To integrate the front-end of your e-commerce website, which is completed using HTML, CSS, and Bootstrap, with a Python backend and host it locally using XAMPP, here's a structured approach:

### 1. \*Project Structure\*:

Create the following structure for your project:

e-commerce-website/

├── templates/

│ ├── index.html

│ ├── product.html

│ ├── cart.html

├── static/

│ ├── css/

│ │ └── styles.css

│ ├── js/

│ │ └── script.js

├── app.py

├── database.db

└── README.md

### 2. \*Flask Backend Setup\*:

Flask will serve as your backend to interact with the front-end.

\*Install Flask:\*

bash

pip install flask

\*\*Create app.py file:\*\*

This file will handle your backend logic, like managing products, displaying them, and handling the cart.

python

from flask import Flask, render\_template, request, redirect, url\_for

import sqlite3

app = Flask(\_\_name\_\_)

# Database setup function

def init\_db():

conn = sqlite3.connect('database.db')

cursor = conn.cursor()

cursor.execute('''CREATE TABLE IF NOT EXISTS products (

id INTEGER PRIMARY KEY AUTOINCREMENT,

name TEXT NOT NULL,

price REAL NOT NULL,

description TEXT

)''')

conn.commit()

conn.close()

# Home route displaying products

@app.route('/')

def index():

conn = sqlite3.connect('database.db')

cursor = conn.cursor()

cursor.execute('SELECT \* FROM products')

products = cursor.fetchall()

conn.close()

return render\_template('index.html', products=products)

# Route for individual product pages

@app.route('/product/<int:product\_id>')

def product(product\_id):

conn = sqlite3.connect('database.db')

cursor = conn.cursor()

cursor.execute('SELECT \* FROM products WHERE id=?', (product\_id,))

product = cursor.fetchone()

conn.close()

return render\_template('product.html', product=product)

# Add to cart route

@app.route('/add\_to\_cart', methods=['POST'])

def add\_to\_cart():

product\_id = request.form.get('product\_id')

# Logic to add product to cart (could use session, or a database table)

return redirect(url\_for('cart'))

# Cart page route

@app.route('/cart')

def cart():

# Logic to display cart items

return render\_template('cart.html')

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

init\_db() # Initialize the database

app.run(debug=True)

### 3. \*Front-End Integration\*:

Your front-end is already completed with HTML, CSS, and Bootstrap. Below is an example template for the home page (index.html) that displays products from the database:

\*\*templates/index.html:\*\*

html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>E-Commerce Website</title>

<link rel="stylesheet" href="{{ url\_for('static', filename='css/styles.css') }}">

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">

</head>

<body>

<div class="container">

<h1 class="mt-4">Welcome to Our E-Commerce Store</h1>

<div class="row">

{% for product in products %}

<div class="col-md-4">

<div class="card mb-4">

<img src="https://via.placeholder.com/150" class="card-img-top" alt="{{ product[1] }}">

<div class="card-body">

<h5 class="card-title">{{ product[1] }}</h5>

<p class="card-text">{{ product[3] }}</p>

<p class="card-text"><strong>${{ product[2] }}</strong></p>

<a href="{{ url\_for('product', product\_id=product[0]) }}" class="btn btn-primary">View Product</a>

</div>

</div>

</div>

{% endfor %}

</div>

</div>

</body>

</html>

### 4. \*Running the Flask App Locally with XAMPP\*:

Although Flask can be run directly using its built-in server, to use XAMPP for hosting, follow these steps:

1. \*Start XAMPP\*: Open the XAMPP Control Panel and start the Apache server.

2. \*Run Flask Application\*:

- Open a terminal and navigate to your project directory.

- Run the Flask application:

bash

python app.py

3. \*Access the Website\*:

- Open your web browser and go to http://localhost:5000/ to see your e-commerce website running.

### 5. \*Database Initialization\*:

In app.py, the init\_db() function initializes the SQLite database, creating a products table. You should populate this table with your products, either through a script or manually.

### 6. \*Expand the Project\*:

This is a foundational setup. You can expand this project by adding features like:

- User authentication.

- Cart management (session-based or using a database).

- Payment gateway integration.

- Product categories and filters.

### 7. \*Deployment\*:

For production, consider using a more robust environment, such as deploying on a cloud service with a WSGI server like Gunicorn, and integrating with a real database like MySQL.

This approach sets up a basic e-commerce website that integrates your front-end with a Python backend and allows local hosting using XAMPP.