Online Store

Nikhil Mahalingam & Andrew Xie

Project Goals

- Construct a robust online store database system
- Handle inventory of various quantities and types of products
- Create stored procedures for updating inventory
- Create a user registration and login system
- Learn about new databases!

craigslist

ebay™

Stack

- HTML/CSS

- Python + Flask

- Databases:
  - PostgreSQL: Open source relational database
    - Data storage

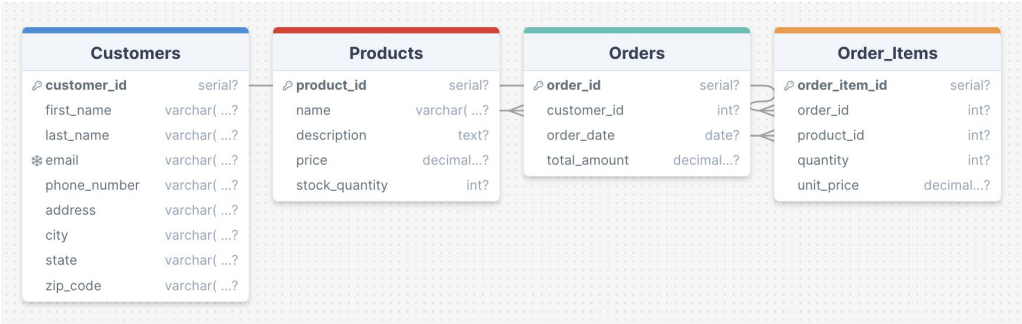
  - SQLAlchemy: Object relational mapper (ORM) for Python
    - Querying

  - Supabase: Open source Firebase alternative - PostgreSQL database
    - User management + Authentication

  - AWS S3: Cloud storage key-value store database
    - Image stored in buckets

Table Design

- Simple design
  - Still many variables/testing/edge cases to take into account
- Customers
  - **User info**
  - Authenticated using Supabase
- Products
  - **Inventory, name, description**
- Orders
  - Enum status: cart, pending, completed
  - **Total price**, customer, orderID
- OrderItems - Items in an Order
  - Items, unit price



Schema Code

```sql
-- CUSTOMERS
CREATE TABLE Customers (
    customer_id SERIAL PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100) UNIQUE,
    phone_number VARCHAR(20),
    address VARCHAR(255),
    city VARCHAR(100),
    state VARCHAR(50),
    zip_code VARCHAR(20)
);
```

```sql
CREATE TYPE order_status AS ENUM ('cart', 'pending', 'completed');

-- ORDERS
CREATE TABLE Orders (
    order_id SERIAL PRIMARY KEY,
    customer_id INT,
    order_date DATE,
    total_amount DECIMAL(10, 2),
    status order_status NOT NULL,
    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id)
);
```

```sql
-- PRODUCTS
CREATE TABLE Products (
    product_id SERIAL PRIMARY KEY,
    name VARCHAR(100),
    description TEXT,
    slug VARCHAR(255),
    price DECIMAL(10, 2),
    stock_quantity INT
);
```

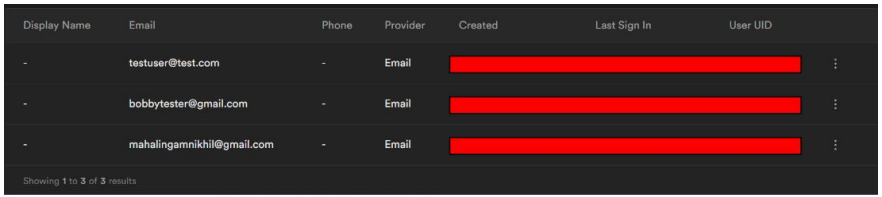
```sql
-- ORDER ITEMS (ITEMS IN AN ORDER)
CREATE TABLE Order_Items (
    order_item_id SERIAL PRIMARY KEY,
    order_id INT,
    product_id INT,
    quantity INT,
    unit_price DECIMAL(10, 2),
    FOREIGN KEY (order_id) REFERENCES Orders(order_id),
    FOREIGN KEY (product_id) REFERENCES Products(product_id)
);
```

Features/Demo

User Authentication

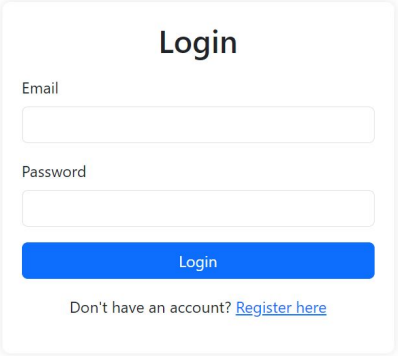
supabase

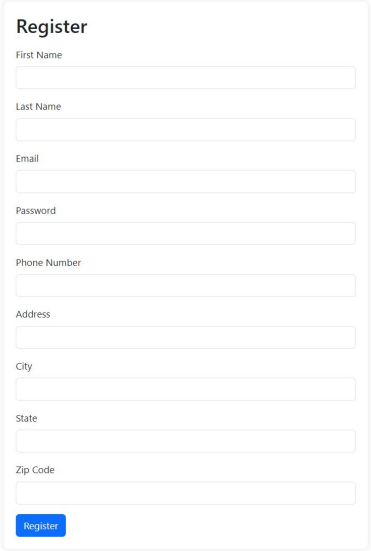
- Handled with supabase
  - Login/Registration
  - Unique email
    - Get email from session on login
    - Query PostgreSQL database with email to get Customer_ID
- Protected Routes
  - Must be logged in to access pages



Display Name	Email	Phone	Provider	Created	Last Sign In	User UID
-	testuser@test.com	-	Email	[redacted]		
-	bobbytester@gmail.com	-	Email	[redacted]		
-	mahalingamnikhil@gmail.com	-	Email	[redacted]		

Showing 1 to 3 of 3 results





User Authentication Code

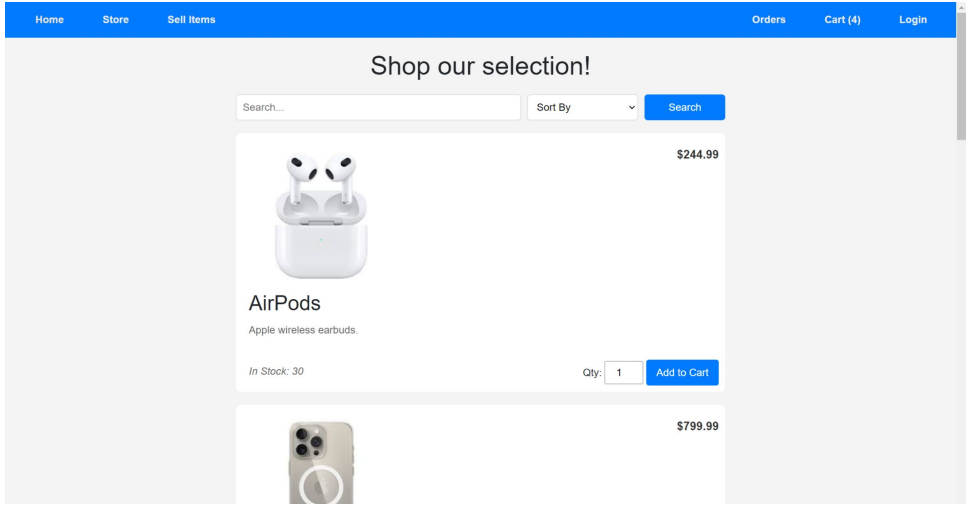
```python
#Routes for Functions
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']

        try:
            result = supabase.auth.sign_up({'email':email, 'password':password})
            print("Supabase result:", result)

            if result:
                sql = """
                INSERT INTO customers (first_name, last_name, email, phone_number, address, city, state, zip_code)
                VALUES (:first_name, :last_name, :email, :phone_number, :address, :city, :state, :zip_code);
                """
                db.session.execute(text(sql), {
                    'first_name': request.form['first_name'],
                    'last_name': request.form['last_name'],
                    'email': email,
                    'phone_number': request.form['phone_number'],
                    'address': request.form['address'],
                    'city': request.form['city'],
                    'state': request.form['state'],
                    'zip_code': request.form['zip_code']
                })
                db.session.commit()

                flash('Registration successful! Please log in.', 'success')
                return redirect(url_for('login'))
            else:
                flash('Failed to register with Supabase. Please try again.', 'error')
                return redirect(url_for('register'))

        except Exception as e:
            db.session.rollback()
            flash(f'Registration failed: {str(e)}', 'error')
            return redirect(url_for('register'))

    return render_template('register.html')
```

```python
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']

        response = supabase.auth.sign_in_with_password({"email":email, "password":password})
        if response:
            session_obj = supabase.auth.get_session()
            if session_obj:
                    access_token = session_obj.access_token
                    session['access_token'] = access_token
                    flash('Login successful!', 'success')

            return redirect(url_for('index'))
        else:
                flash('Login failed. Please check your credentials.', 'danger')
                return redirect(url_for('login'))
    return render_template('login.html')
```

Store

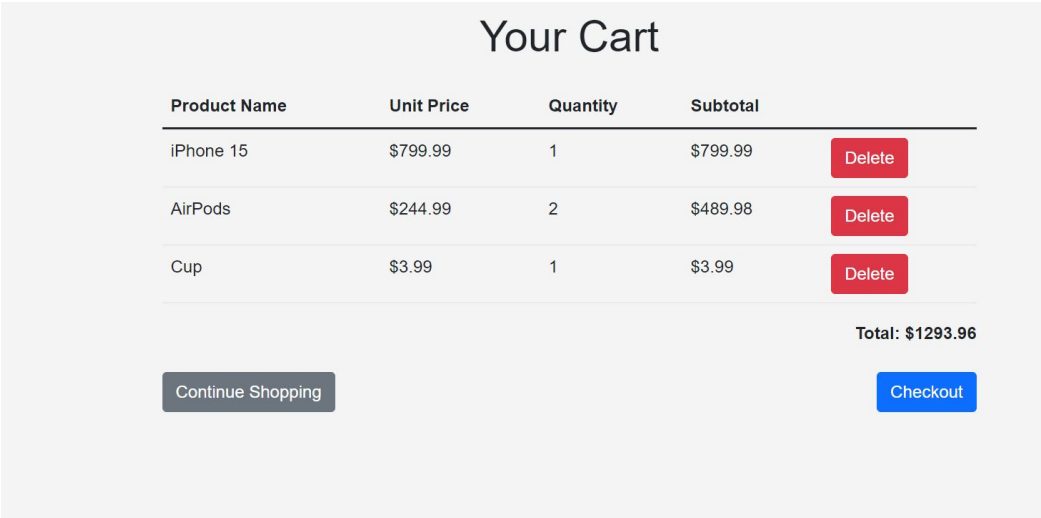
- Searching/sorting
  - Price: low to high/ high to low
  - Stock

- Add to cart
  - Quantity > 0 and Quantity <= Stock



View Cart

- Delete items
- Total cost is a calculated field
  - Σ Unit price * quantity
- Order status = cart
- Checkout pressed ->
  - Order status = completed
    - Runs check on quantity <= stock



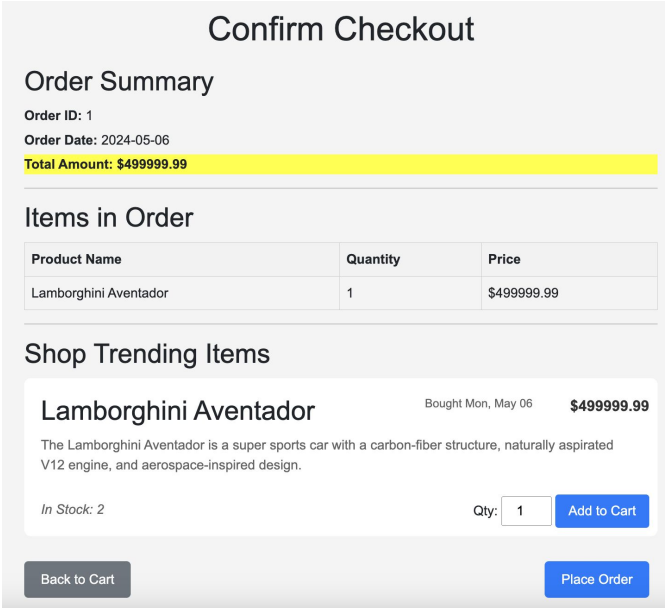
Orders

-Lists items of completed status

-Trending items

-Users buy more

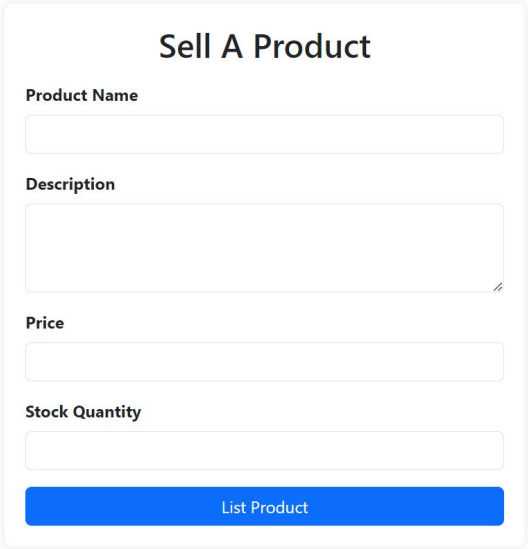
-Stock check valid



Error: Not enough stock for 6 units of Lamborghini Aventador. There are only 2 units available.

Sell items

- Adds to the product table
  - Can be used to update stock if item already exists



Scalability

-Stored Procedures for many orders

-Errors handled by order conditions aforementioned

-Stock check

-Can handle many users

```
CREATE OR REPLACE FUNCTION add_customers()
RETURNS void AS $$
BEGIN
    FOR i IN 1..100 LOOP
        INSERT INTO Customers (first_name, last_name, email, phone_number, address, city, state, zip_code)
        VALUES (
            'FirstName' || i,
            'LastName' || i,
            'user' || i || '@example.com',
            '1234567890',
            '123 Main St',
            'Anytown',
            'CA',
            '90210'
        );
    END LOOP;
END;
$$ LANGUAGE plpgsql;
```

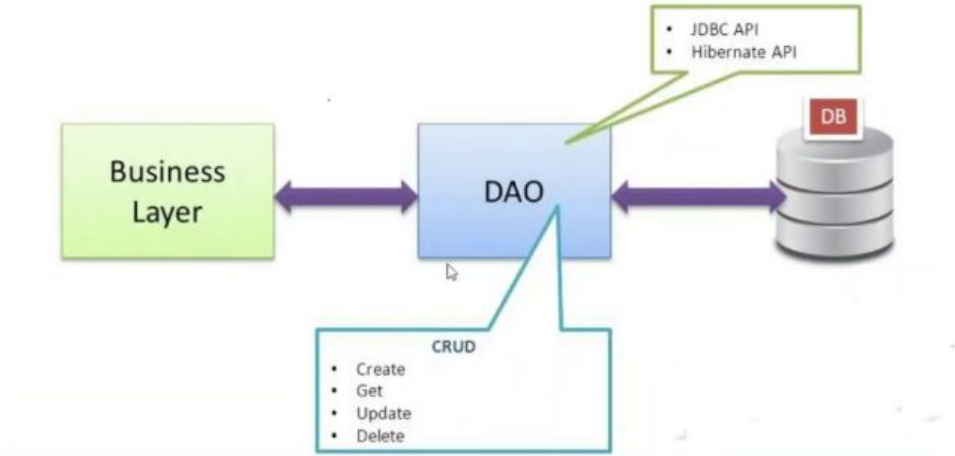
```
CREATE OR REPLACE FUNCTION add_orders()
RETURNS void AS $$
DECLARE
    selected_customer_id INT;
    selected_product_id INT;
BEGIN
    SELECT product_id INTO selected_product_id FROM Products LIMIT 1;

    FOR i IN 1..100 LOOP
        SELECT customer_id INTO selected_customer_id FROM Customers ORDER BY RANDOM() LIMIT 1;

        INSERT INTO Orders (customer_id, order_date, status)
        VALUES (
            selected_customer_id,
            CURRENT_DATE,
            'cart'
        );

        INSERT INTO Order_Items (order_id, product_id, quantity)
        VALUES (
            currval('orders_order_id_seq'),
            selected_product_id,
            1
        );
    END LOOP;
END;
$$ LANGUAGE plpgsql;

SELECT add_orders();
```

	customer_id [PK] integer	first_name character varying (50)	last_name character varying (50)	email character varying (100)	phone_number character varying (20)
4	4	FirstName1	LastName1	user1@example.com	1234567890
5	5	FirstName2	LastName2	user2@example.com	1234567890
6	6	FirstName3	LastName3	user3@example.com	1234567890
7	7	FirstName4	LastName4	user4@example.com	1234567890
8	8	FirstName5	LastName5	user5@example.com	1234567890
9	9	FirstName6	LastName6	user6@example.com	1234567890
10	10	FirstName7	LastName7	user7@example.com	1234567890
11	11	FirstName8	LastName8	user8@example.com	1234567890
12	12	FirstName9	LastName9	user9@example.com	1234567890
13	13	FirstName10	LastName10	user10@example.com	1234567890
14	14	FirstName11	LastName11	user11@example.com	1234567890
15	15	FirstName12	LastName12	user12@example.com	1234567890
16	16	FirstName13	LastName13	user13@example.com	1234567890
17	17	FirstName14	LastName14	user14@example.com	1234567890
18	18	FirstName15	LastName15	user15@example.com	1234567890
19	19	FirstName16	LastName16	user16@example.com	1234567890
20	20	FirstName17	LastName17	user17@example.com	1234567890
21	21	FirstName18	LastName18	user18@example.com	1234567890
22	22	FirstName19	LastName19	user19@example.com	1234567890
23	23	FirstName20	LastName20	user20@example.com	1234567890
24	24	FirstName21	LastName21	user21@example.com	1234567890
25	25	FirstName22	LastName22	user22@example.com	1234567890

	order_id [PK] integer	customer_id integer	order_date date
19	19	78	2024-05-06
20	20	75	2024-05-06
21	21	20	2024-05-06
22	22	84	2024-05-06
23	23	67	2024-05-06
24	24	55	2024-05-06
25	25	33	2024-05-06
26	26	38	2024-05-06
27	27	41	2024-05-06
28	28	19	2024-05-06
29	29	18	2024-05-06
30	30	62	2024-05-06
31	31	51	2024-05-06
32	32	49	2024-05-06
33	33	6	2024-05-06
34	34	44	2024-05-06
35	35	8	2024-05-06
36	36	71	2024-05-06
37	37	65	2024-05-06
38	38	91	2024-05-06
39	39	67	2024-05-06

Next Steps

- Data Access Object Structure + ORM
  - More custom queries
  - Stored Procedures
    - Integrity
- Returns
- Marketplace ecosystem
  - Profiles
  - Authorized sellers



Demo Agenda notes

1. Registration & Login
2. Store page
    a. Search & Sort
    b. Add to Cart (various quantities)
3. View Cart
    a. Remove items from cart
    b. Checkout
    c. Trending Items & purchase dates
    d. Order summary
4. View Orders
    a. Both completed and in progress orders
5. **Demo that product stock decreases on successful order**
6. **Demo that user cannot purchase more than the amount of product in stock**
7. Sell Items page
    a. List items for sale & show them populated in Store page