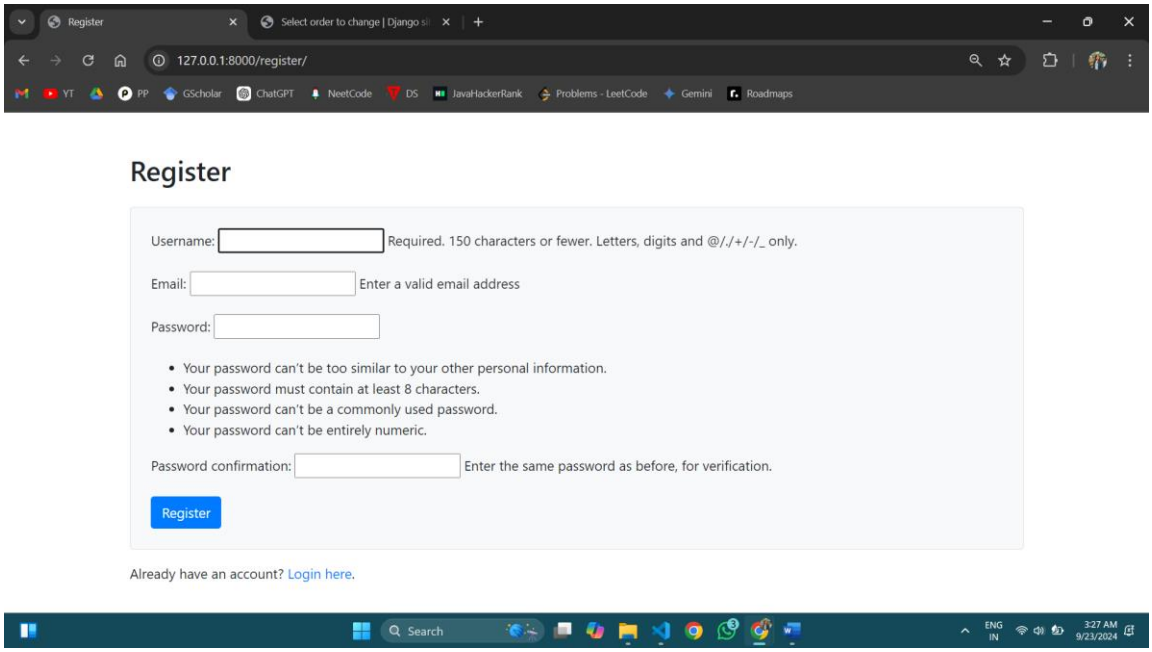
The platform is a full-stack web application built using Django, providing a user-friendly interface for browsing food items, adding them to a cart, and completing orders. Key components of the system include:

- User Authentication: Secure login and registration using Django's built-in authentication.
- Item Browsing: Users can browse items by category (e.g., fruits, vegetables, non-veg).
- Shopping Cart: Users can add items to the cart, view the total cost, and proceed to checkout.
- Checkout: Secure order processing with inventory management.
- Order History: Users can view past orders.
- Admin Features: Manage inventory, update stock, and handle orders.

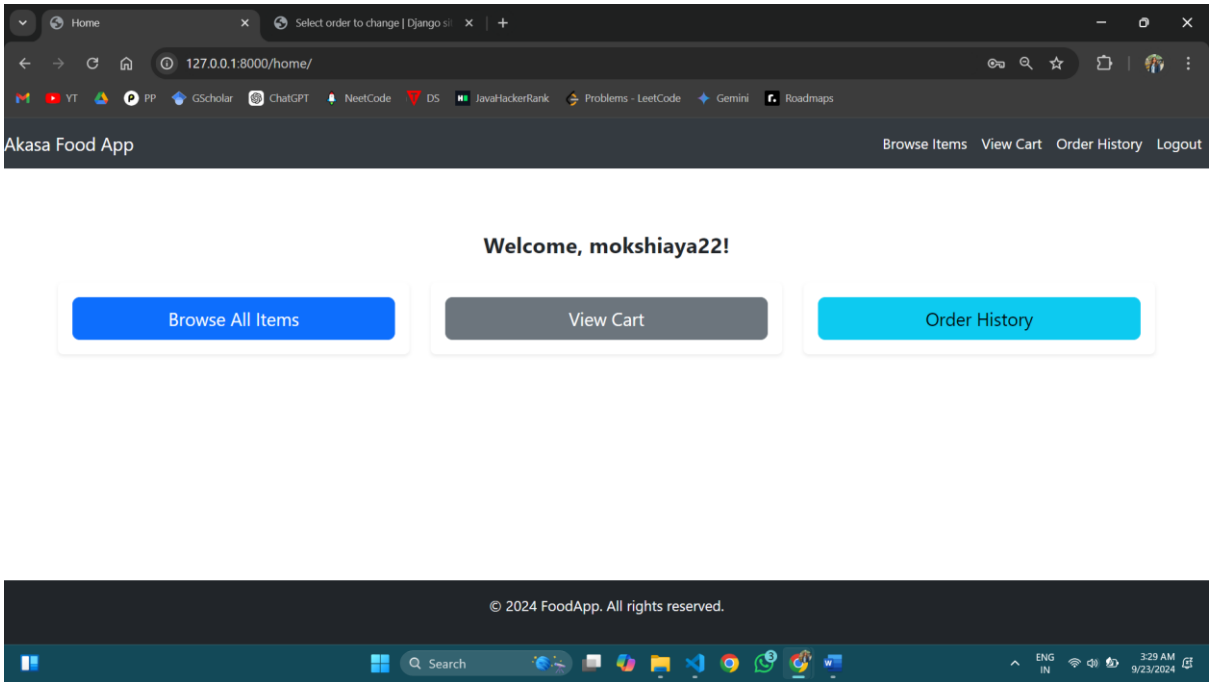
Login page



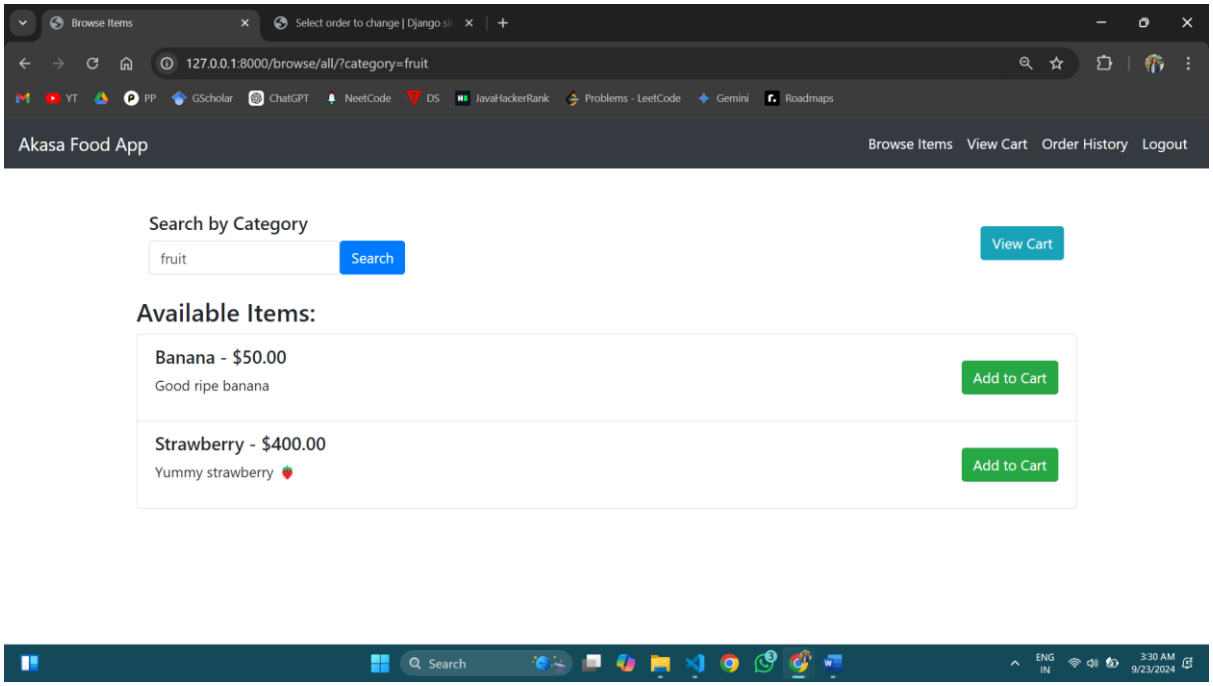
Register page



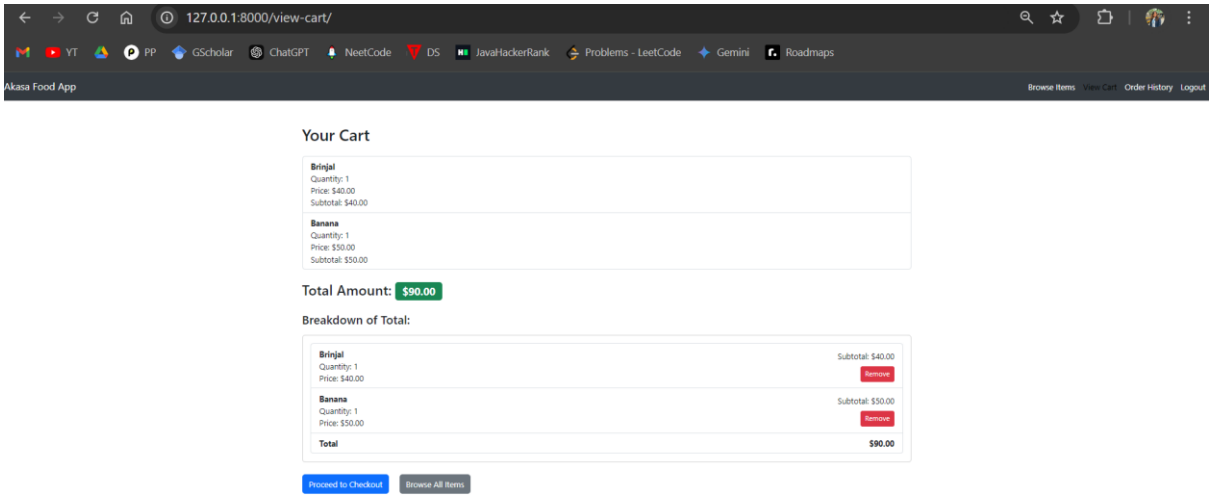
Home page



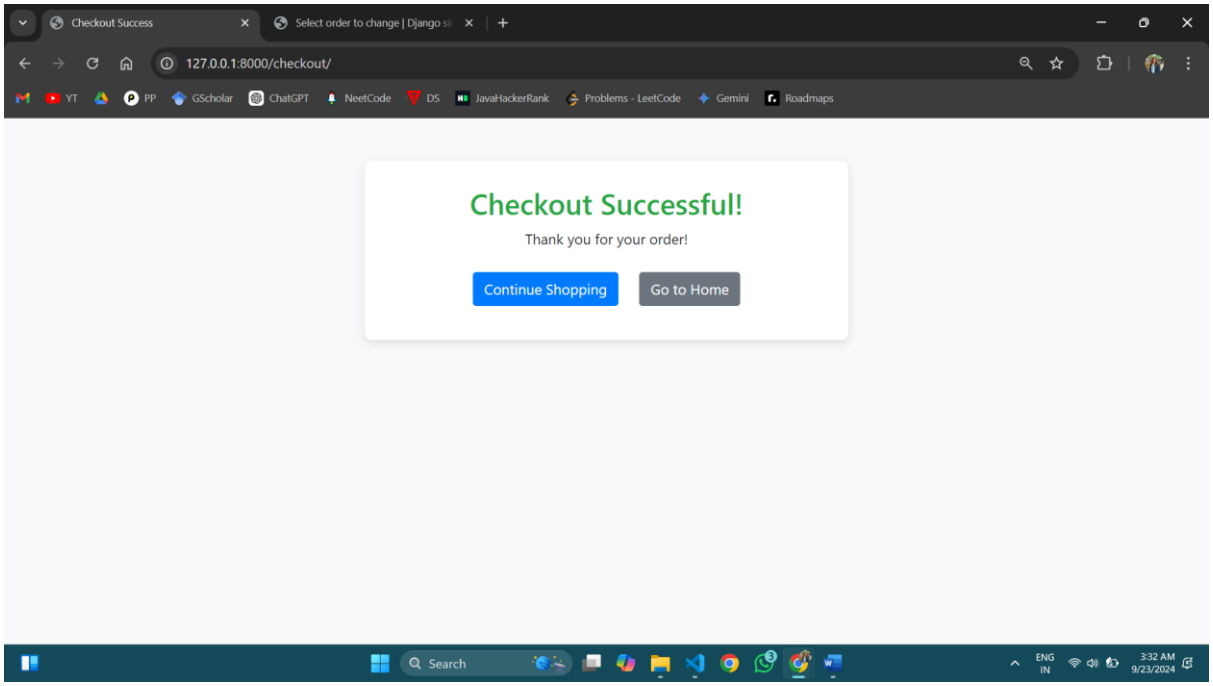
Browse Items page



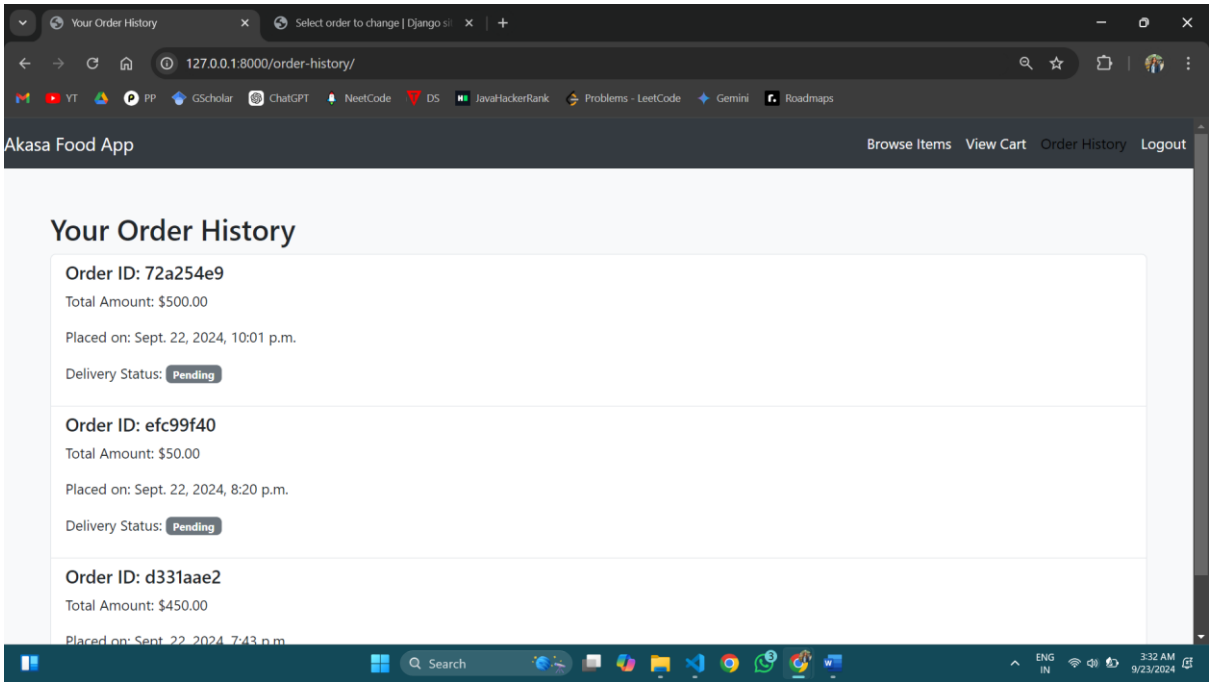
View Cart Page



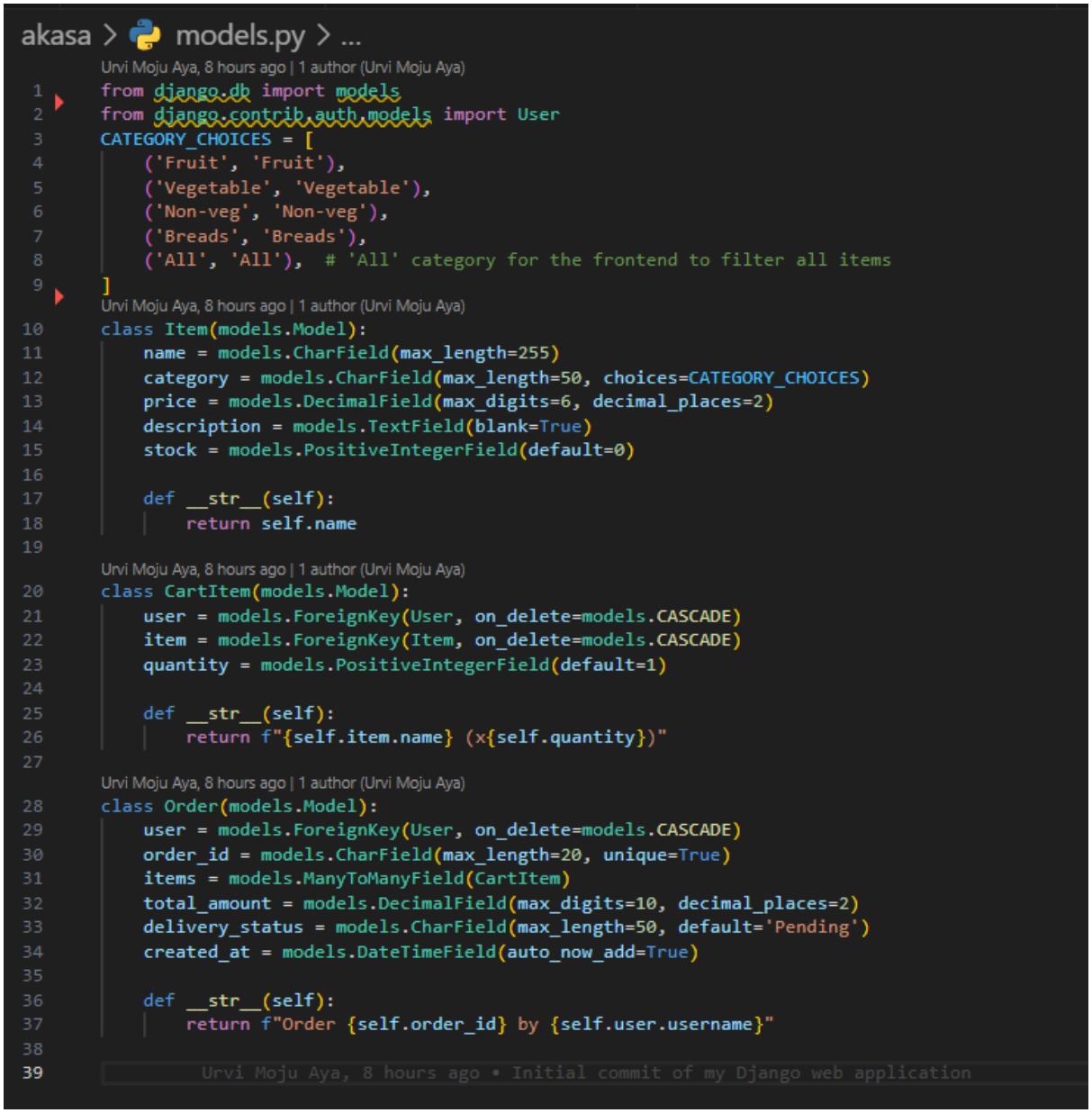
Checkout Page



Order History Page



Database:



```
akasa > models.py > ...
Urvi Moju Aya, 8 hours ago | 1 author (Urvi Moju Aya)
from django.db import models
from django.contrib.auth.models import User
CATEGORY_CHOICES = [
    ('Fruit', 'Fruit'),
    ('Vegetable', 'Vegetable'),
    ('Non-veg', 'Non-veg'),
    ('Breads', 'Breads'),
    ('All', 'All'),  # 'All' category for the frontend to filter all items
]
Urvi Moju Aya, 8 hours ago | 1 author (Urvi Moju Aya)
class Item(models.Model):
    name = models.CharField(max_length=255)
    category = models.CharField(max_length=50, choices=CATEGORY_CHOICES)
    price = models.DecimalField(max_digits=6, decimal_places=2)
    description = models.TextField(blank=True)
    stock = models.PositiveIntegerField(default=0)

    def __str__(self):
        return self.name

Urvi Moju Aya, 8 hours ago | 1 author (Urvi Moju Aya)
class CartItem(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    item = models.ForeignKey(Item, on_delete=models.CASCADE)
    quantity = models.PositiveIntegerField(default=1)

    def __str__(self):
        return f"{self.item.name} (x{self.quantity})"

Urvi Moju Aya, 8 hours ago | 1 author (Urvi Moju Aya)
class Order(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    order_id = models.CharField(max_length=20, unique=True)
    items = models.ManyToManyField(CartItem)
    total_amount = models.DecimalField(max_digits=10, decimal_places=2)
    delivery_status = models.CharField(max_length=50, default='Pending')
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"Order {self.order_id} by {self.user.username}"

        Urvi Moju Aya, 8 hours ago • Initial commit of my Django web application
```

Architecture

The application follows a **Model-View-Template (MVT)** architecture, typical in Django applications. The main components include:

- **Models**: Define the data structure (e.g., User, Item, Order).
- **Views**: Handle the business logic and interactions between models and templates.
- **Templates**: Render the user interface.

User Interface

- **Responsive Design**: Utilizes Bootstrap for a responsive layout, ensuring accessibility on both desktop and mobile devices.
- **Navigation**: A navbar provides easy access to main features like browsing items, viewing the cart, and order history.

Features

- **User Registration and Authentication**: Users can register, log in, and log out.
- **Item Browsing**: Users can browse items by category and view detailed descriptions.
- **Shopping Cart**: Users can add items to their cart, view their cart contents, and proceed to checkout.
- **Order History**: Users can view past orders.

Implementation

Technologies Used

- **Backend: Implemented using Django, following the Model-View-Template (MVT) architecture.**
- **Database: SQLite is used as the development database (can be replaced with PostgreSQL/MySQL for production).**
- **Frontend: Uses HTML5, CSS (inline), and Bootstrap for responsive design.**
- **Cart Management: The cart is session-based and is persistent across logins.**
- **Inventory Management: Item stock is automatically updated after a successful checkout.**

File Structure

```
food_app/
|
├── akasa/              # Django app
|   ├── migrations/     # Database migrations
|   ├── templates/      # HTML templates
|   ├── __init__.py
|   ├── admin.py
|   ├── apps.py
|   ├── models.py       # Data models
|   ├── tests.py
|   ├── urls.py         # URL routing
|   ├── views.py        # Business logic
|   └── forms.py        # Forms for user input
|
├── food_app/           # Project settings
|   ├── __init__.py
|   ├── settings.py     # Configuration settings
|   ├── urls.py         # Project URL routing
|   └── wsgi.py
|
└── manage.py           # Command-line utility for managing the app
```

Key Implementation Details

- **Models**: Define the structure of users, items, and orders.
- **Views**: Implement functions to handle requests and return appropriate responses.
- **Templates**: Utilize Django's templating language to render dynamic content.

How to Run the Application

Prerequisites

- Python 3.x
- Django installed (run pip install django)

Steps

1. **Clone the Repository**:

git clone https://github.com/mokshiaya/Akasa_food_app

cd food_app

2. **Install Dependencies**: If you have a requirements.txt, install dependencies using:

pip install -r requirements.txt

3. **Migrate the Database**: Initialize the database:

python manage.py migrate

4. **Create a Superuser** (optional for admin access):

python manage.py createsuperuser

5. **Run the Development Server**: Start the server to view the application:

python manage.py runserver

6. **Access the Application**: Open a web browser and go to http://127.0.0.1:8000.

Additional Features and Improvements

Potential Improvements:

- **Payment Integration**: Implementing a payment gateway (e.g., Stripe or PayPal) to allow users to make online payments.
- **Search Functionality**: Enhancing the item browsing experience with a search bar to filter items by name or category.
- **User Reviews and Ratings**: Allow users to leave reviews and ratings for items.
- **Wishlist Feature**: Enabling users to save items they are interested in for later purchase.
- **Enhanced UI/UX**: Further improving the user interface with more animations and better layout designs for a smoother user experience.
- **Admin Dashboard**: Creating a dedicated dashboard for administrators to manage items, view orders, and analyze sales data.