Author

MOHAMMED HASSAN

21F1001017

21f1001017@ds.study.iitm.ac.in

I am currently in the diploma level of IITM BS Degree in Data Science and Applications.

Description

The aim of this project is to develop a Grocery Store Application. The Application needs to be developed in such a way where users can buy the products and sell their products by registering as store managers. The Admin is the overall administrator of the application who manages the Application.

Technologies Used:

1. Flask for Backend Application code .
2. SQLALCHEMY - For creating models
3. FLASK-SECURITY - For Authentication
4. Vuejs and JavaScript - For Frontend Application code.
5. Redis and Celery - For Backend Jobs

DB Schema Design

1) The User Model:

Id column which is a primary key and acts as user id. Integer type and gets auto increment.
Username which is the name of the user and every username should be unique.
Email column which is a string and should be unique.
Is_active is a boolean field. For normal users it is set to True . Whereas for Store Managers it is set to False. Once the admin approves the Store Manager it becomes true.
Fs_uniquifier is used for flask security authentication purposes
Roles relationship captures the roles of a user . Ideally there are three roles - Admin, Store Manager and the User.
Categories relationship gives the list of categories created by the user who is the store manager,
Products relationship gives the list of products created by the user who is the store manager.
Orders relationship gives the list of orders placed by the user who is store manager.

2) The Category Model:

   Every category object has a unique id and a non-nullable value of category name .
   The boolean field is_approved is set to the default value of 0 . If the admin approves the category created by the store manager , then it is set to 1,
   Products relationship gives the list of products under a category,

3) The Product Model:

   Every product has an id , name , unit , price per unit . These fields are self-explanatory.
   Creator_id is the id of the store manager who is the creator of this product.
   Category_id is the id of the category under which this product falls.
   Sold specifies the quantity of this product sold out . It is set to 0 as default.

4) Category Edit and Category Delete Models.
   These two models are required to do the crud operations on category objects .
   These models acts as an intermediary for the request and approval of Updating and Deleting the category objects where the store manager requests and the admin approves the request.

5) The Cart Model:

   This model is used to add the products in the cart when the user is trying to buy products from the store . It has cart_id , product_id , product_name, price, quantity selected to buy and the total amount for this cart item . It has a boolean field value is_buyed set to 0 as default .

6) The Orders Model and the OrderItem Model:

   Each element in the orders model has a unique order id and for each order object we have a list of Order items associated with that order id . The items relationship in Order model gives the list of Ordered items under this order. It captures one to many relationship.

Architectures and Frameworks:

The application is divided into two components. The application folder contains all the python files for creating the backend which is written in flask . It has models.py for designing the database models , resource.py which is the api file for doing operations on categories , mail_service.py has the application code to send emails. Views.py containing all the routes needed for the operations on backend triggered from the frontend . The security.py file has the necessary flask - security objects for the user Authentication. Tasks.py contains the python code for triggering asynchronous tasks that run in the backend. Worker.py has the function for initializing the celery and combining it with a flask task . Static has all the components required to render the frontend . All these files are written in javascript and vue cdn for combining vue . main.py and other configuration files are kept in parallel with these components along with upload initial_data.py

Video:
https://drive.google.com/file/d/1RE-8XRPr0_mhnu8gwJ3lEMDUipWrxDjT/view?usp=sharing