B2C E-commerce

CS232 A Semester Project

Saad Khurshid Qurashi

FCSE/Ai

2023622

U2023622@giki.edu.pk

M1 – Project Proposal

SYNOPSIS

Our project involves the development of a B2C e-commerce website using Python Flask, Postgres SQL, and SQLAlchemy ORM. The website offers a user-friendly interface for customers to browse and purchase products online, with features such as product search, cart management, and secure checkout using the Stripe API.

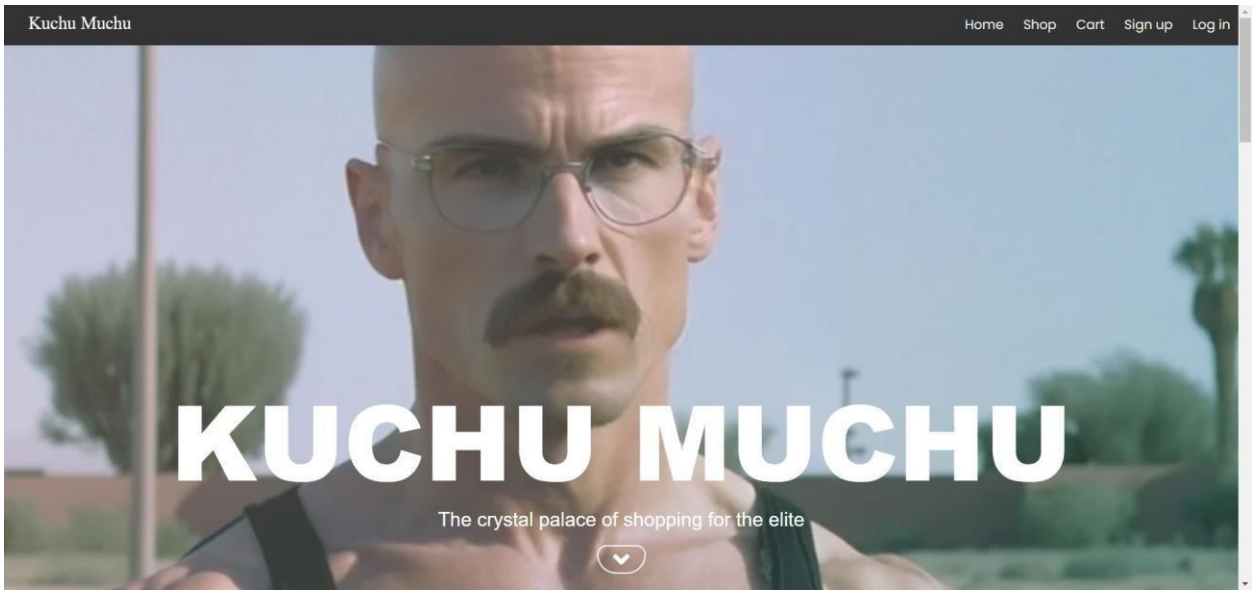
Key Features:

- **User Registration and Authentication:** Customers can create accounts and securely log in to access personalized features and make purchases.
- **Product Catalog:** The website showcases a wide range of products with detailed descriptions, images, and pricing information.
- **Search Functionality:** Customers can easily search for specific products based on keywords or categories, improving their shopping experience.
- **Cart Management:** The website allows customers to add products to their cart, update quantities, and remove items before proceeding to checkout.
- **Secure Checkout:** Integration with the Stripe API ensures secure payment processing, allowing customers to complete transactions with confidence.
- **Admin Panel:** Administrators have access to an admin panel where they can manage products, including adding, editing, and removing items from the inventory.
- **Analytics Dashboard:** The website incorporates Plotly Dash to provide administrators with valuable insights into product performance and revenue, empowering them to make informed business decisions.
- **GSAP Scrolltrigger Animations:** The website utilizes GSAP Scrolltrigger to create engaging and interactive animations, enhancing the overall user experience.

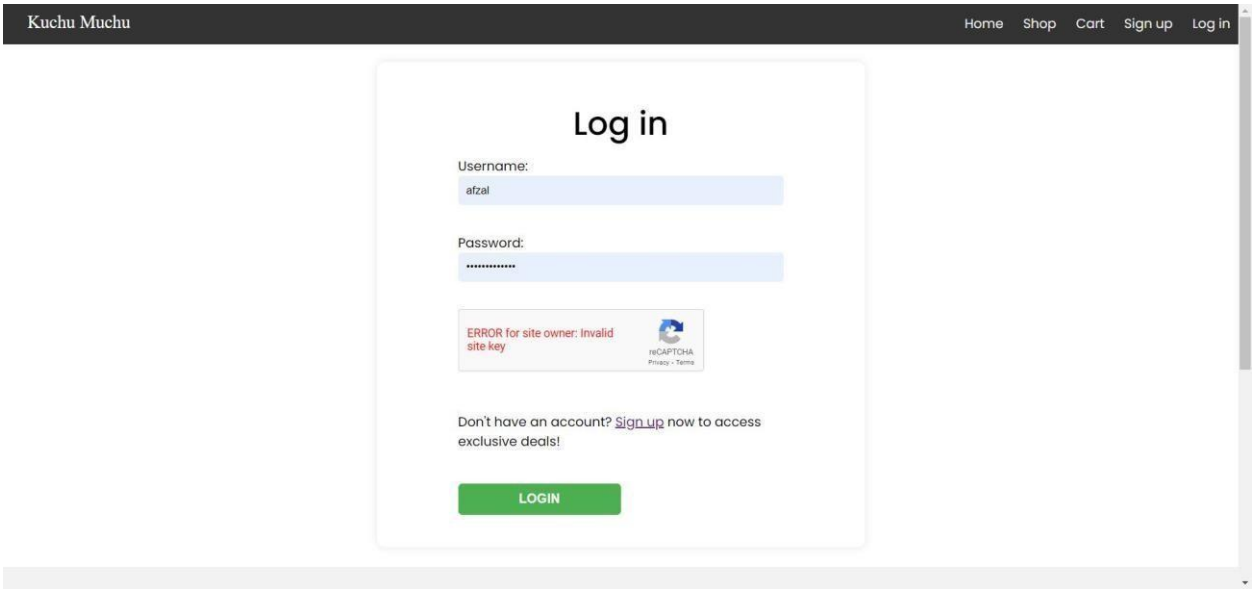
Our project aims to deliver a robust and feature-rich e-commerce website that meets the needs of both customers and administrators. By leveraging Python Flask, Postgres SQL, and SQLAlchemy ORM, we have created a scalable and efficient platform that facilitates seamless online shopping while empowering administrators with valuable analytics data for strategic decision-making.

USER STORY

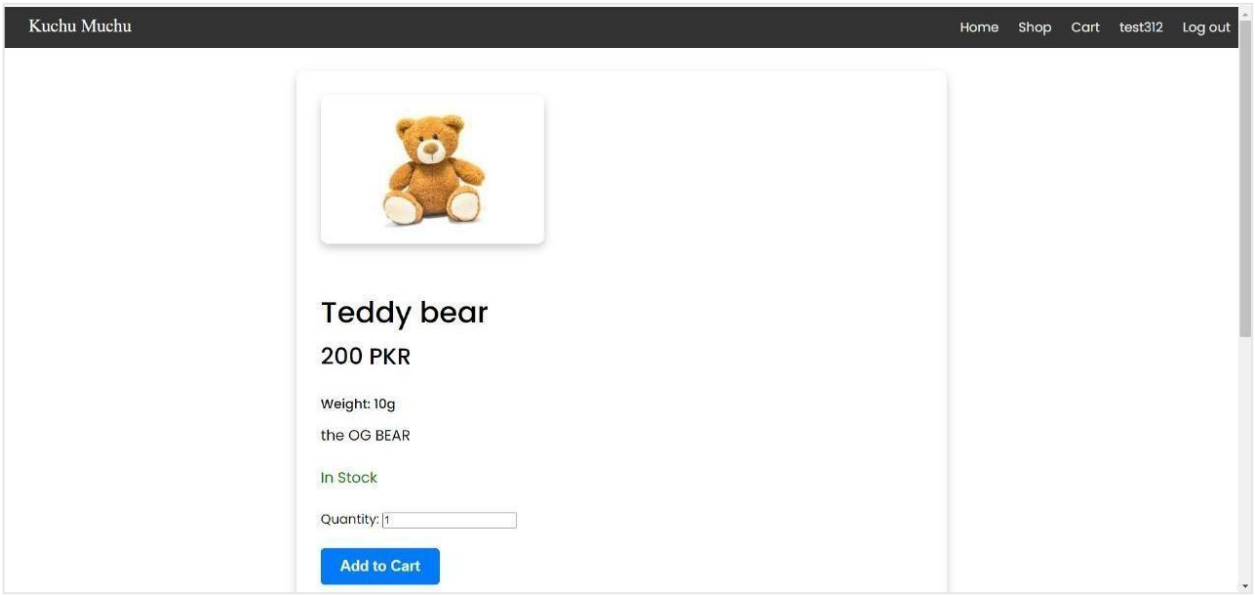
Customers want to have an e-commerce website that is easy to navigate. They expect a smooth and secure checkout process that allows them to make payments using their preferred payment method. The website should also provide clear product descriptions, images, and pricing information to help them make informed purchasing decisions. With our e-commerce website, we aim to exceed customer expectations by delivering a seamless, secure, and enjoyable shopping experience. We prioritize usability, reliability, and accessibility to create an online platform that meets the needs of modern shoppers.



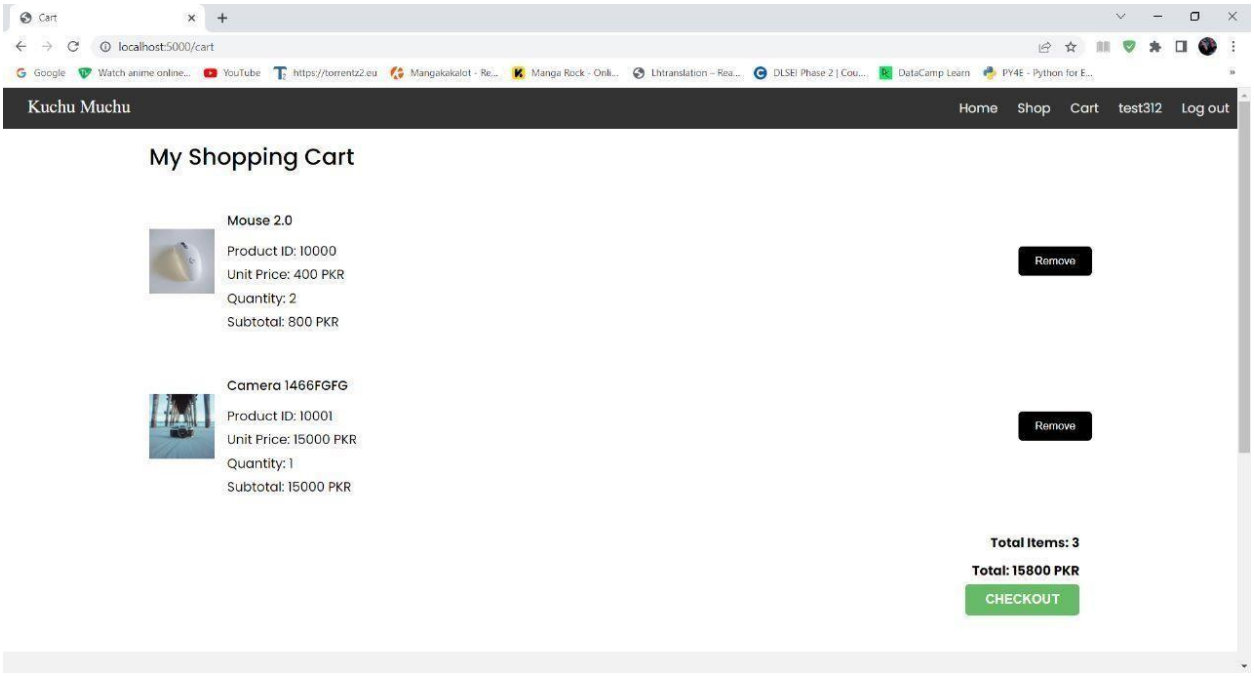
Home page



Customer login page

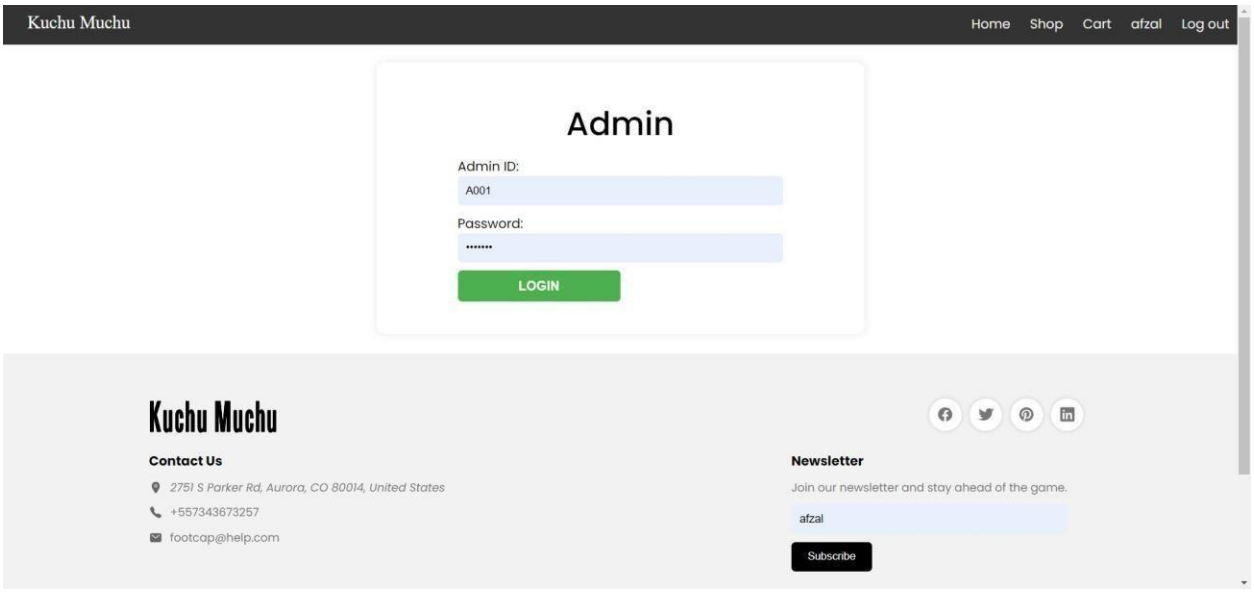


Product page

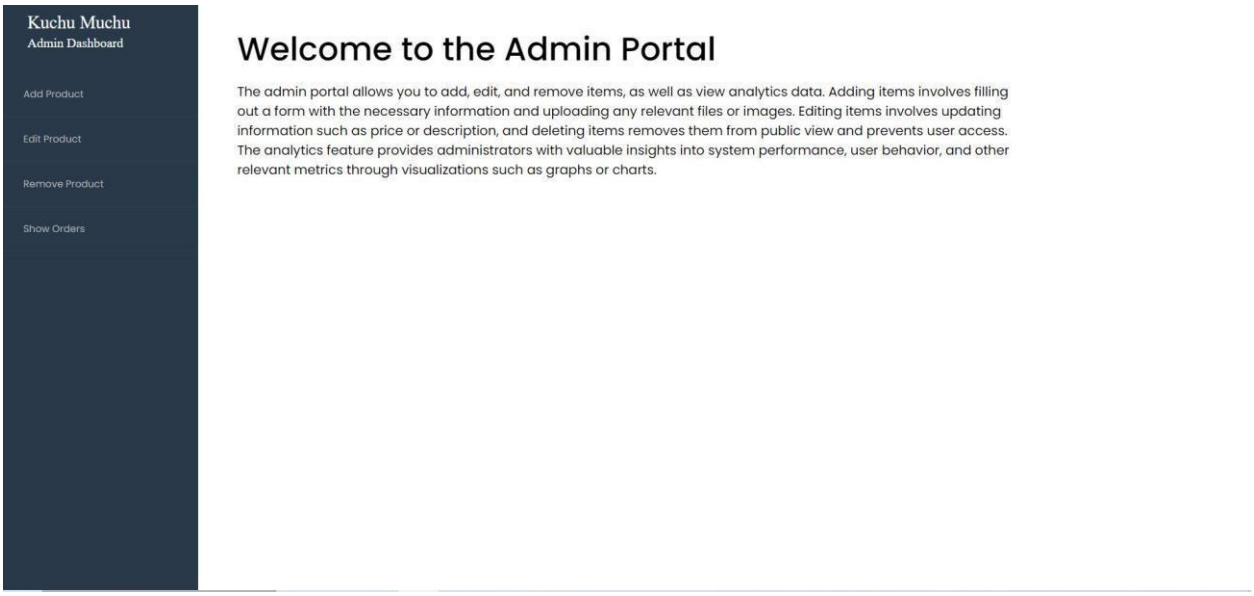


Shopping cart

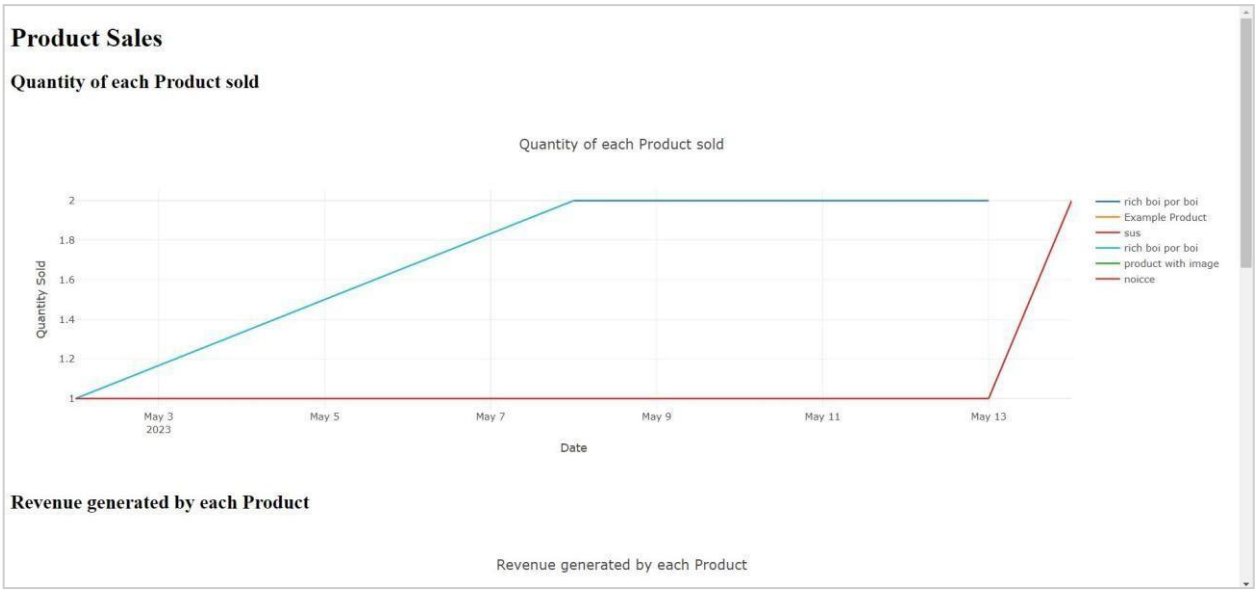
Our admin panel allows administrators to efficiently manage the product inventory. They should be able to add new products, update existing product information, and remove products that are no longer available. The admin panel also provides analytics data such as product performance and revenue to help me monitor the success of different products and make data-driven decisions.



Login page for Admins.



Admin Portal



Analytics Portal

Overall, our e-commerce website provides a seamless and enjoyable shopping experience for customers while offering comprehensive management tools for administrators. We prioritize userfriendly interfaces, responsive design, efficient navigation, and reliable functionality to meet the needs and expectations of both customers and administrators.

TECHNOLOGIES

Our e-commerce website encompasses a powerful stack of cutting-edge technologies that work seamlessly together to provide a robust and efficient platform for online shopping. Here are the key technologies we utilized:

Python Flask Web Framework: We leveraged the Flask framework, a lightweight and flexible Python web framework, to build the foundation of our website. Flask allowed us to develop the application quickly, while providing a solid structure for handling routing, request handling, and server-side logic.

Postgres SQL: We opted for the Postgres SQL database to store and manage our e-commerce data efficiently. Postgres is a reliable and feature-rich relational database management system that offers excellent performance and scalability. It enables us to store and retrieve product information, customer details, and order data with ease.

SQLAlchemy ORM: To streamline our database operations and facilitate interaction with the database, we integrated Flask-SQLAlchemy ORM (Object-Relational Mapping) into our Flask application. SQLAlchemy simplifies database queries and manipulations by allowing us to work with Python objects directly, abstracting the underlying SQL syntax.

Stripe API: Ensuring a secure and seamless payment experience is crucial for any e-commerce platform. We integrated the Stripe API, a popular payment gateway, to handle online transactions securely. Stripe offers a wide range of payment options, including credit cards, digital wallets, and other popular payment methods, providing our customers with flexibility and peace of mind.

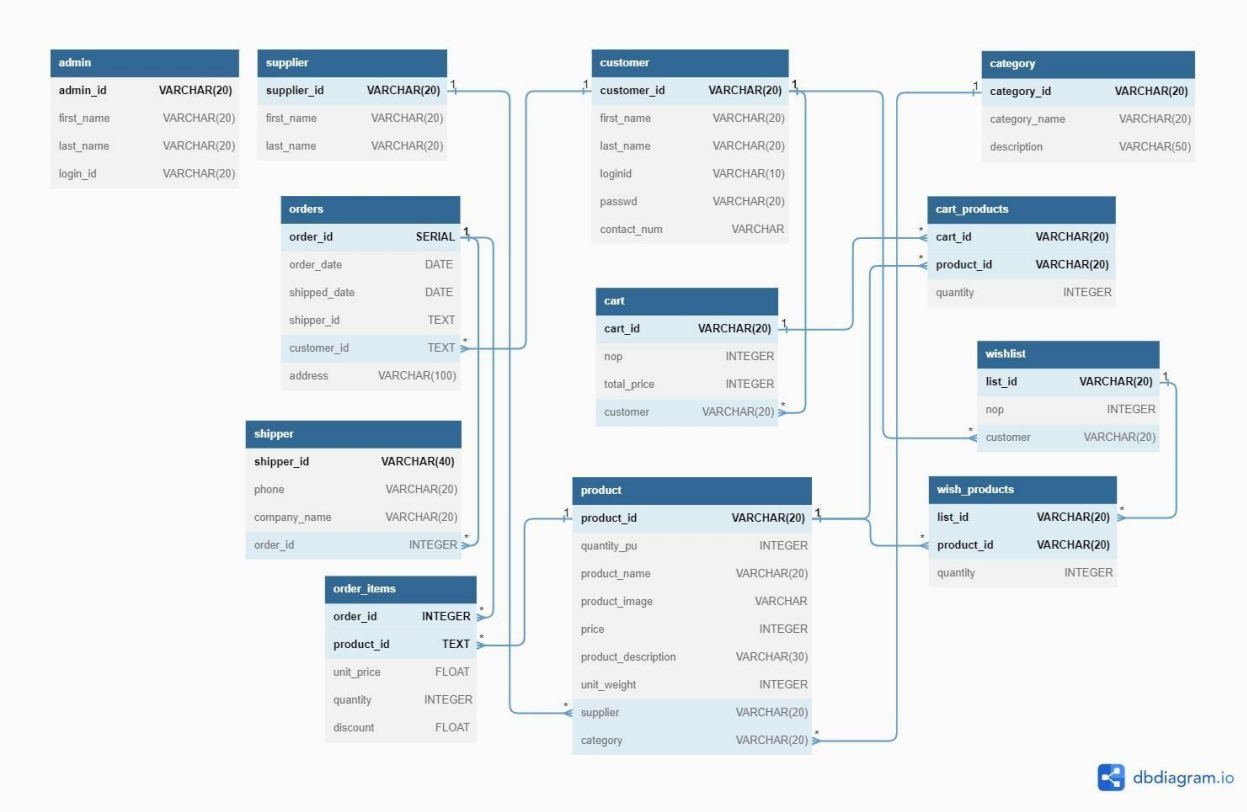
GSAP ScrollTrigger: To enhance the user experience and add engaging visual elements, we implemented the GSAP ScrollTrigger library. This powerful JavaScript animation library allowed us to create captivating scroll-based animations and interactive effects, bringing our website to life and making it more visually appealing.

Plotly Dash: Understanding the performance of our products and the overall revenue generated is vital for informed decision-making. To provide comprehensive analytics, we integrated Plotly Dash, a Python framework for creating interactive dashboards and data visualizations. With Dash, we can present meaningful insights, such as product performance and revenue trends.

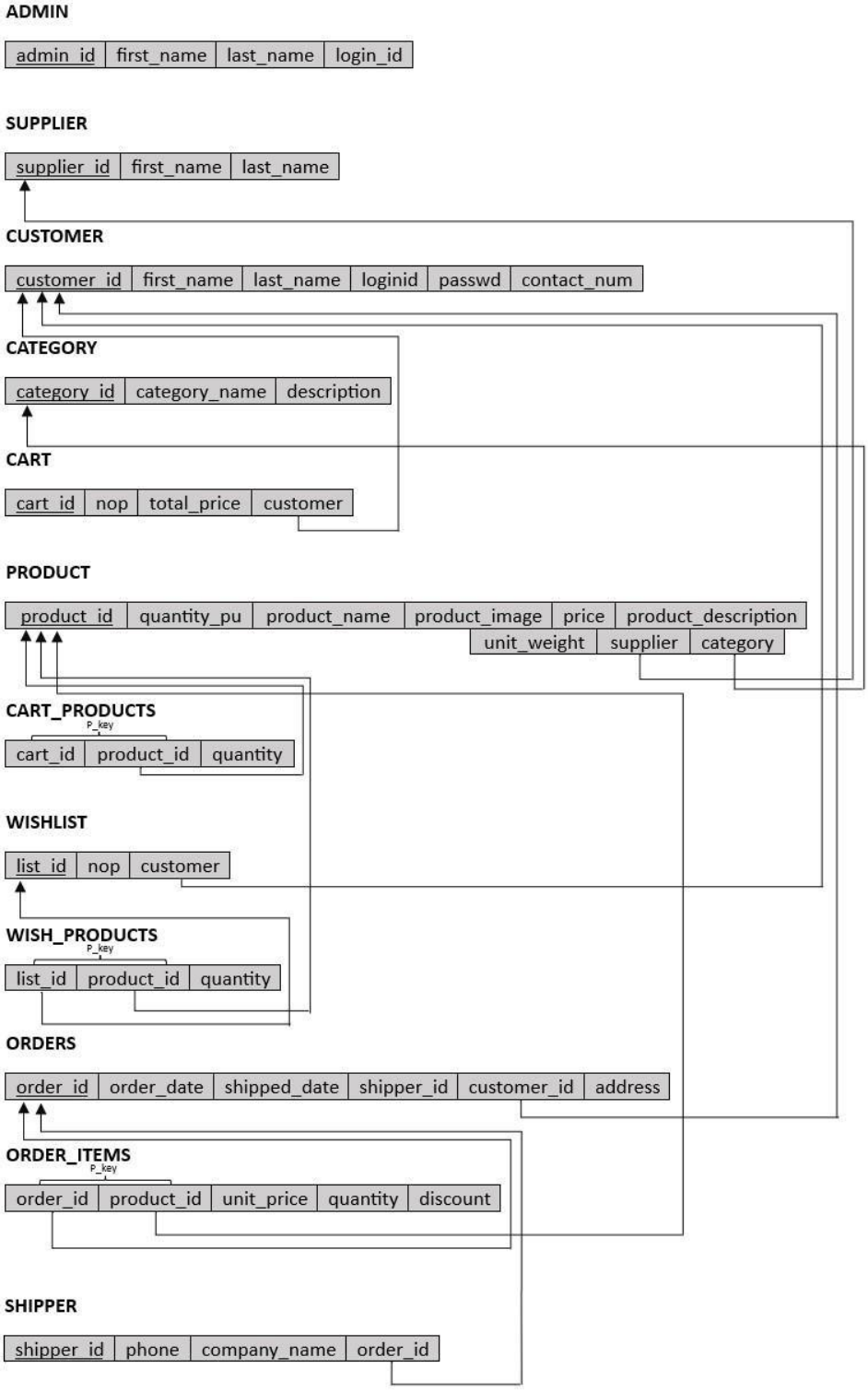
By harnessing the capabilities of these advanced technologies, we have built a sophisticated and user-friendly e-commerce website that offers an exceptional shopping experience while ensuring security, scalability, and data-driven insights.

Design

M2 - ERD



M3 - Relational Schema



Implementation

M4 – Basic SQL

```sql
CREATE TABLE product (
    product_id VARCHAR(20) PRIMARY KEY,
    quantity_pu INTEGER,
    product_name VARCHAR(20),
    product_image VARCHAR,
    price INTEGER,
    product_description VARCHAR(30),
    unit_weight INTEGER,
    supplier VARCHAR(20),
    category VARCHAR(20),
    FOREIGN KEY (supplier) REFERENCES supplier (supplier_id),
    FOREIGN KEY (category) REFERENCES category (category_id)
);

```

```sql
CREATE TABLE order_items (
    order_id INTEGER,
    product_id TEXT,
    unit_price FLOAT NOT NULL,
    quantity INTEGER NOT NULL,
    discount FLOAT NOT NULL,
    PRIMARY KEY (order_id, product_id),
    FOREIGN KEY (order_id) REFERENCES orders (order_id),
    FOREIGN KEY (product_id) REFERENCES product (product_id)
);
```

```sql
INSERT INTO category (category_id, category_name, description)
VALUES ('C003', 'Miscellaneous', 'Miscellaneous products');

insert into supplier values('ABC123','Ali','Khan');

INSERT INTO product (product_id, quantity_pu, product_name,product_image, unit_weight, price, product_description, supplier, category)
VALUES ('10000', 10, 'Mouse 2.0','mouse.JPG', 20,  400, 'A good camera fella', 'ABC123', 'C001'),
('10001', 10, 'Camera 1466FGFG','camera.JPG', 400,  15000, 'the OG camera', 'ABC123', 'C001'),
('10002',10,'Small adapters 2.0','smalladapters.JPG',50,800, 'adapters for everything ','ABC123','C001');
```

M5 – Advanced SQL

Triggers

```
CREATE OR REPLACE FUNCTION update_product_quantity_on_cart_product_add()
RETURNS TRIGGER AS $$
BEGIN
    UPDATE product
    SET quantity_pu = quantity_pu - NEW.quantity
    WHERE product_id = NEW.product_id;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER update_product_quantity_trigger
AFTER INSERT ON cart_products
FOR EACH ROW
EXECUTE FUNCTION update_product_quantity_on_cart_product_add();
```

We implemented a trigger function that reduces the quantity of the product once it is added to the card.

```
# Define a function to update product quantity when a cart product is added
def update_product_quantity_on_cart_product_add(mapper, connection, target):
    product_id = target.product_id
    quantity = target.quantity
    product = Product.query.get(product_id)
    product.quantity_pu -= quantity

# Register the trigger function to execute when a CartProduct object is added
event.listen(CartProduct, 'after_insert', update_product_quantity_on_cart_product_add)
```

event.listen function of SQLAlchemy library can also be used instead of SQL triggers.

Function

(calling functions using an ORM library)

```
182  CREATE OR REPLACE FUNCTION count_it()
183  RETURNS integer
184  LANGUAGE plpgsql
185  AS
186  $$
187  DECLARE
188      pcount int;
189  BEGIN
190      SELECT count(*) INTO pcount FROM product;
191      RETURN pcount;
192  END;
193  $$;
```

```
221  @app.route("/view")
222  def view():
223      Data = db.session.query(MyView).all()
224
225      cursor = conn.cursor()
226
227      # Call the function
228      cursor.execute("SELECT count_it();")
229
230      # Fetch the result
231      result = cursor.fetchone()[0]
232
233      # Close the cursor and the connection
234      cursor.close()
235      conn.close()
236      return render_template('view.html', AllData=Data, count=result)
```

View

Calling view is done in ORM as if it is an actual table. We used class ProductView to show all the products in the customer's cart.

```sql
CREATE VIEW product_view AS
SELECT p.*, c.customer, cp.quantity
FROM product p
JOIN cart_products cp ON p.product_id = cp.product_id
JOIN cart c ON cp.cart_id = c.cart_id;
```

```python
class ProductView(db.Model):
    __tablename__ = 'product_view'

    product_id = db.Column(db.String(20), primary_key=True)
    quantity_pu = db.Column(db.Integer)
    product_name = db.Column(db.String(20))
    product_image = db.Column(db.String)
    price = db.Column(db.Integer)
    product_description = db.Column(db.String(30))
    category = db.Column(db.ForeignKey('category.category_id'))
    quantity = db.Column(db.Integer)
    customer = db.Column(db.ForeignKey('customer.customer_id'))
```

We have used the class OrderInfo to show orders.

```sql
CREATE VIEW order_info AS
SELECT
    o.order_id,
    o.order_date,
    o.shipped_date,
    o.shipper_id,
    o.customer_id,
    c.first_name || ' ' || c.last_name AS customer_name,
    o.address,
    oi.product_id,
    p.product_name,
    oi.unit_price,
    oi.quantity
FROM
    orders o
    JOIN order_items oi ON o.order_id = oi.order_id
    JOIN customer c ON o.customer_id = c.customer_id
    JOIN product p ON oi.product_id = p.product_id;
```

```python
class OrderInfo(db.Model):
    __tablename__ = 'order_info'

    order_id = db.Column(db.Integer, primary_key=True)
    order_date = db.Column(db.Date, nullable=False)
    shipped_date = db.Column(db.Date)
    shipper_id = db.Column(db.Integer, db.ForeignKey('shipper.shipper_id'))
    customer_id = db.Column(db.Integer, db.ForeignKey('customer.customer_id'))
    customer_name = db.Column(db.String(40))
    address = db.Column(db.String(100))
    product_id = db.Column(db.Integer, db.ForeignKey('product.product_id'))
    product_name = db.Column(db.String(20))
    unit_price = db.Column(db.Float, nullable=False)
    quantity = db.Column(db.Integer, nullable=False)

    product = db.relationship("Product")
```

Source Code

```
git clone https://github.com/Isaadqurashi/B2C-Ecommerce
```