COURSE END PROJECT

Shopping App in Python

Murali Krishnan Solayappan
Muralikrishnan.s@gmail.com

Table of Contents

Overview 2

Requirements 2

Shopping App - Core Requirements 2

Software version 3

Code repository 3

folder structure 3

Database schema 3

ER Diagram 5

Tables as in the database 6

User Interface 7

Key Features 8

SQlite3 database 8

Mock data 8

Text based User Interface (TUI) 8

Role based Access Control (RBAC) 8

Data persistence 8

Database initialization 9

Predefined Users 9

Execution 9

Conclusion 10

Overview

This is the project developed for Course end assessment of the below course under **Caltech Post Graduate Program in AI and Machine Learning.**

PGC AIML Foundations: Programming Refresher

The requirement is to develop a shopping cart app (backend) using Python that provides few user (shopping user) level features and few administrator level features.

The full problem statement is available in word document format here

This app has been developed in python programming language and *sqlite3* as database store.

Requirements

After reviewing the problem statement, the following software requirements have been derived to ease out the development process, so all features are implemented.

Shopping App - Core Requirements

Req-Id	Requirement description	Status
1	A welcome message should initially display	**Completed**
2	demo database to be created for user and admin login	**Completed**
3	construct sample product catalog	**Completed**
4	product id, category id, price should present in database	**Completed**
5	Both administrator and users can view the catalog	**Completed**
6	user could view cart items	**Completed**
7	user could add items to cart	**Completed**
8	user could remove items from cart	**Completed**
9	demo payment checkout options	**Completed**
10	gateway redirect message after checkout	**Completed**
11	admin has exclusive login	**Completed**
12	error to be displayed on all invalid actions	**Completed**
13	admin could add new products to catalog	**Completed**
14	admin could modify existing products in catalog	**Completed**
15	admin could remove product from catalog	**Completed**
16	admin could add new catalog/category	**Completed**
17	admin could remove existing catalog/category	**Completed**
18	user should be prevented from all admin actions	**Completed**

Software version

The following are the versions of different software used in this application.

Software	Version
Python Interpreter	3.12.0
Sqlite3 database	3.49.1 (64 bit)
Dbeaver	25.0.1 (Community edition)

Code repository

The code including the sql files, documents, etc., of this application is available in the following GitHub repository.

https://github.com/msolayap/shopping_app

folder structure

For simplicity, the project is not so modularized. Each *class* should be developed in separate files, but instead all the code with classes, configurations, methods are placed in one single file called “app.py.” The code is organized in the following manner.

Folder	Purpose and contents
shopping_app/src/	Main folder for application source. app.py is the primary application file.
shopping_app/src/db/	Folder holding all SQL initialization file and actual database file (.db).
shopping_app/docs/	Folder containing all project documents like problem statement, ER diagram, Application screenshots, this writeup document, etc.,

Database schema

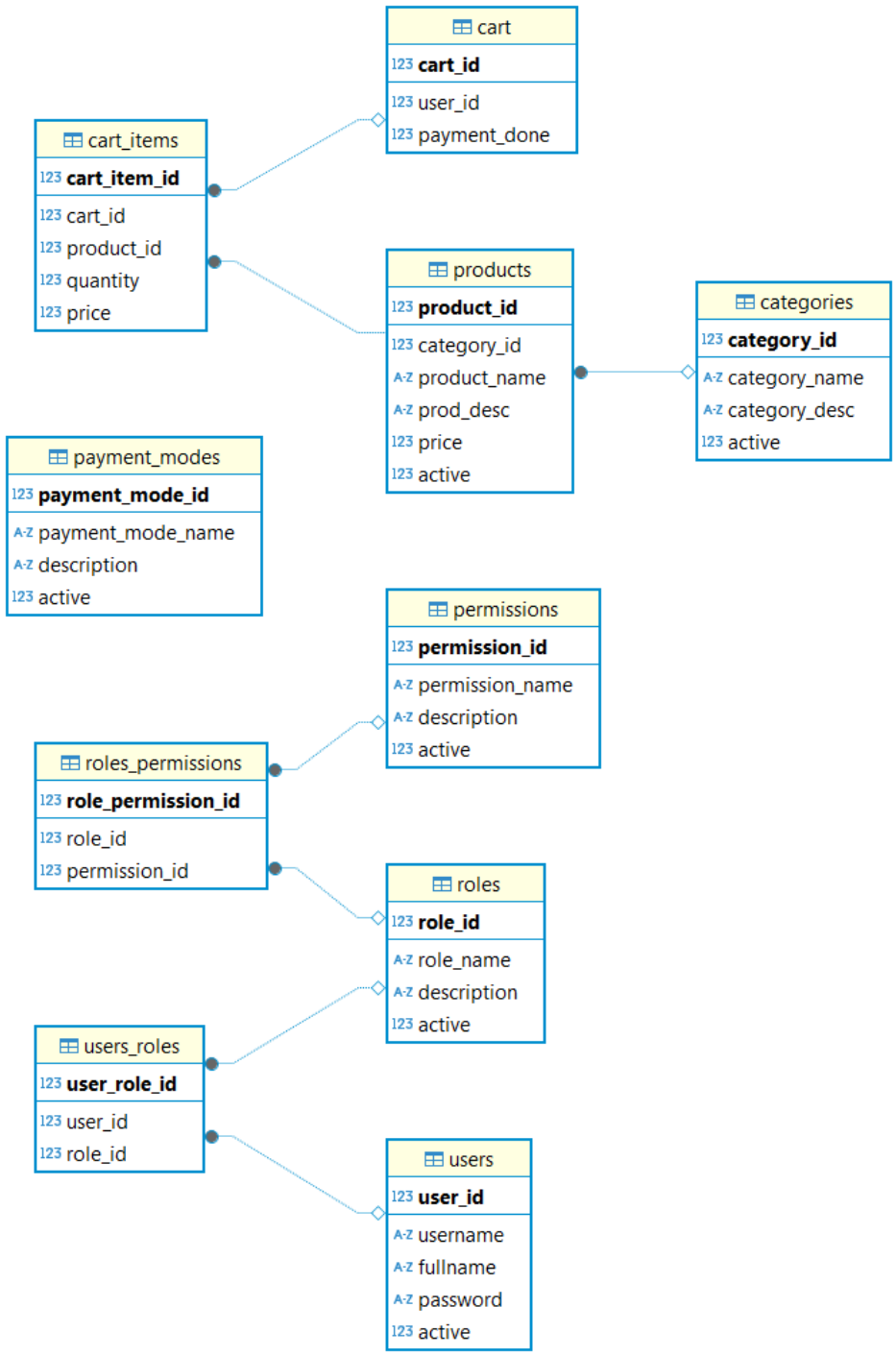
Based on the requirements, table schemas were created for various features like products, catalog, users, user roles, user permissions, cart management, etc.,

Following are the list of tables created in the database.

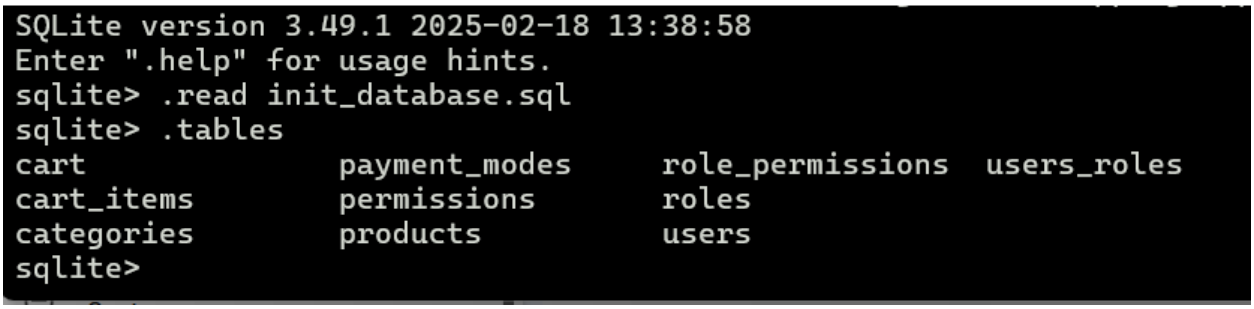
Table Name	Purpose
categories	Table to hold list of product catalogs
products	Table to hold list of available products under each catalog. Foreign Key reference categories for category type
roles	Table to hold list of roles in the application
permissions	Table to hold list of all permissions available in the application
role_permissions	Table to hold map of permissions applicable for each role in the application Foreign key reference roles and permissions tables
users	Table to hold list of users in the application
users_roles	Table to hold map of users to permitted roles in the application Foreign key reference users and roles tables.
cart	Table to hold all carts created by users
cart_items	Table to hold all items within each cart created by users during application usage. Foreign key reference cart table for cart id of each item
payment_modes	Table to hold list of available payment methods in the application.

ER Diagram

The following image shows the entity relationships between the tables present in the database.



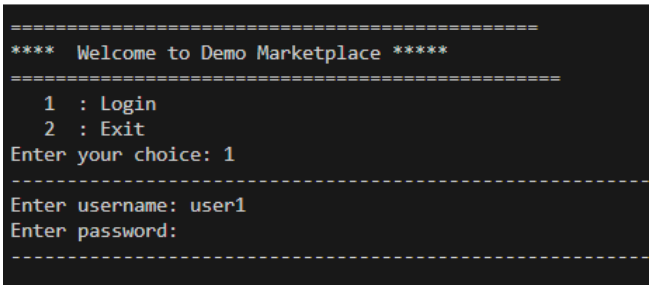
Tables as in the database



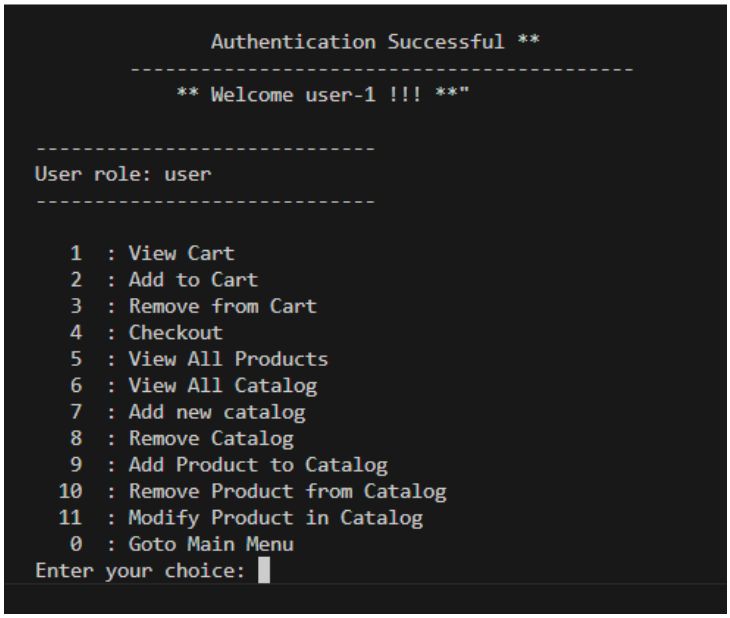
User Interface

Though the problem statement does not mandate User Interface as requirement, anyway its developed to test and show case the features of the application. Text based menu interface and prompt-based user input have been implemented for simplicity. For e.g., see the images below.

Login screen



Main Menu showing various user options.



Key Features

Apart from completing all the core requirements mentioned in the problem statement, the app has some features to note, such as

SQlite3 database

Sqlite3 based database implementation (instead of mock data in process instance).

Mock data

Mock data can be filled to the relevant Database Tables, using SQL DDL statements.

Text based User Interface (TUI)

User Interface Menus were designed close to a professional application with proper alignment of elements using Python format strings.

Role based Access Control (RBAC)

Role Based Access Control has been implemented using a full normalization model. The following four tables play a vital role in that.

- users
- roles
- users_roles
- permissions
- roles_permission

Users were mapped to their roles in *the users_roles* table, correspondingly roles and applicable permissions were mapped in *roles_permissions* table.

In this way, not just users or admin, but any number of roles can be created and assigned to users with specific permissions. For e.g., a category_admin to manage product catalog or product_admin to add, remove products alone can be created easily.

This provides fine-grained access control in the application.

Data persistence

Any modification to the product catalog will be persisted as the actual data were written to database tables in real time.

Database initialization

Before running the app, the database must be initialized with Tables and Mock data in the tables. The DDL SQL statements and mock data are available in the SQL file *src/db/init_database.sql*

The file can be executed by the following steps.

Launch sqlite CLI interface. **Mickoo** is the application name, so the database is created in the file *mickoo.db*

```
shopping_app>sqlite3 mickoo.db
```

Execute the init SQL file using the ".read" command in SQLITE cli interface.

```
SQLite version 3.49.1 2025-02-18 13:38:58
Enter ".help" for usage hints.
sqlite> .read init_database.sql
sqlite>
```

Predefined Users

user name	password	role
user1	abc123	User (Normal shopping user)
user2	abc123	User (Normal shopping user)
admin1	abc123	Administrator
admin2	abc123	Administrator

Execution

The app can be started in command line interface. Use "bash" in Linux or "command" shell in Windows operating system.

```
cd .\shopping_app\src
shopping_app\src> python .\app.py
```

```
================================================
****  Welcome to Demo Marketplace *****
================================================
   1  : Login
   2  : Exit
Enter your choice:
```

Please refer to the screenshot document for full usage of the application.

Conclusion

This project is a simple implementation of the problem statements, still there is ample room for improvement. For example, the following features could be implemented.

- Security
- Modularization
- Scalability
- Asynchronization
- Cache implementation
- Data security
- REST API based backend
- Real Payment interface.