Author

A T Aswini
21f1001442
21f1001442@ds.study.iitm.ac.in
Computer Science Engineering Student

Description

The Flask-based multi-user e-commerce app facilitates grocery shopping with efficient product management. Users can purchase products from various sections, while the store manager can add sections and products with unique IDs, names, and details such as manufacture and prices. It ensures a seamless shopping experience for customers across different categories.

Technologies used

Flask: A micro web framework used to build the backend of the application.
Flask-RESTful: Extension for creating RESTful APIs to manage product and section data.
Flask-SQLAlchemy: Object-Relational Mapping (ORM) tool for interacting with the database and allows easy integration with the Flask application.
Flask-Login: Extension for user authentication and session management
SQLite: Relational database used for storing product, section, and order data
matplotlib : Library for creating charts and graphs in the application.
HTML , Jinja2 templates, Bootstrap: Frontend technologies for user interface design

DB Schema Design

1. Order Model:
   - Columns: id (Primary Key), user_id (Foreign Key), order_date (DateTime), total_amount (Float)
2. Section Model:
   - Columns: id (Primary Key), name (String)
3. Product Model:
   - Columns: id (Primary Key), name (String), manufacture_date (String), price (Float), quantity (Integer), image_url (String), section_id (Foreign Key)
4. User Model:
   - Columns: id (Primary Key), username (String - Unique), password (String), role (String)
5. Cart Model:
   - Columns: id (Primary Key), user_id (Foreign Key)
6. CartItem Model:
   - Columns: id (Primary Key), cart_id (Foreign Key), product_id (Foreign Key), quantity (Integer)

- Foreign key constraints ensure data integrity by linking related tables.
- Relationships between models facilitate easy querying and accessing related data.
- Utilizes Flask-Login UserMixin for user authentication.
- Supports adding products to carts and placing orders with one-to-one and one-to-many relationships.

API Design

The API was designed using Flask-RESTful. The following endpoints were created:
- /api/sections: GET and POST methods for retrieving all sections and adding a new section.
- /api/sections/<int:section_id>: GET, PUT, and DELETE methods to access, update, and delete a specific section by ID.
- /api/products: GET and POST methods for retrieving all products and adding a new product.
- /api/products/<int:product_id>: GET, PUT, and DELETE methods for accessing, updating, and deleting a specific product by ID.

Architecture and Features

The project follows the MVC (Model-View-Controller) pattern The controllers are defined in the api.py(API endpoints) and routes.py(views) files. The models.py file contains the database models using SQLAlchemy. The templates folder contains the HTML templates for rendering the frontend. The static folder stores static assets like images.

Features implemented include:
- CRUD operations for sections and products through the API.
- Multi-user authentication: Admin and user login and sign up
- Inventory and Product Management: Admin can add, edit, and delete products and sections
- Order Processing: Search for sections and buy products from one or multiple sections

Additional:
- Summary Generation: A summary page displays statistics using pie charts and bar charts of products and orders.

Video

https://drive.google.com/file/d/1tLJYG6vYCNv_mYgZYcgIWO-v0TX1NG3p/view?usp=sharing