E-Commerce Price & Stock Tracker Across Multiple Stores

Anika P - PES1UG23CS073
Anish M - PES1UG23CS078

Description:

This project is a streamlined e-commerce management and storefront system built using Flask and MySQL. It enables customers to register, browse categorized products across multiple stores, track price changes, create wishlists, and set store-specific price alerts. The system maintains product price history and alerts users when desired prices are reached. Administrators can manage products, categories, stores, and inventory, as well as view key analytics and run automated restocking procedures. The application uses Jinja2 templates for dynamic pages and MySQL for data storage, providing a functional end-to-end e-commerce management workflow.

User requirement specification:

- **Purpose of the project:**
  This project implements a small e-commerce web application using Flask (Python) and MySQL. It allows customers to browse products, manage wishlists, set price alerts, and make purchases, while administrators can manage products, inventory, stores and view analytics. The application provides price tracking (price history, alerts), inventory management, and typical e-commerce features (product listing, detail pages, customer accounts).

- **Scope of the project:**
  Includes:
  1) Customer-facing features: account registration/login, product browsing, product detail pages, wishlist management, price alerts, view purchase history, basic expense tracking.
  2) Admin-facing features: manage products, categories, stores, inventory, users, and view analytics (counts, stock totals, alerts).
  3) Backend: MySQL database with tables for Customers, Products, Category, Store, Inventory, Orders, PriceAlerts, PriceHistory, Wishlists, and stored procedures for some operations (app expects procedures like AddNewProduct, ComparePrices, AutoRestock).
  Excludes (out of scope): Payment gateway integration, production-level security hardening (app uses a simple secret key and local DB settings), full CI/CD.

- **Detailed description:**
  This is a small e-commerce management & storefront application. Customers sign up and log in to browse products grouped into categories and offered by different stores. Each store can have its own price for a product (modelled in Inventory with price_in_store), and the application records price history so customers can see price changes. Customers can add products to wishlists and set price alerts for specific products at specific stores; if a product reaches a target price the alert can be triggered (the app queries inventory/price to determine trigger). Administrators can add products (the app calls stored procedures to add products), update inventory levels, view analytics (counts of customers/products/stores and inventory totals), and run an auto-restock procedure. The app uses server-side templates (Jinja2) for pages and mysql-connector-python for DB access**.**

- **Function Requirements:**
  1) User Registration & Login — Register new customers, store basic details and phone numbers; log in/out and maintain session.
  2) Customer Profile Management — View and edit customer details (name, email, phones).
  3) Product Browsing / Search — Browse product catalogue, filter by category, view paginated product lists.
  4) Product Detail — See product details including store availability and prices (using Inventory join).
  5) Wishlist Management — Create wishlists, add/remove products to wishlist (Wishlist, WishlistProduct).
  6) Price Alerts — Create price alerts per product/store with target price, view, and delete alerts (table: PriceAlert).
  7) Price History Viewing — Show historical price changes for a product (PriceHistory).
  8) Shopping / Orders (basic) — Record a purchase/order (Orders table used for purchase history and expense tracking).
  9) Admin Authentication & Dashboard — Admin login + admin dashboard with counts and analytics queries.
  10) Admin Product Management — Add/edit/delete products (app calls stored procedures such as AddNewProduct).
  11) Inventory Management — Manage inventory entries (quantity / price in store) and view total stock.
  12) Store Management — Add/update store records (store name, possibly location).
  13) Analytics & Reports — Dashboard queries count customers/products/stores, active alerts, inventory totals and aggregated category metrics.
  14) Stored Procedures Interface — App calls stored procedures: e.g., ComparePrices, AutoRestock, AddNewProduct.

15) Notifications / Alert Triggering — Mechanism to flag triggered alerts (app queries PriceAlert with status = 'triggered').
16) CRUD via web forms — All entity create/update/delete operations accessible via admin/customer UI.
17) Input validation & Flash messages — Basic form validation with feedback via Flask flash().

List of Software/Tools/Programming languages used:

Programming Languages

- Python 3.x (Backend logic and server-side development)
- HTML5, CSS3, JavaScript (Frontend templates and UI)

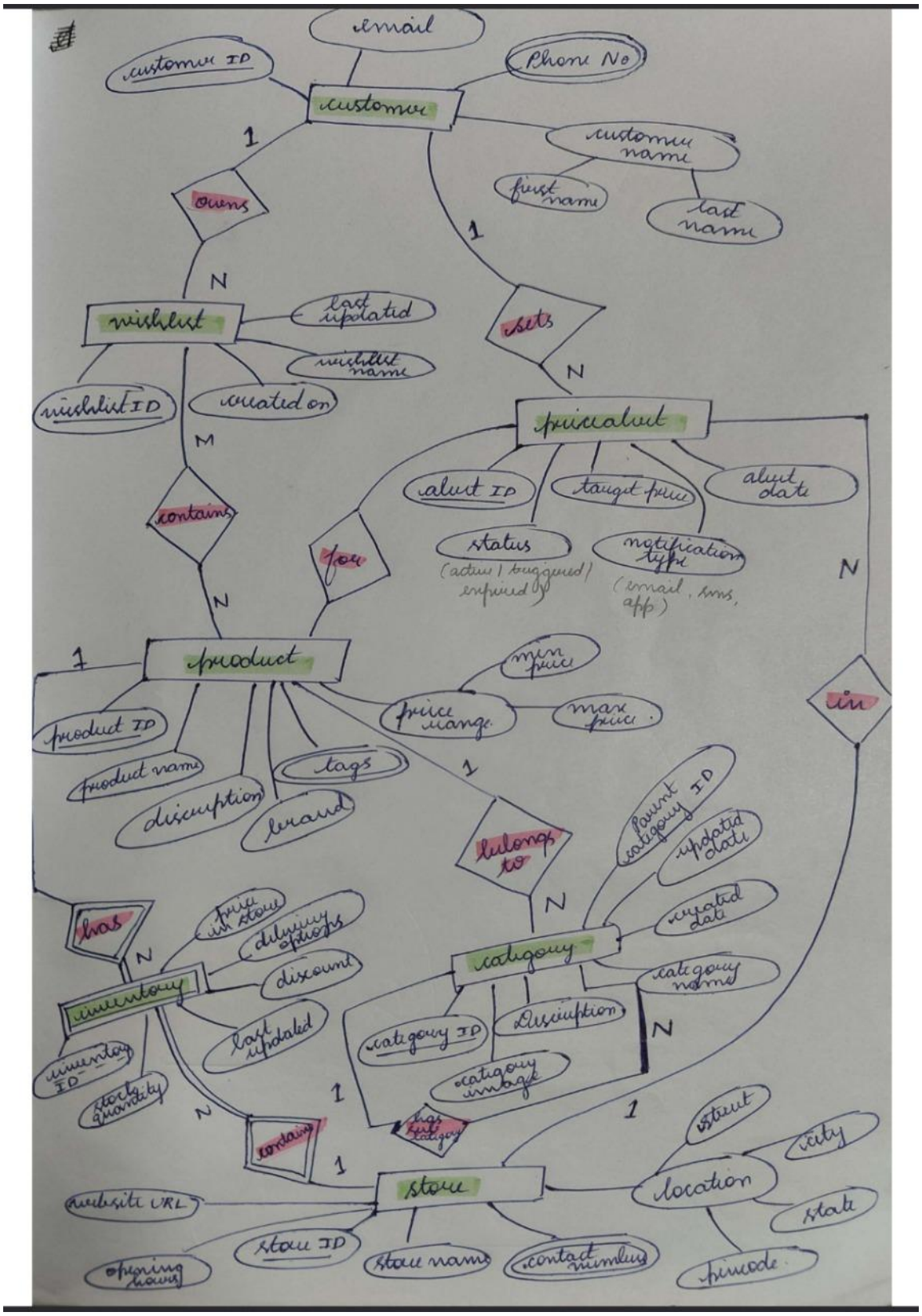
Frameworks & Libraries

- Flask (Web framework)
- Jinja2 (Templating engine)
- mysql-connector-python (Database connectivity)
- Bootstrap (Frontend styling, used in templates)

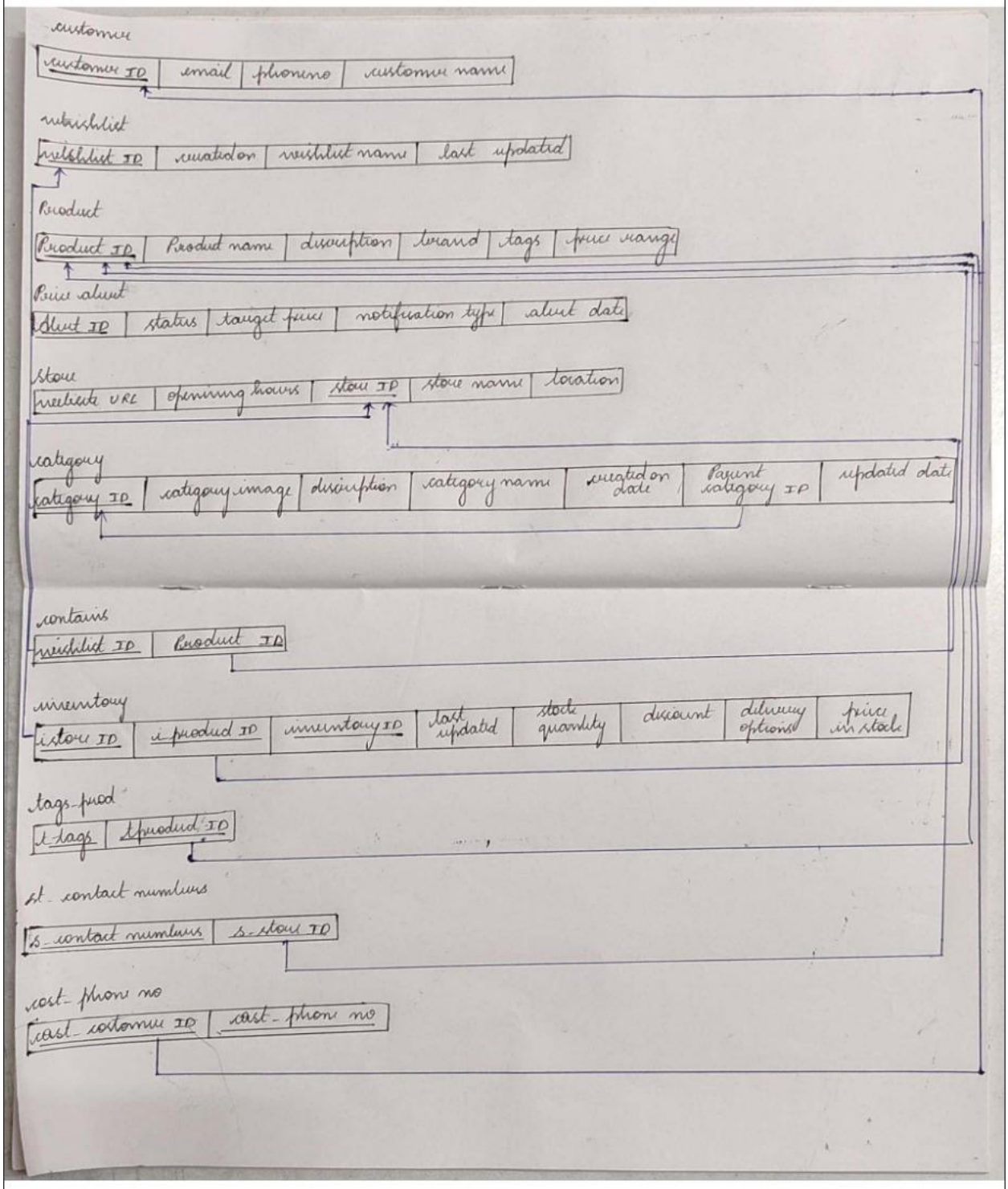
Database & Data Tools

- MySQL (Primary relational database)Development Tools
- Visual Studio Code (IDE for development)
- Git & GitHub (Version control)
- Command-line / Terminal (Running Flask server & dependencies)
- pip / virtualenv (Dependency and environment management)

ER Diagram:


email
customer ID
Phone No
customer
customer name
first name
last name
owns
1
N
sets
1
N
wishlist
last updated
wishlist name
wishlist ID
created on
M
contains
N
for
pricealert
alert ID
target price
alert date
status
(active / triggered / expired)
notification type
(email, sms, app)
N
product
1
min price
price range
max price
product ID
product name
tags
description
brand
belongs to
1
N
has
N
price in store
delivery options
discount
inventory
last updated
inventory ID
stock quantity
N
contains
1
1
parent category ID
updated date
created date
category
category name
category ID
Description
category image
has sub category
N
1
website URL
store
store ID
store name
contact number
opening hours
in
1
street
city
location
state
pincode

Relational Schema:



DDL Commands:

```
-- ================================
-- STRONG ENTITIES
-- ================================

CREATE TABLE Customer (
    customer_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100) UNIQUE
);

-- Multivalued attribute: phone_no
CREATE TABLE CustomerPhone (
    customer_id INT,
    phone_no VARCHAR(15),
    PRIMARY KEY (customer_id, phone_no),
    FOREIGN KEY (customer_id) REFERENCES Customer(customer_id) ON DELETE CASCADE
);

CREATE TABLE Category (
    category_id INT PRIMARY KEY,
    category_name VARCHAR(100),
    description TEXT,
    category_image VARCHAR(255),
    parent_category_id INT,
    FOREIGN KEY (parent_category_id) REFERENCES Category(category_id)
        ON DELETE SET NULL   -- Recursive relationship (subcategory → category)
```

```
);

ALTER TABLE Category
ADD COLUMN created_date DATE,
ADD COLUMN updated_date DATE;

ALTER TABLE Category
MODIFY created_date DATETIME DEFAULT CURRENT_TIMESTAMP,
MODIFY updated_date DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP;


CREATE TABLE Store (
    store_id INT PRIMARY KEY,
    store_name VARCHAR(100),
    opening_hours VARCHAR(100),
    website_url VARCHAR(255),
    street VARCHAR(100),
    city VARCHAR(100),
    state VARCHAR(100)
);

ALTER TABLE Store
ADD COLUMN pincode VARCHAR(10);

ALTER TABLE Store
MODIFY pincode VARCHAR(10) NOT NULL;
```

```
-- Multivalued attribute: contact number
CREATE TABLE StoreContact (
    store_id INT,
    contact_number VARCHAR(15),
    PRIMARY KEY (store_id, contact_number),
    FOREIGN KEY (store_id) REFERENCES Store(store_id) ON DELETE CASCADE
);

CREATE TABLE Product (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(100),
    description TEXT,
    brand VARCHAR(50),
    min_price DECIMAL(10,2),
    max_price DECIMAL(10,2),
    category_id INT,
    FOREIGN KEY (category_id) REFERENCES Category(category_id)
);

-- Multivalued attribute: tags
CREATE TABLE ProductTag (
    product_id INT,
    tag VARCHAR(50),
    PRIMARY KEY (product_id, tag),
    FOREIGN KEY (product_id) REFERENCES Product(product_id) ON DELETE CASCADE
);
```

```
CREATE TABLE Wishlist (
    wishlist_id INT PRIMARY KEY,
    wishlist_name VARCHAR(100),
    created_on DATE,
    last_updated DATE,
    customer_id INT,
    FOREIGN KEY (customer_id) REFERENCES Customer(customer_id)
        ON DELETE CASCADE
);

-- M:N relationship between Wishlist and Product
CREATE TABLE WishlistProduct (
    wishlist_id INT,
    product_id INT,
    PRIMARY KEY (wishlist_id, product_id),
    FOREIGN KEY (wishlist_id) REFERENCES Wishlist(wishlist_id) ON DELETE CASCADE,
    FOREIGN KEY (product_id) REFERENCES Product(product_id) ON DELETE CASCADE
);

CREATE TABLE PriceAlert (
    alert_id INT PRIMARY KEY,
    target_price DECIMAL(10,2),
    alert_date DATE,
    status VARCHAR(50),
    notification_type VARCHAR(50),
    product_id INT,
    customer_id INT,
    FOREIGN KEY (product_id) REFERENCES Product(product_id),
```

```
    FOREIGN KEY (customer_id) REFERENCES Customer(customer_id)
);

ALTER TABLE PriceAlert
ADD COLUMN store_id INT,
ADD CONSTRAINT fk_pricealert_store
    FOREIGN KEY (store_id) REFERENCES Store(store_id)
    ON DELETE CASCADE;



-- WEAK ENTITY: Inventory (depends on Product + Store)
CREATE TABLE Inventory (
    inventory_id INT,
    product_id INT,
    store_id INT,
    quantity INT,
    last_updated DATE,
    PRIMARY KEY (inventory_id, product_id, store_id),
    FOREIGN KEY (product_id) REFERENCES Product(product_id) ON DELETE CASCADE,
    FOREIGN KEY (store_id) REFERENCES Store(store_id) ON DELETE CASCADE
);

ALTER TABLE Inventory
ADD COLUMN price_in_store DECIMAL(10,2),
ADD COLUMN delivery_options VARCHAR(100),
ADD COLUMN discount DECIMAL(5,2);
```

CRUD operation Screenshots:

create:

```
mysql> insert into customer values(8,"Ram","Panday","ram@example.com",0);
Query OK, 1 row affected (0.03 sec)

mysql> select * from customer;
+-------------+------------+-----------+---------------------+----------+
| customer_id | first_name | last_name | email               | is_admin |
+-------------+------------+-----------+---------------------+----------+
|           1 | Anika      | P         | anika@example.com   |        0 |
|           2 | Ravi       | Kumar     | ravi@example.com    |        1 |
|           3 | Sneha      | Sharma    | sneha@example.com   |        0 |
|           4 | Amit       | Rao       | amit@example.com    |        0 |
|           5 | Priya      | Menon     | priya@example.com   |        0 |
|           6 | Anish      | M         | anish@example.com   |        0 |
|           7 | Jeevith    | R         | jeevith@example.com |        0 |
|           8 | Ram        | Panday    | ram@example.com     |        0 |
+-------------+------------+-----------+---------------------+----------+
8 rows in set (0.00 sec)
```

read:

```
mysql> select * from customer;
+-------------+------------+-----------+---------------------+----------+
| customer_id | first_name | last_name | email               | is_admin |
+-------------+------------+-----------+---------------------+----------+
|           1 | Anika      | P         | anika@example.com   |        0 |
|           2 | Ravi       | Kumar     | ravi@example.com    |        1 |
|           3 | Sneha      | Sharma    | sneha@example.com   |        0 |
|           4 | Amit       | Rao       | amit@example.com    |        0 |
|           5 | Priya      | Menon     | priya@example.com   |        0 |
|           6 | Anish      | M         | anish@example.com   |        0 |
|           7 | Jeevith    | R         | jeevith@example.com |        0 |
+-------------+------------+-----------+---------------------+----------+
7 rows in set (0.01 sec)
```

update:

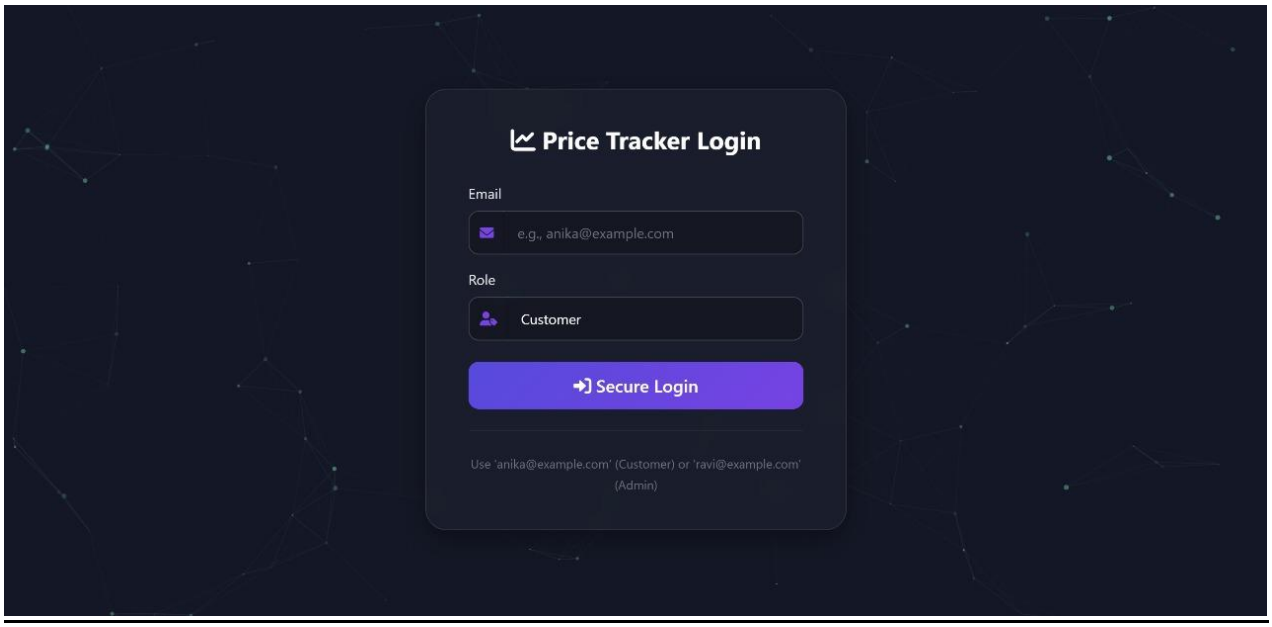
```
mysql> update customer set last_name="Kapoor" where customer_id=2;
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from customer;
+-------------+------------+-----------+---------------------+----------+
| customer_id | first_name | last_name | email               | is_admin |
+-------------+------------+-----------+---------------------+----------+
|           1 | Anika      | P         | anika@example.com   |        0 |
|           2 | Ravi       | Kapoor    | ravi@example.com    |        1 |
|           3 | Sneha      | Sharma    | sneha@example.com   |        0 |
|           4 | Amit       | Rao       | amit@example.com    |        0 |
|           5 | Priya      | Menon     | priya@example.com   |        0 |
|           6 | Anish      | M         | anish@example.com   |        0 |
|           7 | Jeevith    | R         | jeevith@example.com |        0 |
|           8 | Ram        | Panday    | ram@example.com     |        0 |
+-------------+------------+-----------+---------------------+----------+
8 rows in set (0.00 sec)
```

delete:

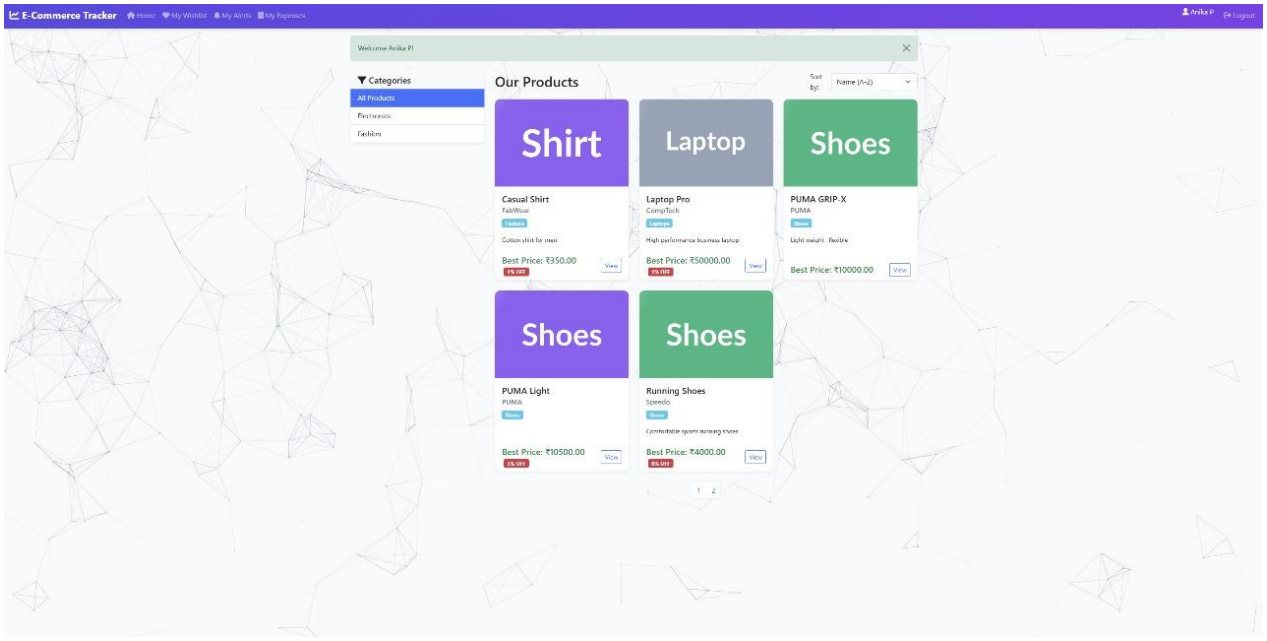
```
mysql> delete from customer where customer_id=8;
Query OK, 1 row affected (0.06 sec)

mysql> select * from customer;
+-------------+------------+-----------+---------------------+----------+
| customer_id | first_name | last_name | email               | is_admin |
+-------------+------------+-----------+---------------------+----------+
|           1 | Anika      | P         | anika@example.com   |        0 |
|           2 | Ravi       | Kapoor    | ravi@example.com    |        1 |
|           3 | Sneha      | Sharma    | sneha@example.com   |        0 |
|           4 | Amit       | Rao       | amit@example.com    |        0 |
|           5 | Priya      | Menon     | priya@example.com   |        0 |
|           6 | Anish      | M         | anish@example.com   |        0 |
|           7 | Jeevith    | R         | jeevith@example.com |        0 |
+-------------+------------+-----------+---------------------+----------+
7 rows in set (0.00 sec)
```

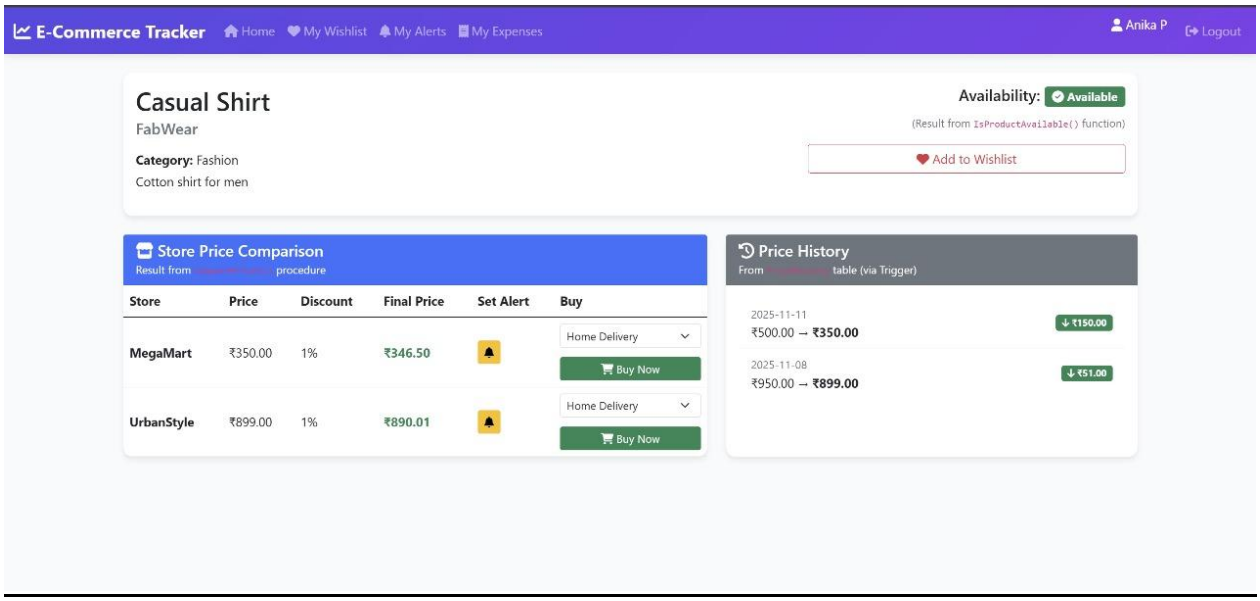
List of functionalities/features of the application screenshots:

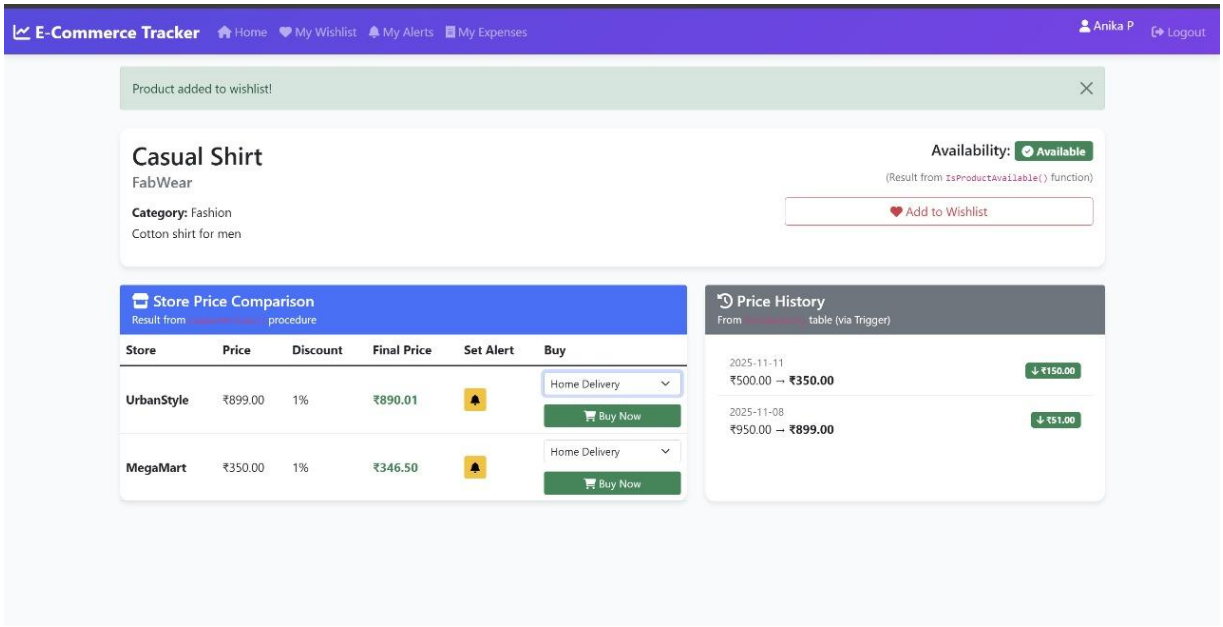
Customer login:

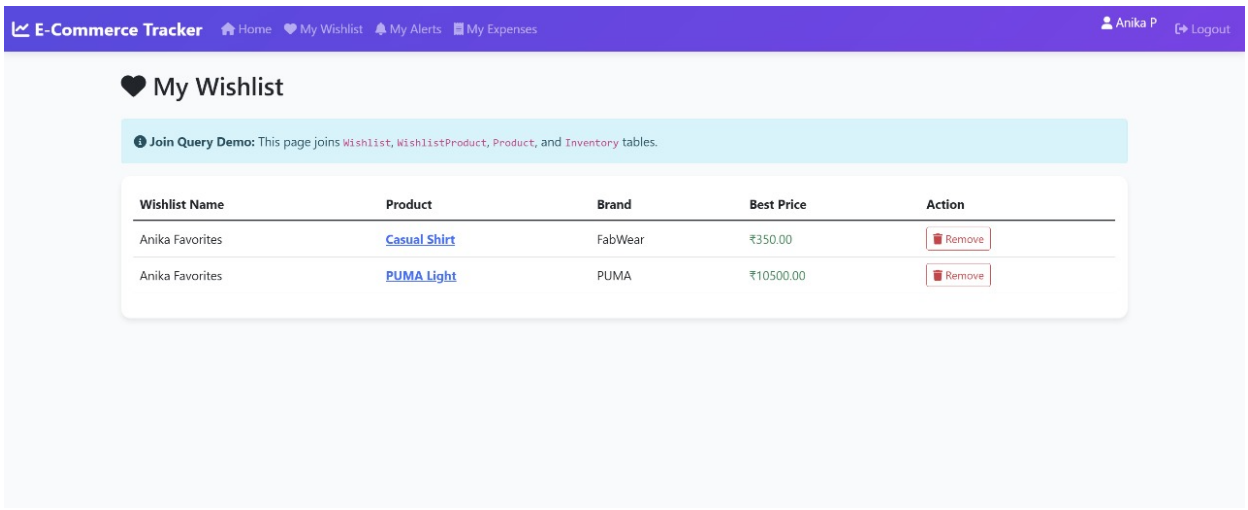


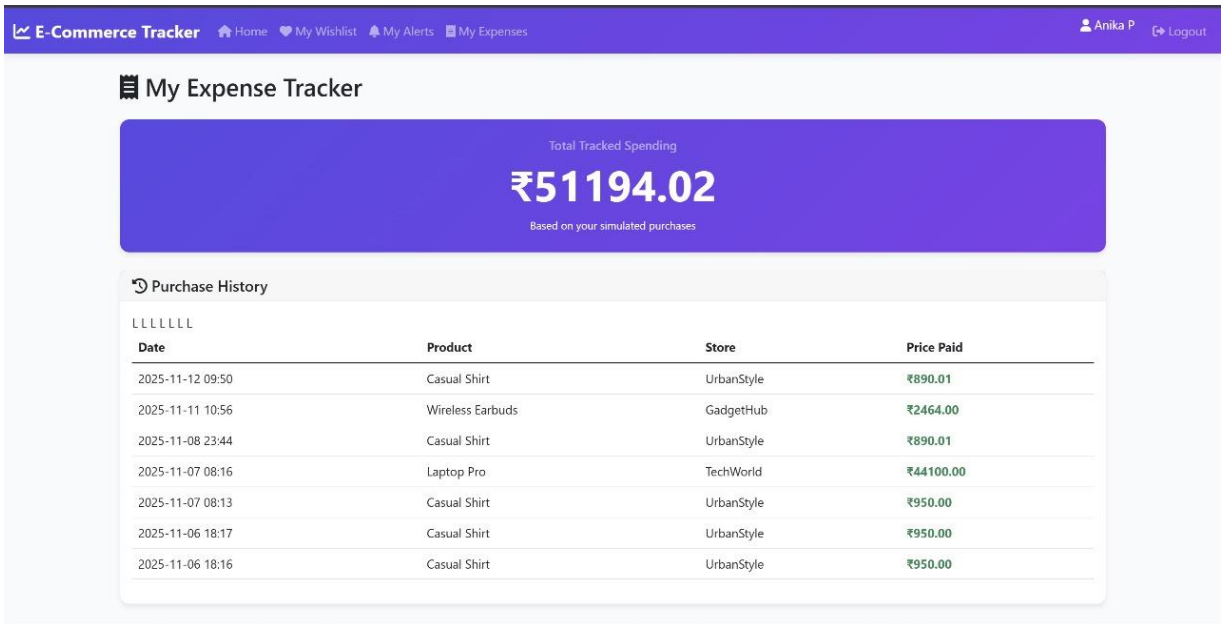
Customer home:

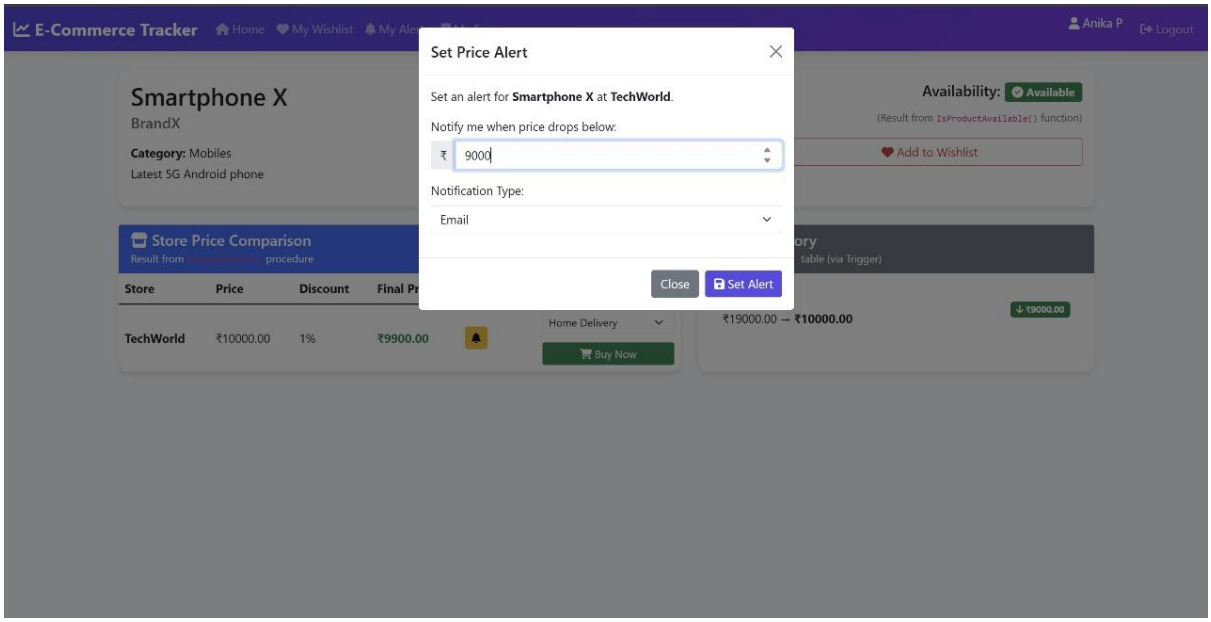




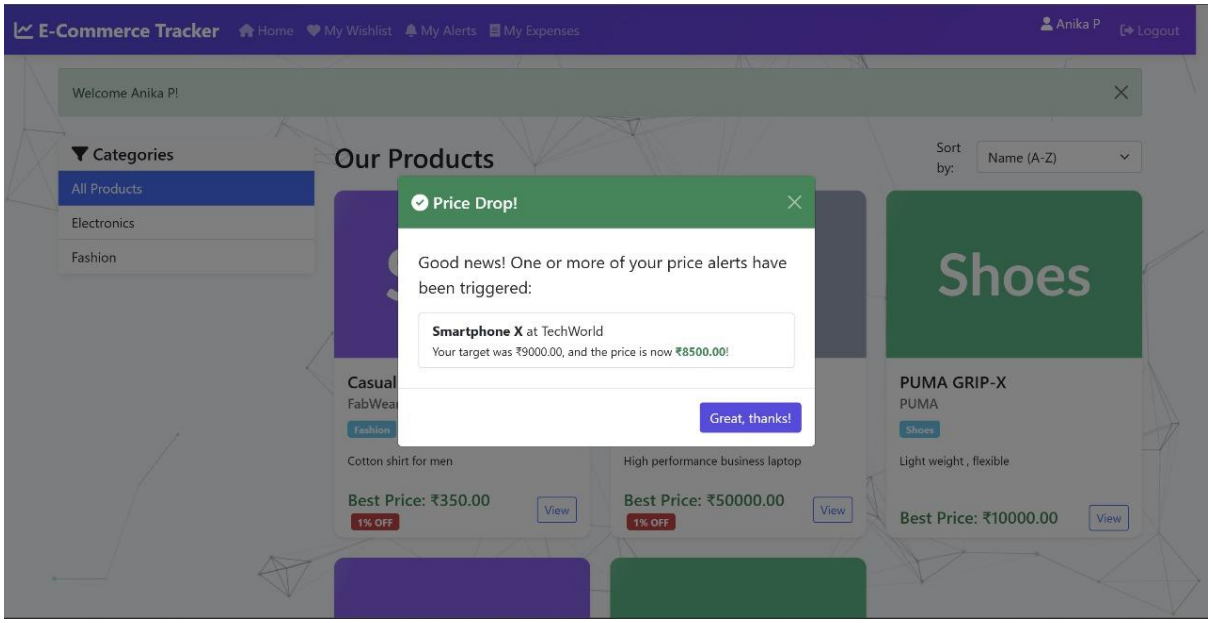

E-Commerce Tracker
Home
My Wishlist
My Alerts
My Expenses
Anika P
Logout
Product added to wishlist!
Casual Shirt
FabWear
Category: Fashion
Cotton shirt for men
Availability:
Available
(Result from IsProductAvailable() function)
Add to Wishlist
Store Price Comparison
Result from procedure
Store
Price
Discount
Final Price
Set Alert
Buy
UrbanStyle
₹899.00
1%
₹890.01
Home Delivery
Buy Now
MegaMart
₹350.00
1%
₹346.50
Home Delivery
Buy Now
Price History
From table (via Trigger)
2025-11-11
₹500.00 → ₹350.00
↓ ₹150.00
2025-11-08
₹950.00 → ₹899.00
↓ ₹51.00

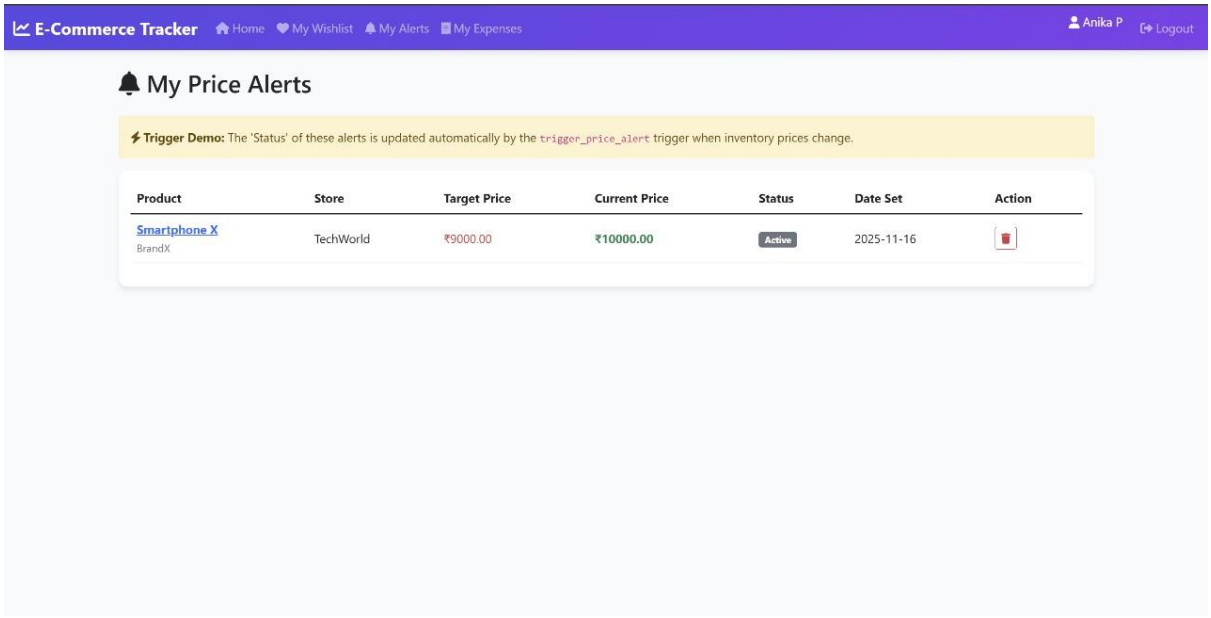

E-Commerce Tracker
Home
My Wishlist
My Alerts
My Expenses
Anika P
Logout
My Wishlist
Join Query Demo: This page joins Wishlist, WishlistProduct, Product, and Inventory tables.
Wishlist Name
Product
Brand
Best Price
Action
Anika Favorites
Casual Shirt
FabWear
₹350.00
Remove
Anika Favorites
PUMA Light
PUMA
₹10500.00
Remove


E-Commerce Tracker
Home
My Wishlist
My Alerts
My Expenses
Anika P
Logout
My Expense Tracker
Total Tracked Spending
₹51194.02
Based on your simulated purchases
Purchase History
LLLLLLL
Date
Product
Store
Price Paid
2025-11-12 09:50
Casual Shirt
UrbanStyle
₹890.01
2025-11-11 10:56
Wireless Earbuds
GadgetHub
₹2464.00
2025-11-08 23:44
Casual Shirt
UrbanStyle
₹890.01
2025-11-07 08:16
Laptop Pro
TechWorld
₹44100.00
2025-11-07 08:13
Casual Shirt
UrbanStyle
₹950.00
2025-11-06 18:17
Casual Shirt
UrbanStyle
₹950.00
2025-11-06 18:16
Casual Shirt
UrbanStyle
₹950.00

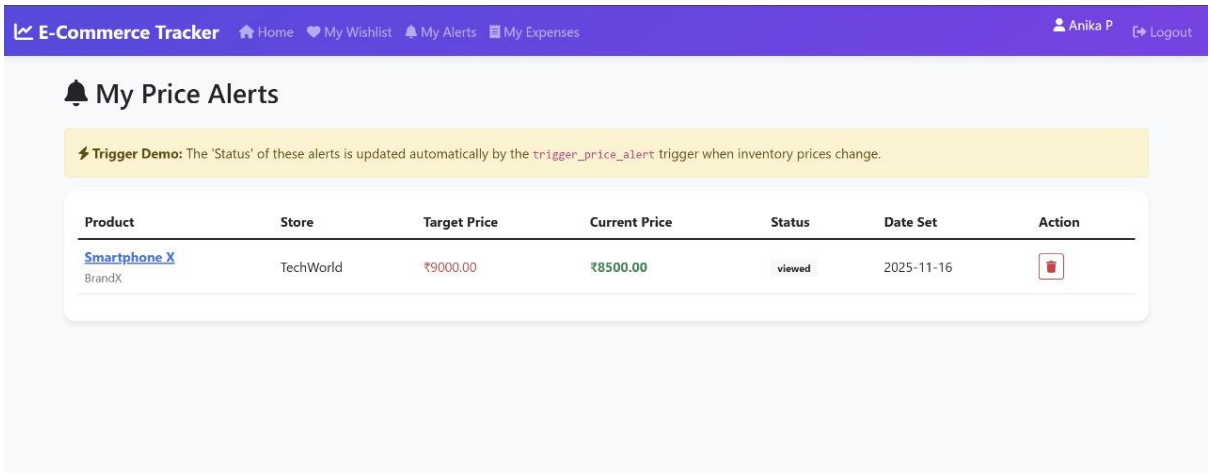


When price meets user requirements:

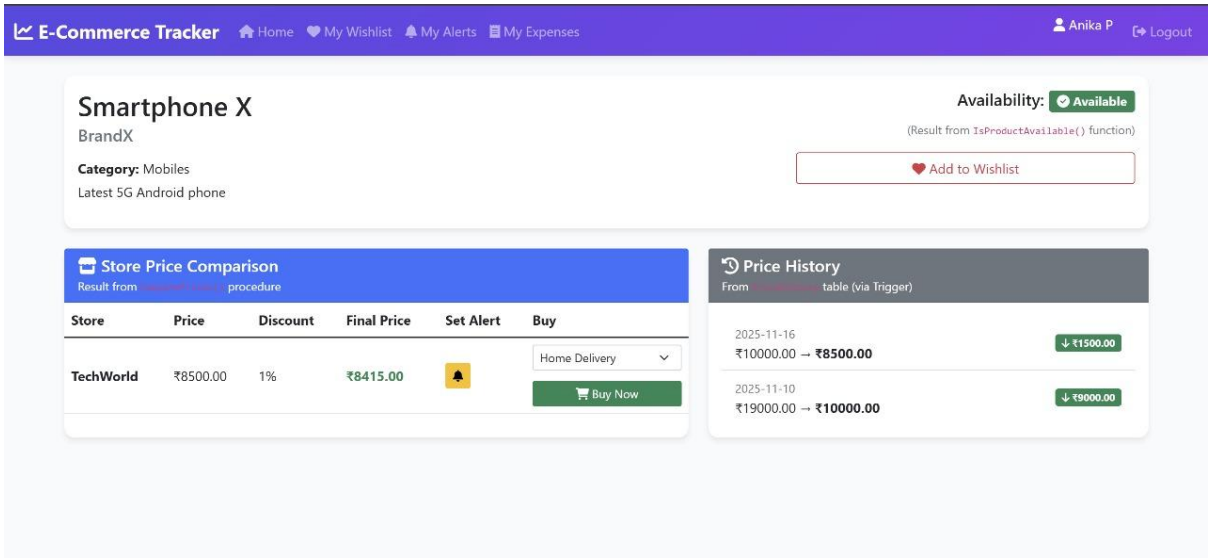




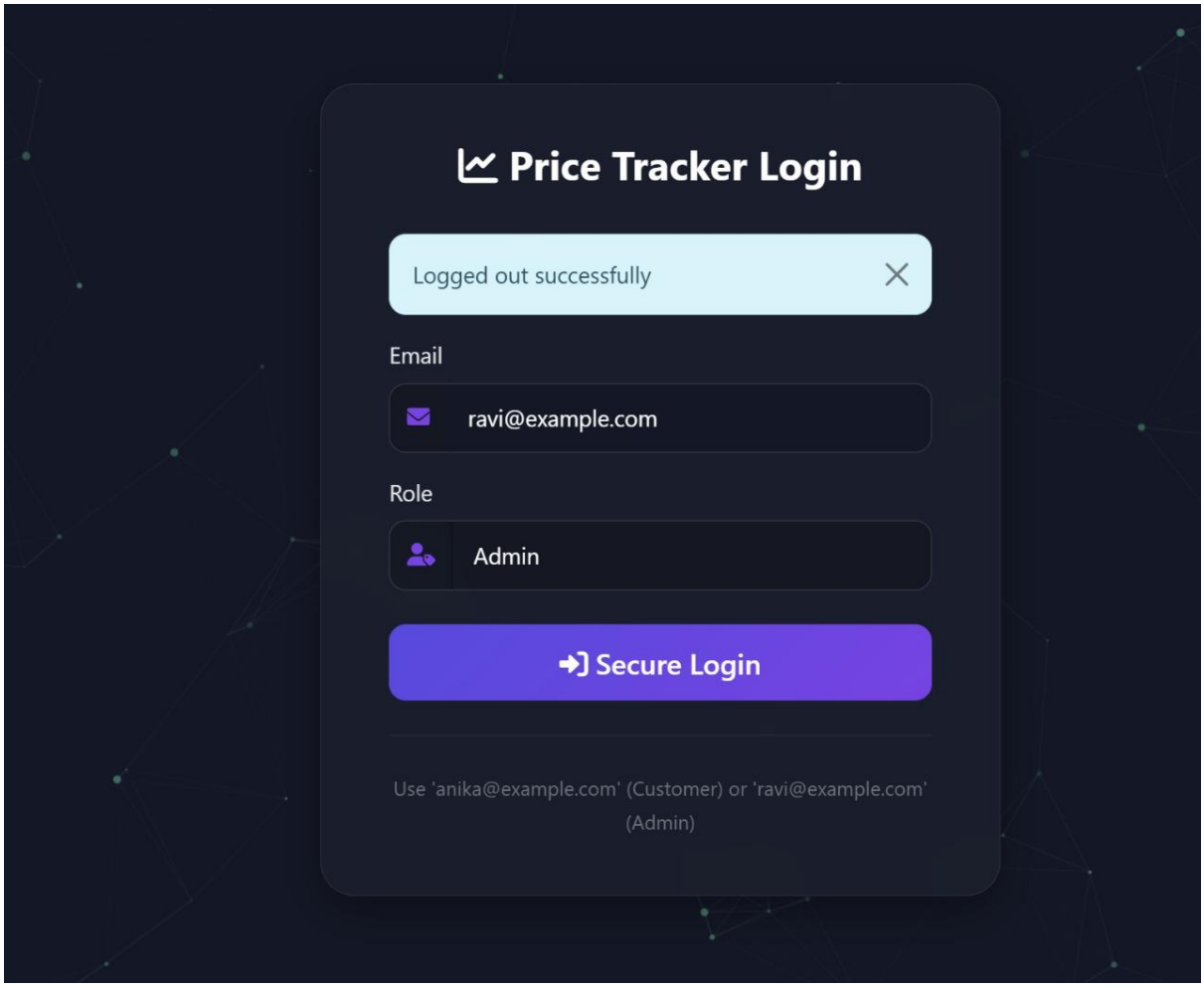
Alerts triggered:



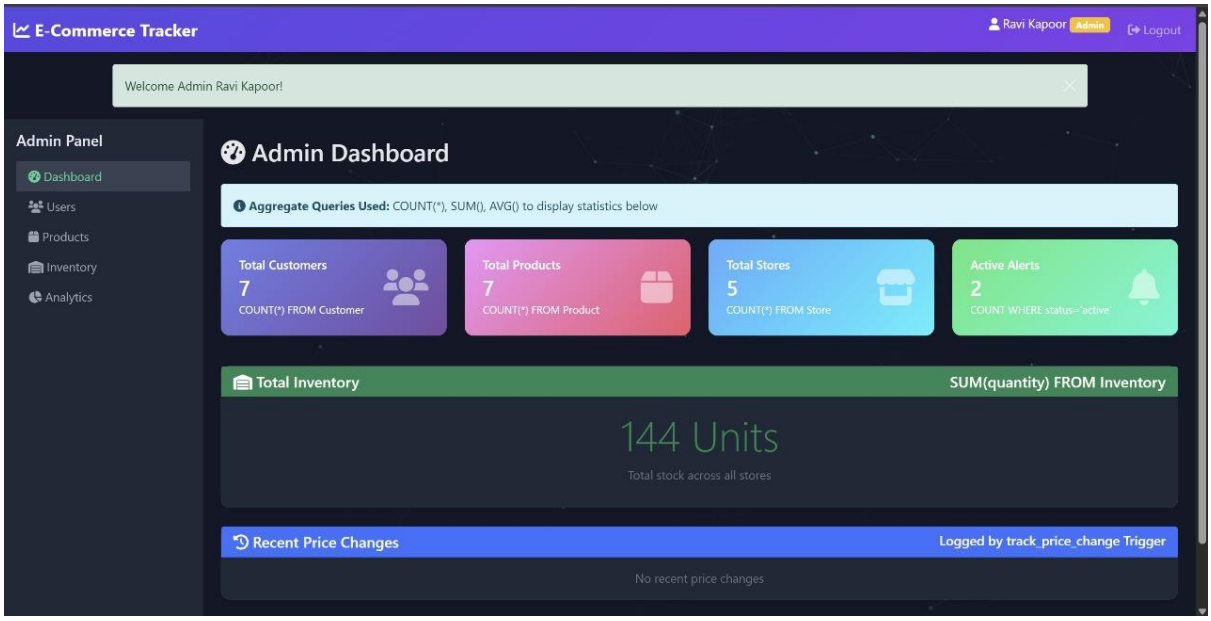
Price history, discount, final price seen in customer page after update by admin



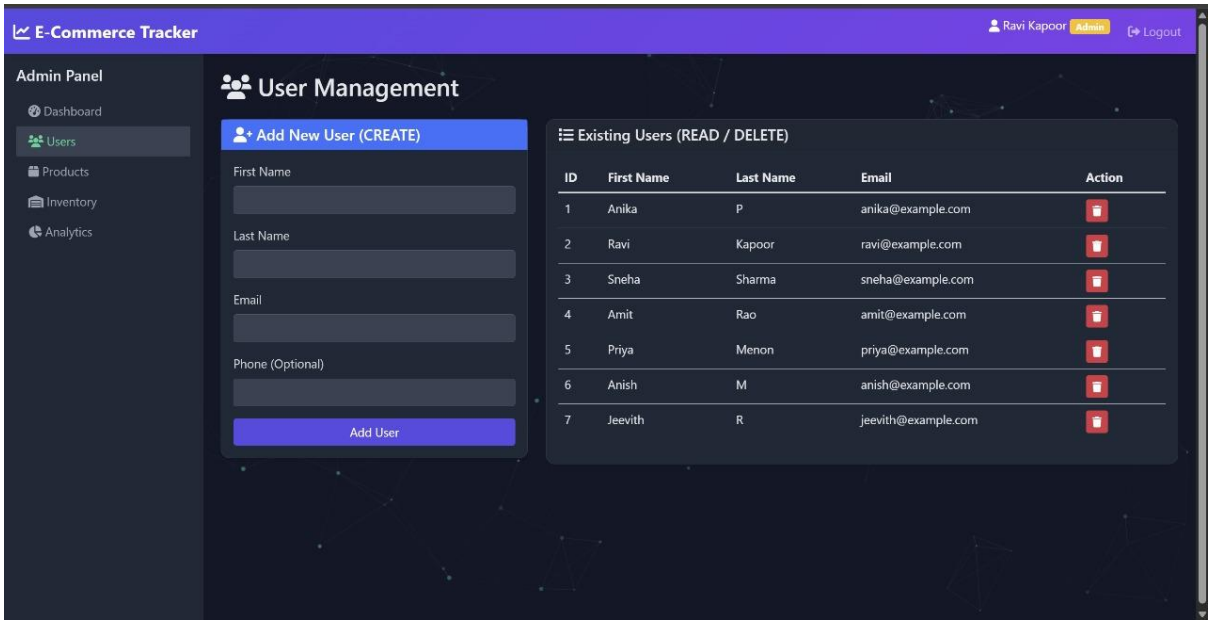
Admin login:



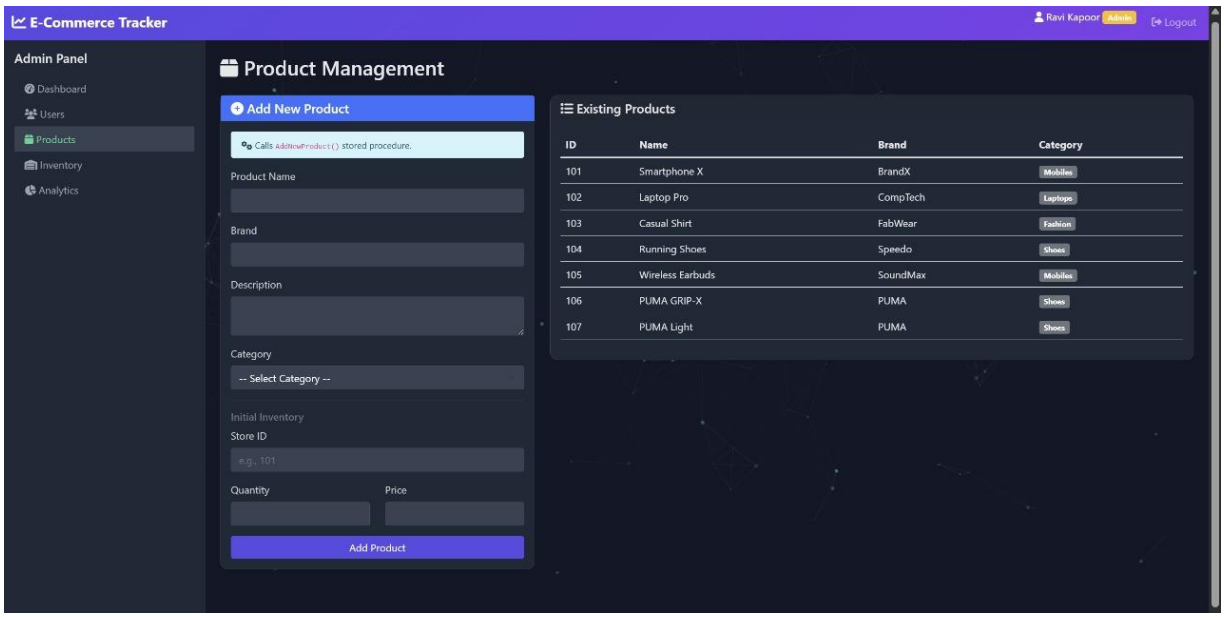
Admin Dashboard:



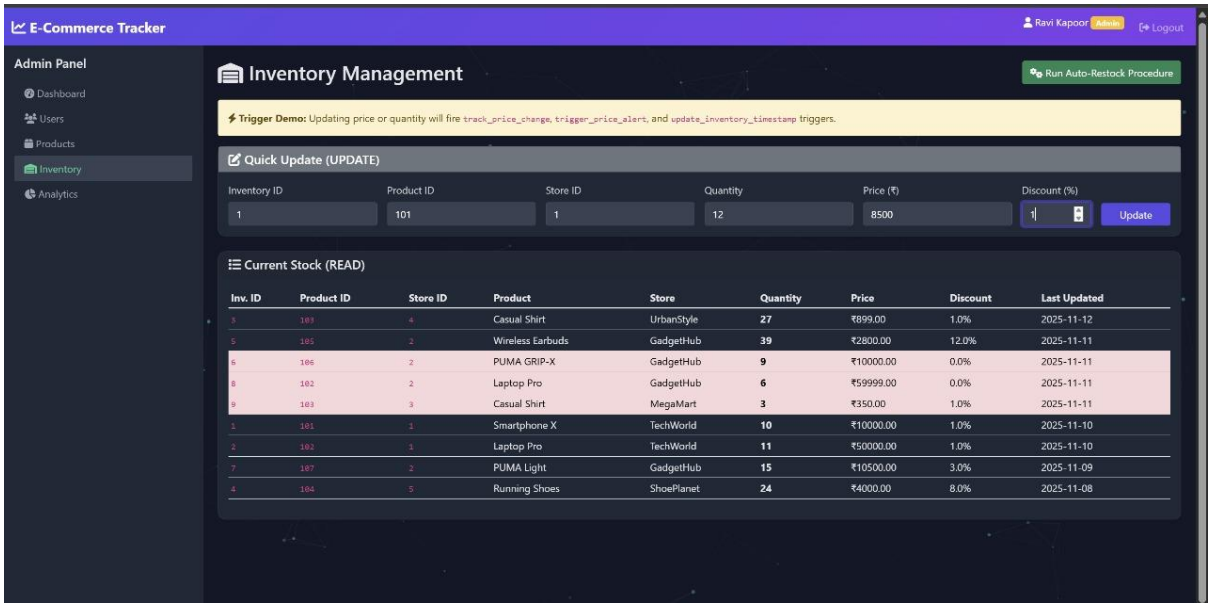
Users visible to the admin:



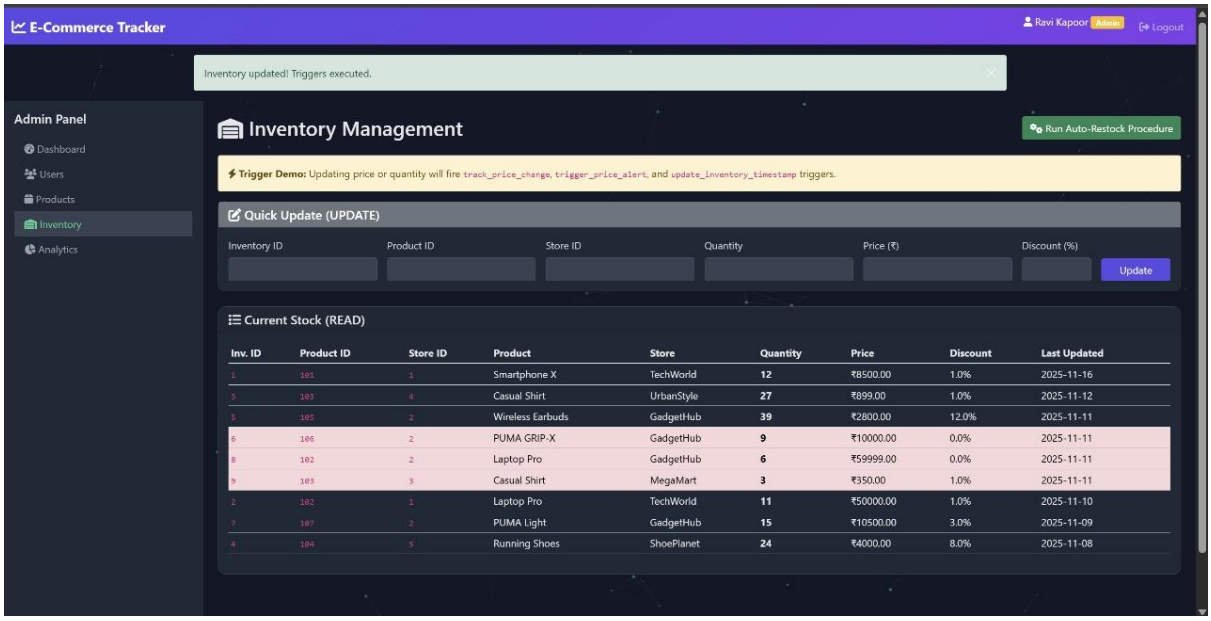
Products visible to the admin:



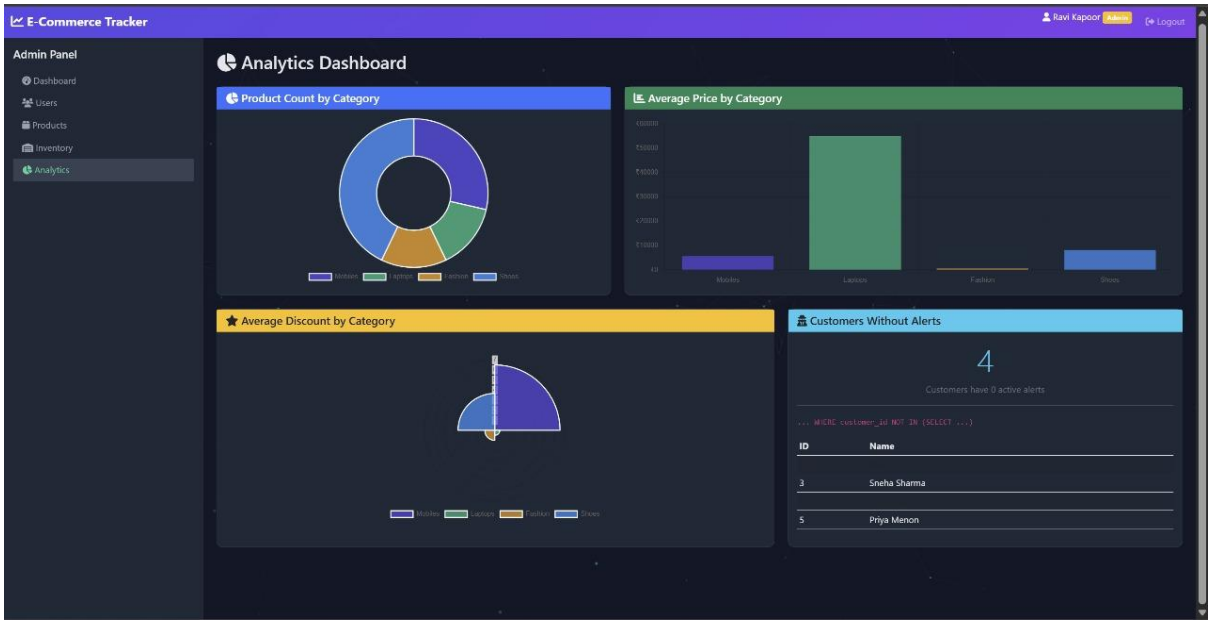
Inventory:



Updated smartphone for alert verification



Analytics:



Triggers, Procedures/Functions, Nested query, Join, Aggregate queries:

Attached in .sql file

Code snippets for invoking the Procedures/Functions/Trigger:

Line 245:

```
228
229  @app.route('/customer/product/<int:product_id>')
230  @login_required
231  def customer_product_detail(product_id):
232      conn = get_db_connection()
233      cursor = conn.cursor(dictionary=True)
234
235      # Get product details
236      cursor.execute("""
237          SELECT p.*, c.category_name
238          FROM Product p
239          JOIN Category c ON p.category_id = c.category_id
240          WHERE p.product_id = %s
241      """, (product_id,))
242      product = cursor.fetchone()
243
244      # Call ComparePrices procedure
245      cursor.callproc('ComparePrices', [product_id])
```

Line 786:

```python
@app.route('/admin/auto_restock')
@admin_required
def admin_auto_restock():
    conn = get_db_connection()
    cursor = conn.cursor()

    try:
        # Call the AutoRestock procedure
        cursor.callproc('AutoRestock')
        conn.commit()
        flash('Auto-restock procedure executed!', 'success')
    except Error as e:
        flash(f'Error running auto-restock: {e}', 'danger')

    cursor.close()
    conn.close()

    return redirect(url_for('admin_inventory'))
```

Line 875:

```python
@app.route('/admin/add_product', methods=['POST'])
@admin_required
def admin_add_product():
    product_name = request.form.get('product_name')
    description = request.form.get('description')
    brand = request.form.get('brand')
    category_id = request.form.get('category_id')
    store_id = request.form.get('store_id')
    quantity = request.form.get('quantity')
    price = request.form.get('price')

    conn = get_db_connection()
    cursor = conn.cursor()

    try:
        # Call AddNewProduct procedure
        cursor.callproc('AddNewProduct',
                        [product_name, description, brand, category_id, store_id, quantity, price])

        conn.commit()
        flash('Product added using stored procedure!', 'success')
    except Error as e:
        flash(f'Error adding product: {e}', 'danger')

    cursor.close()
    conn.close()
```

Github repo link:

https://github.com/anikap2801/E-Commerce-Price-Stock-Tracker-Across-Multiple-Stores