**A Details of the design, architecture, microservices, database scripts, code, unit test cases for this project.**

**Introduction**

Welcome to an e-commerce platform built using Flask!

Our website offers a wide range of products and services, providing customers with a seamless shopping experience.

Using Flask, we have developed a fast, reliable, and secure platform that ensures smooth transactions and a user-friendly interface. Our goal is to provide you with a convenient and enjoyable shopping experience, right from the comfort of your home.

Explore our website to discover amazing deals, exciting offers, and a diverse collection of products. Join us in revolutionizing the online shopping experience with Flask!

**Scope**

The e-commerce website project aims to create a robust online platform for selling a variety of products to customers worldwide. The scope includes developing a user-friendly interface for browsing products, adding items to cart, and completing purchases. Additionally, the project includes features for user authentication, order management, and inventory tracking.

**Target Audience**

The target audience for the e-commerce website includes individuals of all ages and backgrounds who are looking to purchase products online. The website caters to a wide range of interests and preferences, offering a diverse selection of products to meet the needs of different customer segments.

**Technologies Used**

The e-commerce website is built using Flask, a lightweight and efficient web framework for Python. Flask provides a flexible and scalable architecture for developing web applications, making it ideal for this project. The website also utilizes SQLite for database management, providing a reliable and efficient storage solution. Additionally, the website incorporates HTML, CSS, and JavaScript for front-end development, ensuring a responsive and interactive user experience.

**Design and Architecture**

**Overall Architecture**

The e-commerce website follows a client-server architecture, where the Flask application serves as the server-side logic, handling client requests and interacting with the database. The front-end is built using HTML, CSS, and JavaScript, providing a responsive and interactive user interface.

**Design Patterns**

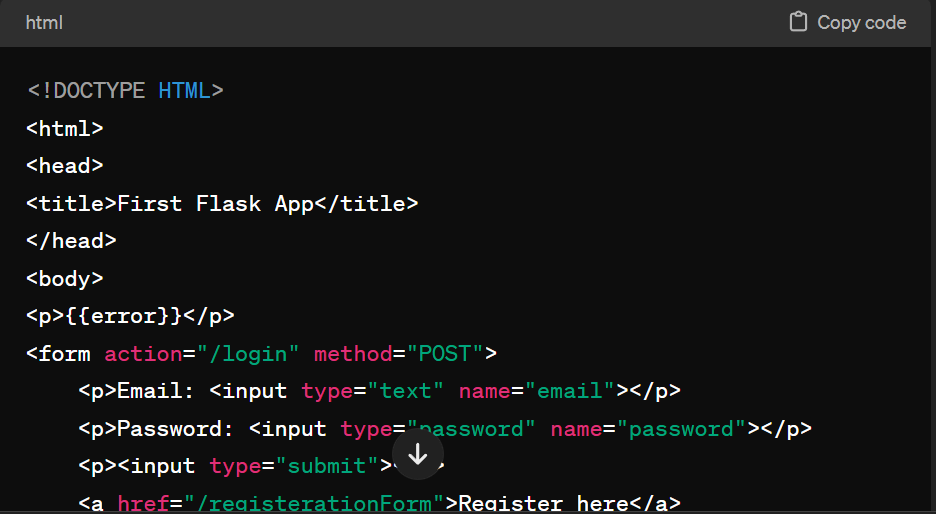
mvc the application follows the MVC design pattern, separating the presentation layer (View) from the business logic (Controller) and data (Model). This allows for easier maintenance and scalability.

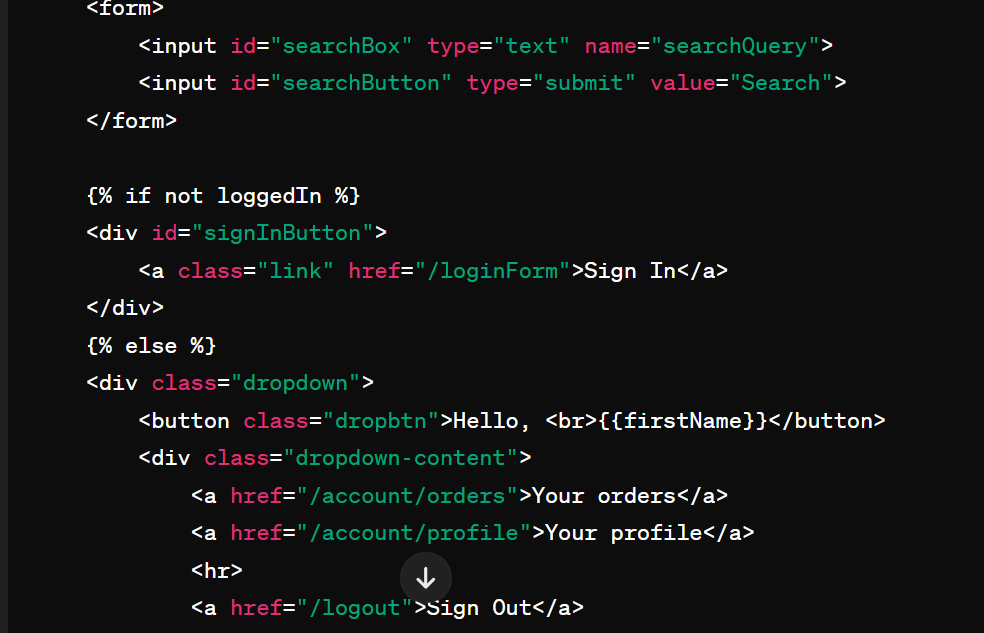
**Microservices Architecture Overview**

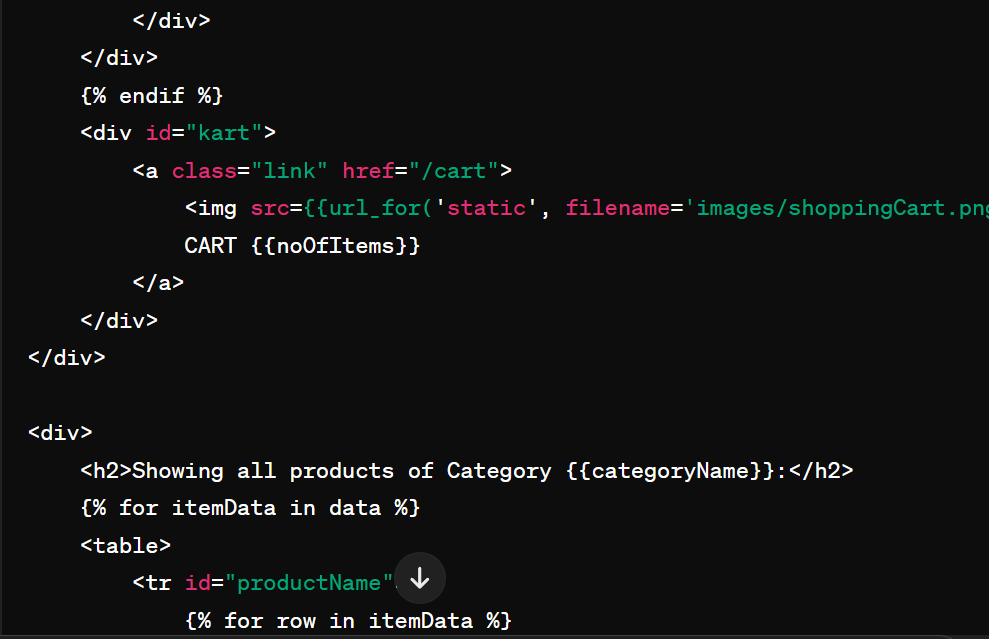
**Service Boundaries**

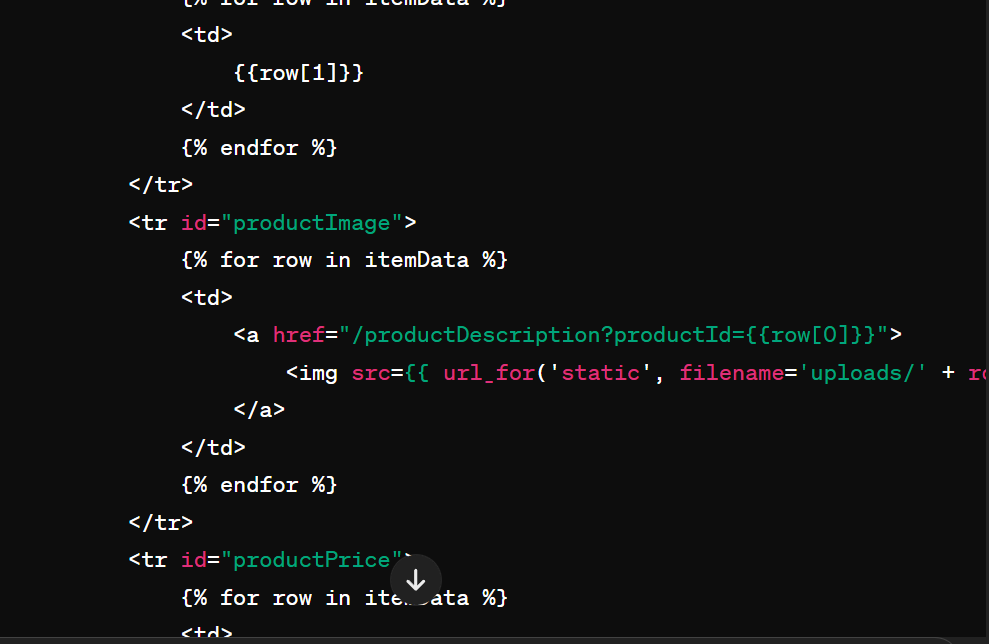
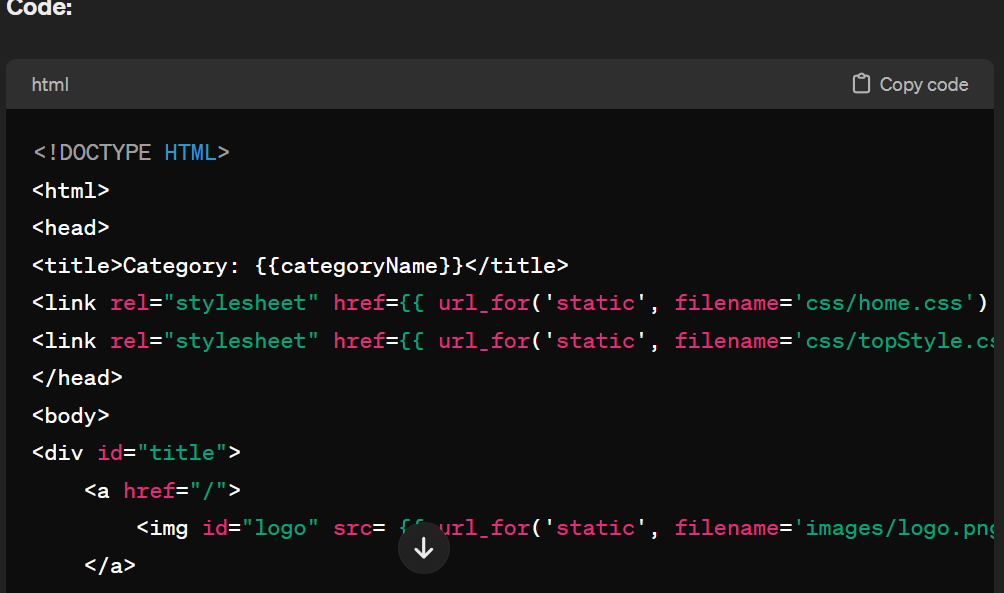
* **User Service: Manages user authentication, registration, and profile management.**
* **Product Service: Handles product-related functionalities such as adding, removing, and updating products.**
* **Order Service: Manages orders, including order creation, updating order status, and order history.**
* **Cart Service: Deals with operations related to the shopping cart, such as adding, removing, and updating items in the cart.**
* **Category Service: Manages product categories and provides functionalities to list products by category.**

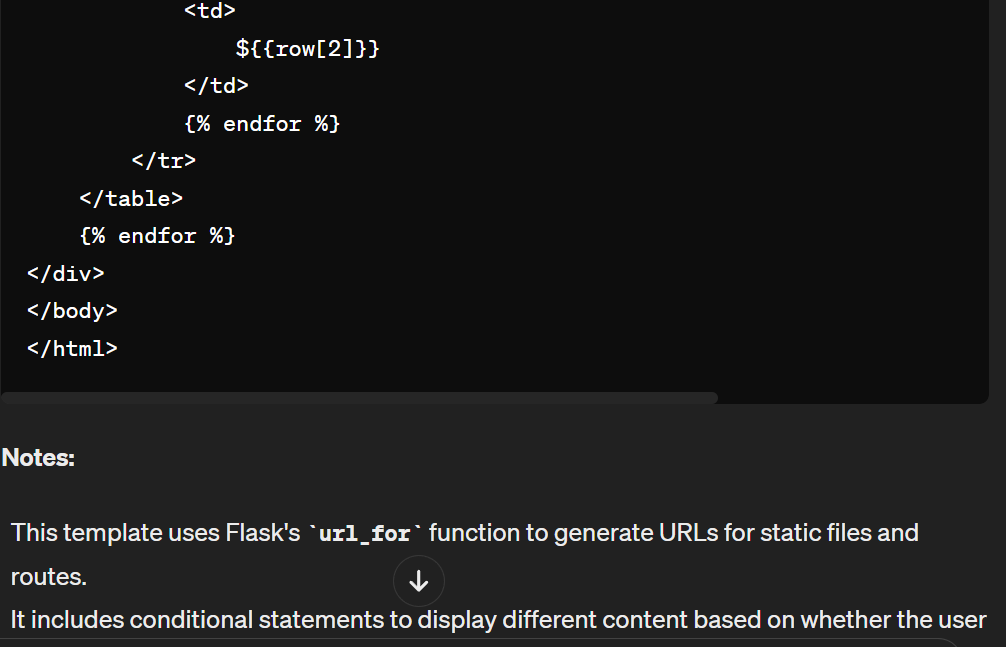
**Code snippet- This is login HTML page code snippet**



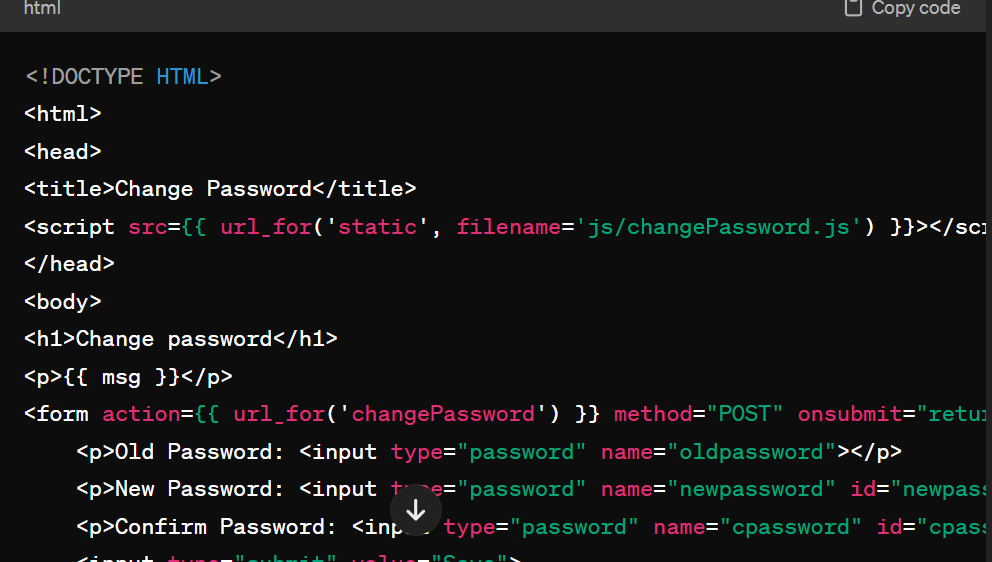
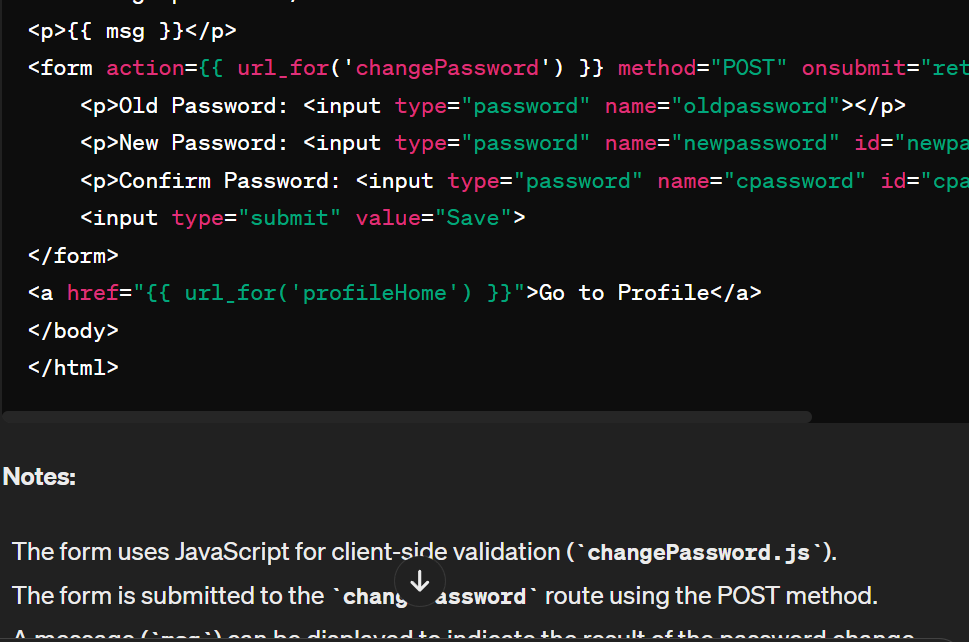
**create display page snippet**



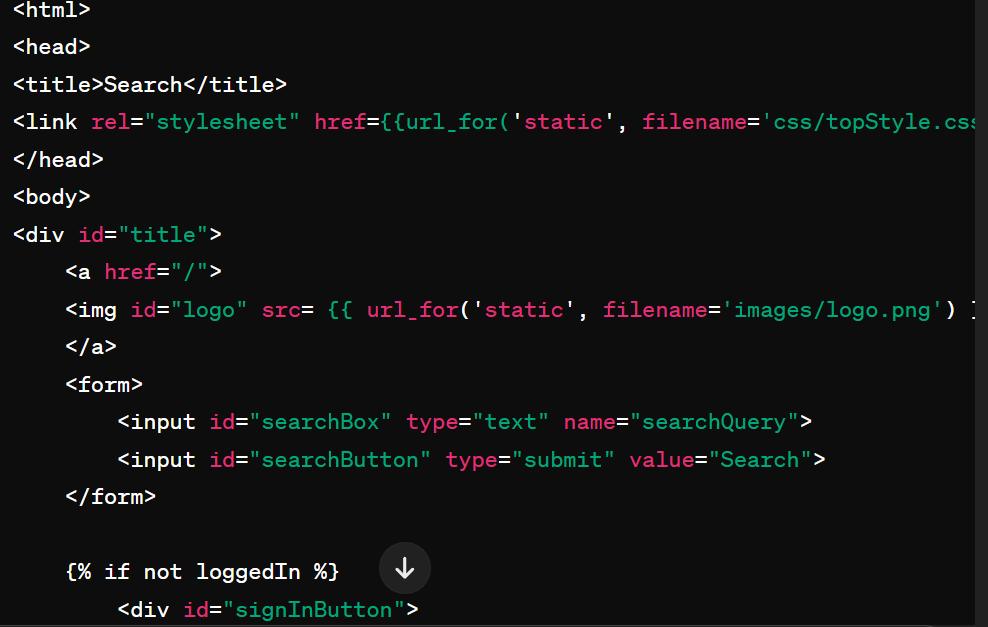


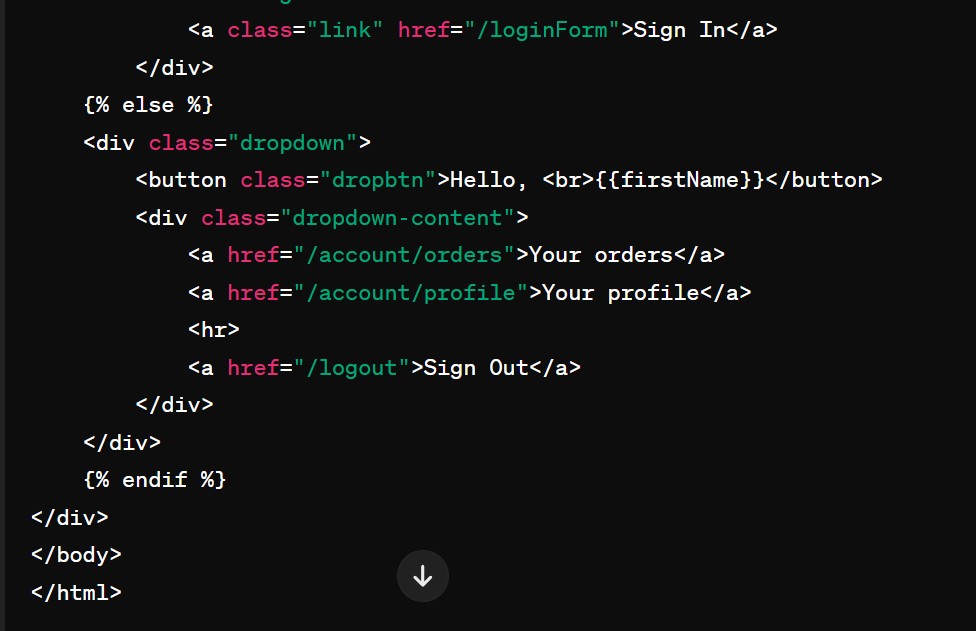


**Change password page snippet**

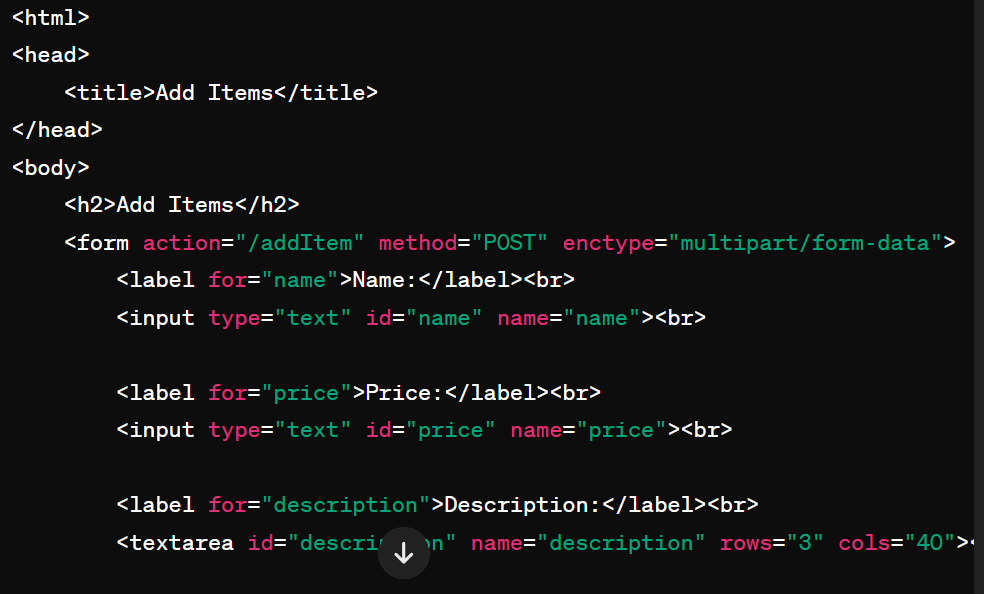


**Card HTML page-**





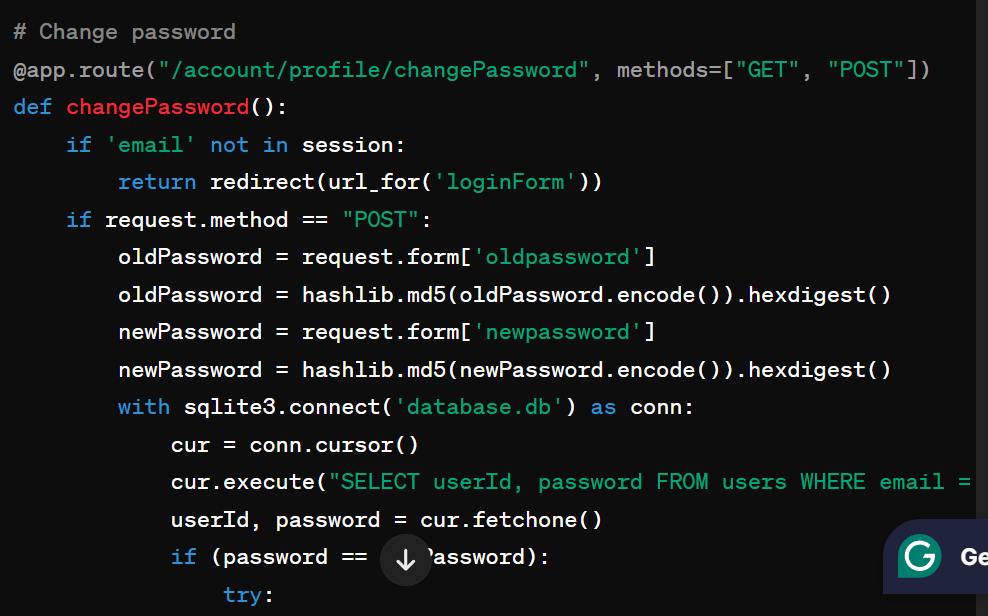
**Add HTML**

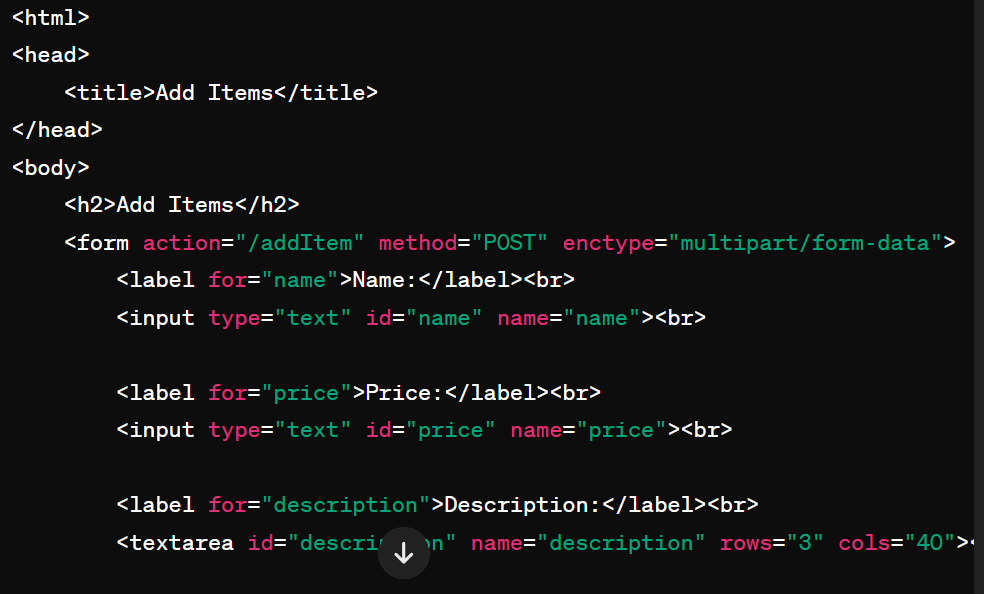


**Update profile**

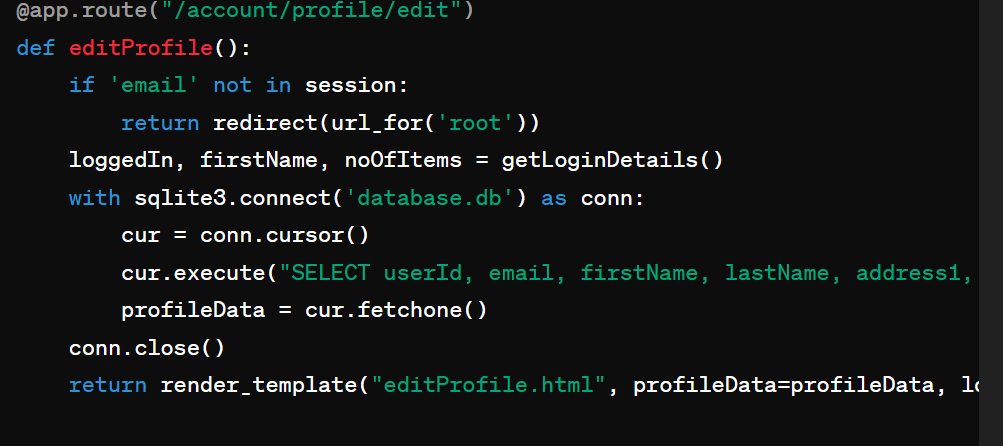


**Change password**

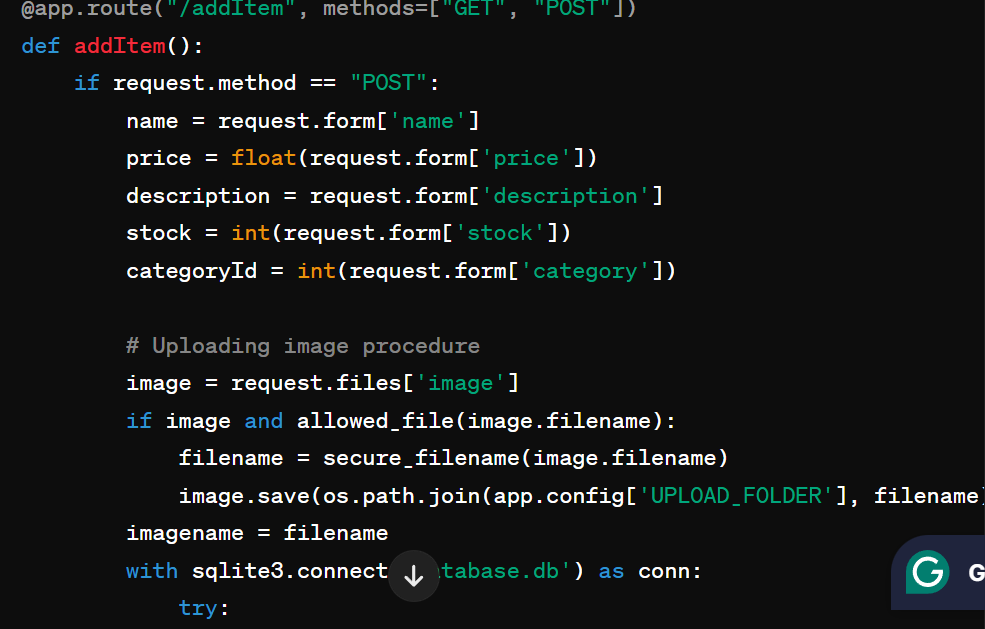


**Add HTML page**

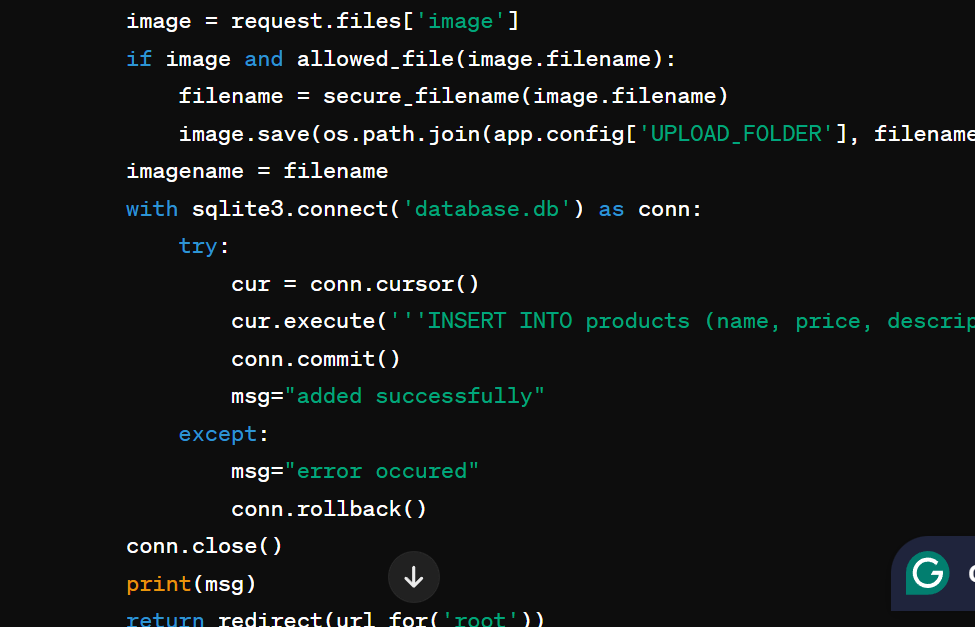
**Edit profile**



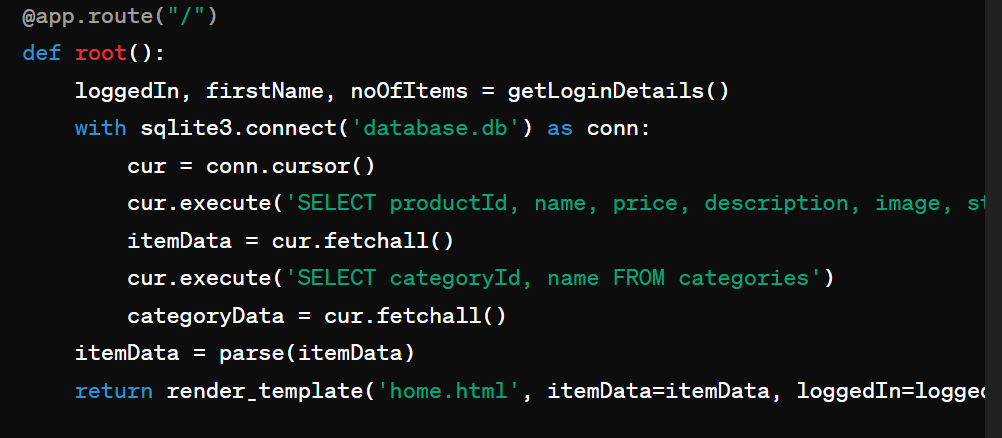
**Add item**



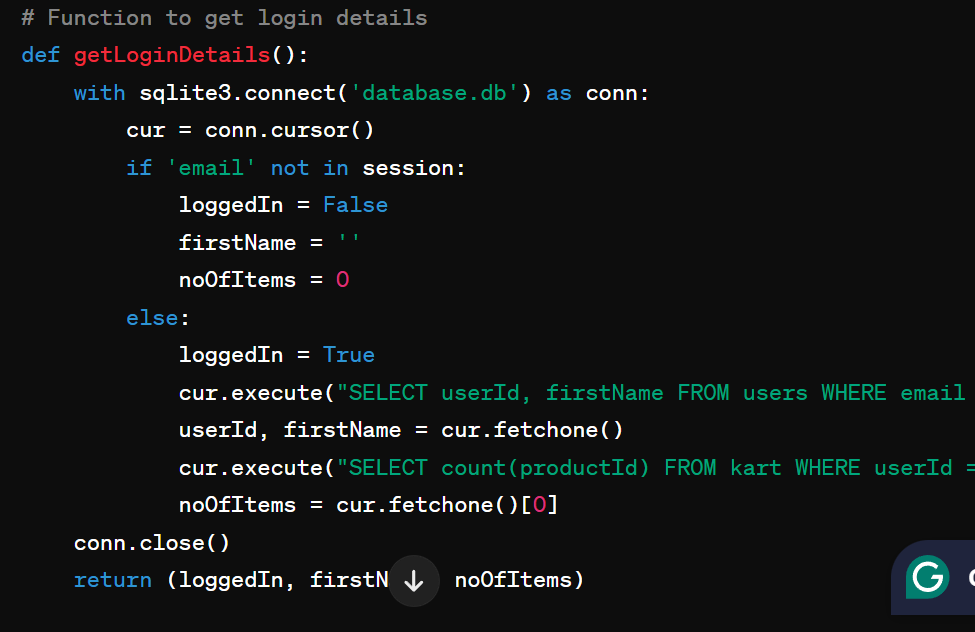
**Upload image procedure**



**Home page**



**Get login details function**



**Unit test cases code :-**

import unittest

from app import app, db

from models import User, Product

class TestApp(unittest.TestCase):

def setUp(self):

app.config['TESTING'] = True

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///test.db'

self.app = app.test\_client()

db.create\_all()

def tearDown(self):

db.session.remove()

db.drop\_all()

def test\_index\_route(self):

response = self.app.get('/')

self.assertEqual(response.status\_code, 200)

def test\_product\_detail\_route(self):

product = Product(name='Test Product', price=10.0, stock=5)

db.session.add(product)

db.session.commit()

response = self.app.get(f'/product/{product.id}')

self.assertEqual(response.status\_code, 200)

self.assertIn(b'Test Product', response.data)

def test\_user\_registration(self):

response = self.app.post('/register', data=dict(

email='test@example.com',

password='password',

confirm\_password='password',

first\_name='Test',

last\_name='User'

), follow\_redirects=True)

self.assertIn(b'Registered Successfully', response.data)

def test\_login(self):

user = User(email='test@example.com', password='password')

db.session.add(user)

db.session.commit()

response = self.app.post('/login', data=dict(

email='test@example.com',

password='password'

), follow\_redirects=True)

self.assertIn(b'Logged in successfully', response.data)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()