ART BUY AND SELL WEBSITE

Assignment Report

Omprakash Nain B22AI062
Yashraj Chaturvedi B22AI059
Anushk Gupta B22AI007

September 1, 2024

Abstract

This report documents the development process and implementation details of a basic buy-and-sell platform for art. The platform includes features such as artist profiles, artwork listings, search functionality, shopping cart and checkout, order management, and user authentication. This report also details the database design, indexing, and other backend operations involved in the system.

GitHub Repository

`https://github.com/introspective321/Data_Engineering_E-Commerce`

Contents

1 Introduction

This section provides an overview of the ART BUY AND SELL website, including the project objectives and the technologies used. The platform was developed to allow artists to showcase and sell their artworks online while providing buyers with an intuitive and secure purchasing experience.

2 ER Diagram

2.1 ER Diagram Description

Include the ER diagram and provide a detailed explanation of the entities, attributes, and relationships.

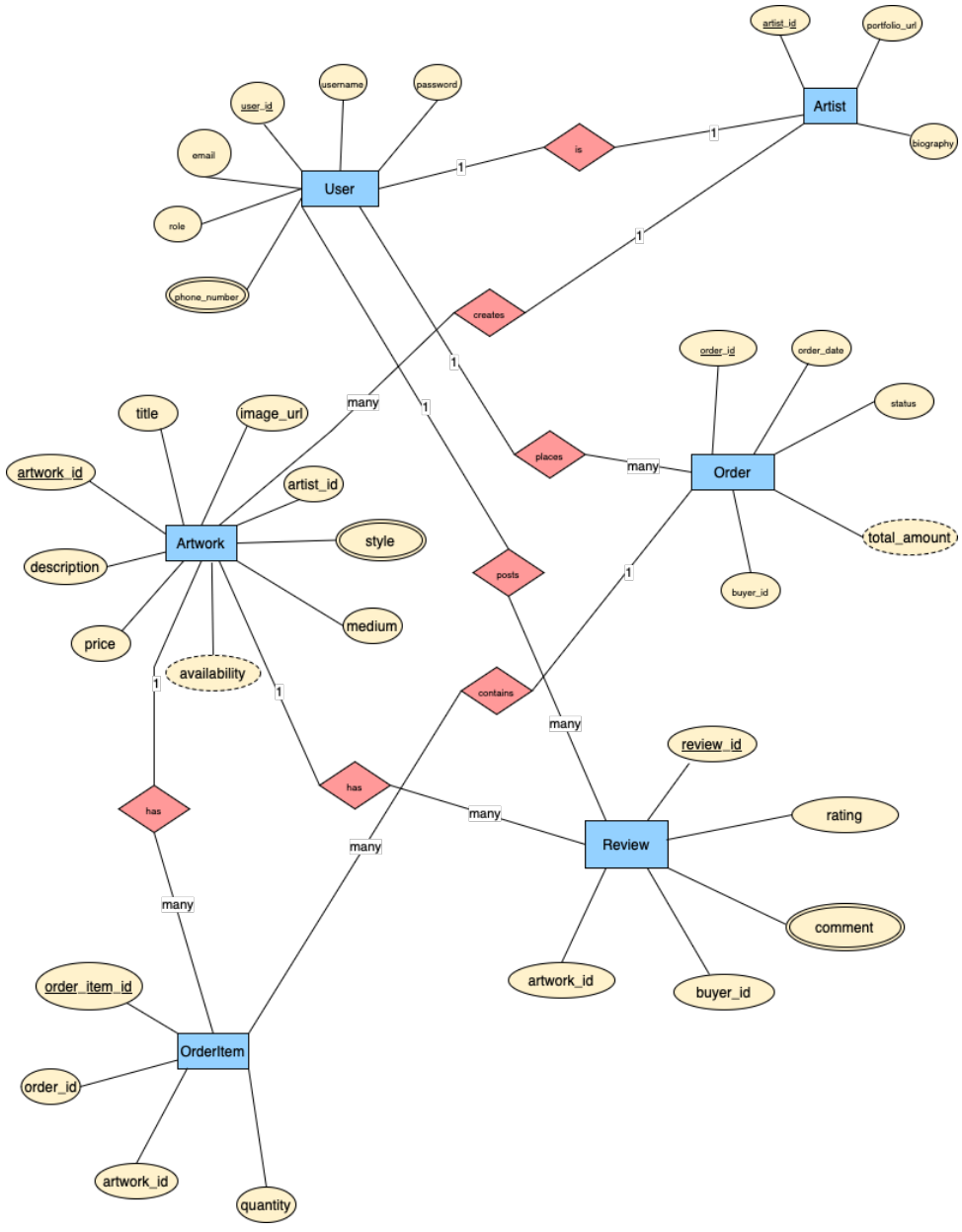

Figure 1: Entity-Relationship Diagram for ART BUY AND SELL website

3 Table Creation and Data Insertion

3.1 Tables Design

The database for the ART BUY AND SELL platform consists of the following tables:

- **artist**
- **artwork**
- **order**
- **orderitem**
- **review**
- **user**

Refer to the table 1 for detailed structure of the tables.

3.2 Data Insertion Using Python

Please refer to referenced Github repository for the same.

4 Normalization

4.1 2NF and Beyond

Based on the provided tables and requirements, the following is a summary of the normalization status and the decisions made:

4.1.1 Current Normal Form

The tables are currently normalized up to the Second Normal Form (2NF). Each table in the database satisfies the following criteria:

- Each table has a single primary key. - All non-key attributes are fully dependent on the entire primary key. - There are no partial dependencies on a subset of a composite key.

4.1.2 Decision on Further Normalization

Normalization to Third Normal Form (3NF): The ‘User‘ table is already in 3NF. There are no transitive dependencies as all attributes depend solely on `user_id`.

The ‘Artist‘ table is also in 3NF. The attributes `biography` and `portfolio_url` depend solely on `artist_id`, which is a foreign key from the ‘User‘ table.

The ‘Artwork‘ table meets the requirements of 3NF. All attributes are directly dependent on `artwork_id`, with no transitive dependencies.

The ‘Order‘ table is in 3NF, with all attributes depending directly on `order_id`.

The ‘OrderItem‘ table is in 3NF as well. The composite keys `order_id` and `artwork_id` are sufficient to ensure that all attributes depend on this composite key.

The ‘Review‘ table is in 3NF, with all attributes directly dependent on `review_id`.

Table Name	Columns	Description
artist	artist_id (INT, PK) biography (TEXT) portfolio_url (VARCHAR255)	Stores information ab
artwork	artwork_id (INT, PK) artist_id (INT, FK) title (VARCHAR) description (TEXT) image_url (VARCHAR) price (DECIMAL) availability (ENUM, available/sold) medium (VARCHAR255) style (VARCHAR)	Details of artworks av
order	order_id (INT, PK) buyer_id (INT, FK) total_amount (DECIMAL) status (ENUM ('PENDING', 'SHIPPED', 'DELIVERED')) order_date (DATE)	Information about ord
orderitem	order_item_id (INT, PK) order_id (INT, FK) artwork_id (INT, FK) quantity (INT)	Details of items withi
review	review_id (INT, PK) artwork_id (INT, FK) buyer_id (INT, FK) rating (INT) comment (TEXT)	Reviews and ratings f
user	user_id (INT, PK) username (VARCHAR) password (VARCHAR) email (VARCHAR) role (ENUM ('Artist', 'Buyer'))	Information about pla

Table 1: Summary of Database Tables

Normalization to Boyce-Codd Normal Form (BCNF): - All tables are also in BCNF, as each determinant is a candidate key. The primary keys in each table uniquely identify rows, and there are no partial dependencies.

4.1.3 Trade-Offs and Recommendations

Simplicity and Performance: Maintaining the schema in 2NF offers a simpler structure with fewer tables, which can be advantageous for performance and easier maintenance in smaller systems.

Data Integrity: Normalization to 3NF or BCNF enhances data integrity by eliminating transitive dependencies and ensuring that every determinant is a candidate key.

Complexity: Further normalization to 3NF and BCNF may introduce additional tables and complexity. For systems with well-structured data relationships, this trade-off may be justified to achieve higher data integrity.

In summary, the tables are effectively in 2NF, and it has been decided to retain this level of normalization unless future requirements dictate the need for further normalization.

5 Hashing and Indexing

5.1 Note on Hashing and Indexing Schemes in MySQL

MySQL uses various hashing and indexing techniques to optimize query performance and manage data efficiently. Below is an overview of the key methods:

5.1.1 Hashing

- **Hash Function:** Maps a key to a hash value, which is then used to directly locate the data record.
- **Hash Indexes:**
  - **Use Case:** Primarily used in MEMORY tables for fast equality lookups (‘=‘).
  - **Efficiency:** Ideal for exact matches due to direct data access.
  - **Limitations:** Not suitable for range queries (‘¡‘, ‘¿‘, ‘BETWEEN‘) as hash values do not maintain order.

5.1.2 B-Tree Indexes

- **Structure:** Balanced tree where nodes contain keys and pointers, ensuring logarithmic search, insert, and delete operations.
- **Usage:** Default indexing method in MySQL (InnoDB, MyISAM).
- **Supports:**
  - **Equality Queries:** Efficiently handles exact matches.
  - **Range Queries:** Maintains key order, ideal for ‘BETWEEN‘, ‘¿=‘, ‘¡=‘ queries.
  - **Prefix Searches:** Works well with ‘LIKE‘ clauses for prefix matching.
- **Considerations:**
  - **Memory Usage:** Requires significant memory for large datasets.
  - **Maintenance Overhead:** Rebalancing during frequent inserts/deletes can impact performance.

5.1.3 Full-Text Indexes

- **Purpose:** Optimizes text search within large text fields.
- **Mechanism:**

  - **Tokenization:** Splits text into individual words (tokens) for indexing.
  - **Inverted Index:** Maps tokens to their locations within the text.
- **Search Efficiency:**
  - **Keywords:** Fast retrieval for keyword-based searches using 'MATCH ... AGAINST'.
  - **Relevance Ranking:** Can rank results based on relevance.
- **Trade-offs:** Increased storage requirements and slower updates due to index complexity.

5.1.4 Spatial Indexes

- **Use Case:** Optimizes queries on spatial data types (e.g., points, polygons).
- **Structure:**
  - **R-Tree Indexes:** Group nearby objects using Minimum Bounding Rectangles (MBRs).
  - **Spatial Operations:** Efficiently supports spatial queries (e.g., containment, intersection).
- **Applications:** Essential for GIS and location-based services, reducing search space for spatial queries.

5.2 Custom Hash Function

To optimize the retrieval of data from the 'Artwork' table, a custom hash function was implemented. This function is specifically designed to efficiently map keys, such as artwork identifiers, to buckets within a hash table. The hash function considers certain common alphabets and adjusts the hash calculation accordingly.

5.2.1 Mathematical Representation of the Hash Function

The custom hash function can be mathematically represented as follows:

$$H(key) = \left(\sum_{i=1}^{n} f(c_i) \right) \mod m$$

Where:

- $H(key)$ is the hash value for the given key.
- n is the length of the key.
- c_i is the i^{th} character of the key.
- m is the number of buckets in the hash table.
- $f(c_i)$ is the function defined as:

$$f(c_i) = \begin{cases} (H_{prev} \times 31 + \text{ord}(c_i)) \mod m & \text{if } c_i \in \{B, A, I\} \\ (H_{prev} \times 37 + \text{ord}(c_i)) \mod m & \text{if } c_i \notin \{B, A, I\} \end{cases}$$

- H_{prev} is the cumulative hash value from processing previous characters in the key.
- ord(c_i) is the ASCII value of the character c_i.

5.2.2 Design of the Custom Hash Function

The custom hash function is designed with the following key considerations:

- **Common Alphabets:** The function prioritizes the alphabets 'B', 'A', and 'I', which are presumed to be common in the keys (e.g., artwork IDs). These letters are factored into the hash calculation to ensure a more even distribution of keys across the buckets.
- **Hash Value Calculation:**
  - For each character in the key, if the character is one of the common alphabets ('B', 'A', 'I'), the hash value is computed using a multiplication factor of 31 (a prime number often used in hash functions to reduce collisions) and then taking the modulus with the number of buckets.
  - If the character is not one of the common alphabets, a different multiplication factor of 37 is used, followed by a modulus operation with the number of buckets.
- **Purpose of Modulus Operation:** The modulus operation ensures that the resulting hash value maps to a valid index within the hash table, corresponding to one of the available buckets.

5.2.3 Implementation of the Custom Hash Function

The custom hash function is implemented in the 'HashTable' class as follows:

```
def custom_hash_function(self, key):
    common_alphabets = "BAI"
    hash_value = 0
    for char in key:
        if char in common_alphabets:
            hash_value = (hash_value * 31 + ord(char)) % self.num_buckets
        else:
            hash_value = (hash_value * 37 + ord(char)) % self.num_buckets
    return hash_value
```

This function processes each character of the key and applies the appropriate hashing rule based on whether the character is one of the common alphabets or not. The cumulative hash value is used to determine the final bucket index.

5.2.4 Usage Example

The custom hash function is used within the 'HashTable' class to both insert and retrieve data from the 'Artwork' table. Below is an example of how the function maps keys to buckets:

```
keys = ['Artwork1', 'Artwork15', 'Artwork3']
```

```
for key in keys:
    bucket = hash_table.custom_hash_function(key)
    print(f"Key: {key} is mapped to bucket: {bucket}")
```

5.3 Clustering & Secondary Indexing

When comparing clustering and secondary indexing schemes, the key factors include storage requirements and execution time for queries:

5.3.1 Clustering Indexing

- Clustering indexing organizes data rows physically based on the clustering key, with only one clustering index allowed per table.
- **Storage:** Efficient with less overhead, but can increase with fragmentation and frequent updates.
- **Execution Time:** Ideal for range queries and sorting by the clustering key, but can slow down insert, update, and delete operations.

5.3.2 Secondary Indexing

- Secondary indexes provide quick data access without altering the physical order of rows, allowing multiple indexes per table.
- **Storage:** Requires additional storage for each index, potentially leading to higher overhead.
- **Execution Time:** Improves query performance for non-clustering keys, but can slow down write operations due to index maintenance.

5.4 Analytical Comparison of Indexing Schemes

The performance of clustering and secondary indexing schemes was evaluated based on their execution times for queries using different indexes. Below is the analysis:

1. **Clustering Index (‘artwork_id‘):**
    - **Execution Time:** 0.001000 seconds
    - **Explanation:** This query utilized the primary key index, which is clustered by default. As the query is an exact match on a unique identifier, the retrieval is expected to be extremely fast.
2. **Secondary Index (‘style‘):**
    - **Execution Time:** 0.001004 seconds
    - **Explanation:** The query employed a secondary index on the ‘style‘ column. The execution time is nearly identical to that of the clustering index. This demonstrates that secondary indexes can perform almost as efficiently as clustering indexes for queries on selective columns.
3. **Secondary Index (‘medium‘):**

- **Execution Time:** 0.000000 seconds
- **Explanation:** The execution time reported as zero could be due to the granularity of the timing measurement or caching effects. In practice, secondary indexes like 'medium' may take slightly longer to execute, especially if the indexed column is less selective and the dataset is larger.

This analysis shows that clustering indexes offer predictable performance advantages for primary key lookups, while secondary indexes can also provide efficient query performance, particularly when the indexed columns are highly selective.

Aspect	Clustering Indexing	Secondary Indexing
Physical Storage	Data is physically ordered based on clustering key.	Data is not physically reordered; indexes are stored separately.
Storage Overhead	Generally lower, but can increase with fragmentation.	Higher, as each index requires additional storage.
Query Performance	Efficient for range queries and sorting based on the clustering key.	Efficient for exact match queries and retrieval based on indexed columns.
Update Cost	Can be high due to physical reordering of data.	Can be high due to maintenance of multiple indexes.
Flexibility	Limited to a single clustering index.	Multiple indexes can be created for different columns.

Table 2: Comparison of Clustering and Secondary Indexing

6 SQL Queries

6.1 Inclusion of 5 New Contemporary Artists

As part of the project, five new contemporary artists were added to the 'art$_platform$'$database.Thistaski$

6.1.1 SQL Script for Adding New Artists

The following SQL scripts were executed to include the new artists in the database:

Step 1: Inserting New Users as Artists

```
INSERT INTO User (username, password, email, role) VALUES
('contemporary_artist1', 'password5', 'contemporary1@example.com', 'Artist'),
('contemporary_artist2', 'password6', 'contemporary2@example.com', 'Artist'),
('contemporary_artist3', 'password7', 'contemporary3@example.com', 'Artist'),
('contemporary_artist4', 'password8', 'contemporary4@example.com', 'Artist'),
('contemporary_artist5', 'password9', 'contemporary5@example.com', 'Artist');
```

Step 2: Adding Artist Details

Assuming the 'artist$_id$'$valuesstartfrom5, thefollowingscriptwasusedtoaddthespecificdetailsforeacha$

```
INSERT INTO Artist (artist_id, biography, portfolio_url) VALUES
(5, 'Biography of contemporary artist 1', 'http://portfolio5.com'),
(6, 'Biography of contemporary artist 2', 'http://portfolio6.com'),
```

```
(7, 'Biography of contemporary artist 3', 'http://portfolio7.com'),
(8, 'Biography of contemporary artist 4', 'http://portfolio8.com'),
(9, 'Biography of contemporary artist 5', 'http://portfolio9.com');
```

6.1.2 Execution and Outcome

The above SQL scripts were executed successfully, resulting in the addition of five new contemporary artists to the database. Each artist's user credentials and profile details were accurately inserted, ensuring they are now part of the 'art_platform' system.

6.2 Artwork Listings in August 2024

As part of the project, a report was generated to list all artwork listings made in August 2024. The following SQL query was used to retrieve the relevant data from the 'Artwork' table:

SQL Query to Retrieve Artwork Listings

```
SELECT artwork_id, title, description, image_url, price, availability, medium, style,
FROM Artwork
WHERE artwork_id IN (
    SELECT artwork_id
    FROM OrderItem
    JOIN 'Order' ON OrderItem.order_id = 'Order'.order_id
    WHERE 'Order'.order_date BETWEEN '2024-08-01' AND '2024-08-31'
);
```

This query retrieves all artwork details for listings made between August 1, 2024, and August 31, 2024. It does so by:

- Selecting records from the 'Artwork' table that match 'artwork_id's from the 'OrderItem' table.
- Joining 'OrderItem' with the 'Order' table to filter records based on the 'order_date'.
- Filtering the results to only include orders placed within the specified date range.

The results include the 'artwork_id', 'title', 'description', 'image_url', 'price', 'availability', 'medium', 'style', and 'artist_id' for each artwork listed during August 2024. This data provides a comprehensive view of the artwork transactions within the specified period.

6.3 Remove Purchases after August 15, 2024

To remove all artwork purchases made after 7 PM on August 15, 2024, the following SQL query was used:

SQL Query to Remove Purchases

```
DELETE FROM OrderItem
WHERE order_id IN (
    SELECT order_id
    FROM 'Order'
    WHERE order_date > '2024-08-15 19:00:00'
);
```

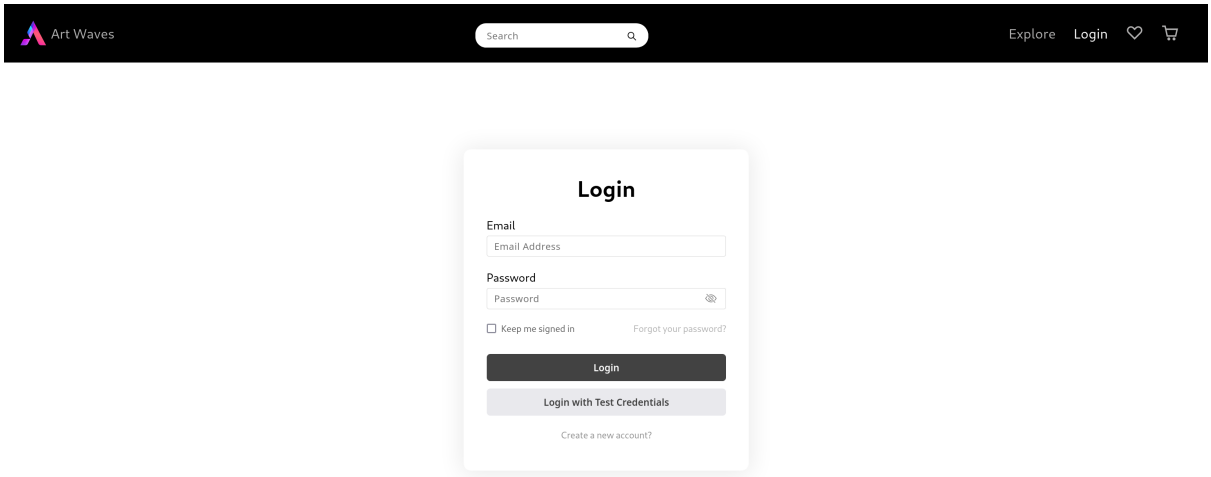

Figure 2: Login Screen

This query deletes all records in the ‘OrderItem‘ table where the ‘order_id‘ corresponds to orders placed after 7 PM on August 15, 2024. The process involves:

- Selecting ‘order_id‘s from the ‘Order‘ table where ‘order_date‘ is later than ’2024-08-15 19:00:00’.
- Deleting the associated entries in the ‘OrderItem‘ table using these ‘order_id‘s.

This operation ensures that any purchases made after the specified date and time are removed from the database, maintaining the integrity of the transaction records.

7 Web Interface

The Buy-and-Sell Art Platform features a user-friendly and aesthetically modern web interface. The homepage directs authenticated users to their respective dashboards or the artwork browsing page. The login and registration screens are clean and straightforward, with clearly labeled input fields and validation feedback. Artists have a dedicated dashboard to manage their artworks, and a responsive grid layout showcases available artworks for buyers. The design employs a consistent color scheme and intuitive navigation for a seamless user experience. Refer figures 3, 4 and 5.

8 Conclusion

We have successfully integrated essential e-commerce functionalities, providing a seamless experience for both artists and buyers. The project highlights effective database design, including normalization, indexing, and hashing techniques, ensuring optimal performance and data integrity. The system’s user-friendly interface and robust backend operations meet the project objectives, creating a solid foundation for future enhancements and scalability. Overall, the platform exemplifies a well-executed solution for an art-focused online marketplace.

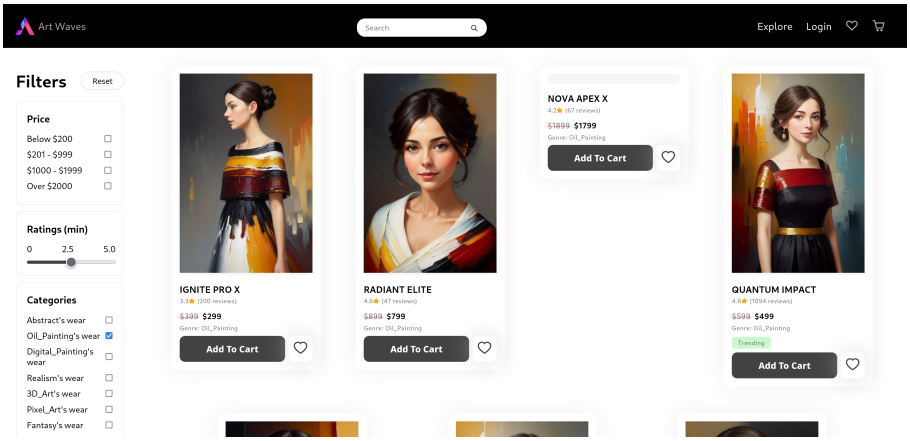

Figure 3: Browse Artworks

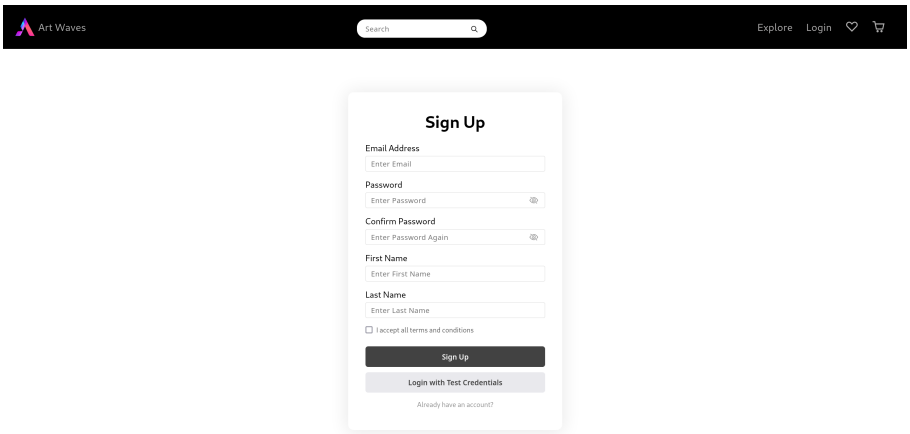

Figure 4: Sign Up Screen

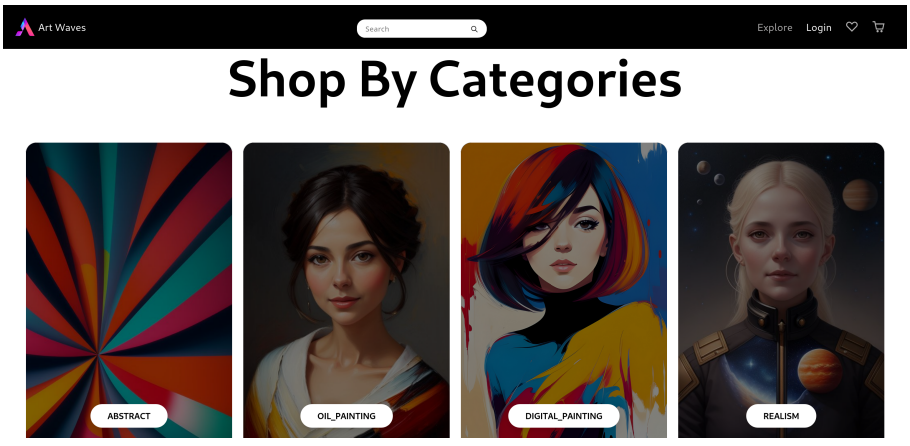

Figure 5: Landing Page