Final Report

CraftValley: Online Handcrafted Goods Marketplace Platform

GitHub link

Group 9

Berkay Gündüz, 22103402

Ege Şirvan, 22102289

Eren Aslan, 22102329

Kemal Sarper Şahin, 22103801

Öykü Elis Türegün, 21902976

Department of Computer Engineering

CS353 Database Systems

Özgür Ulusoy

Table of Contents

Table of Contents 1
1. Project Description 2
2. Individual Contributions 3
3. Final E/R Model 5
4. Final List of Tables, Views, Procedures 5
4.1 User 5
4.2 Small_Business 6
4.3 Customer 7
4.4 Admin 7
4.5 Product 8
4.6 Balance_Record 8
4.7 Recipient 9
4.8 Material 9
4.9 Main_Category 10
4.10 Sub_Category 10
4.11 In_Category 11
4.12 Has_Reported 12
4.13 Ban 12
4.14 System_Report 13
4.15 Create_Report 14
4.16 Rate 14
4.17 Wish 15
4.18 Add_To_Shopping_Cart 15
4.19 Business_Has_Record 16
4.20 Customer_Has_Record 17
4.21 Transaction 17
4.22 Add_Product 18
4.23 Made_By 19
4.24 Is_For 19
4.25 CartAdder 20
4.26 Rate Product 21
4.27 ProductPrinter 21
This procedure retrieves a paginated list of products with details such as title, description, price, available amount, average rating, number of ratings, images, business name, and wishlist status. It also supports pagination. 21
4.28 ProductFilter 22
4.29 CategoryProductFilter 23
4.30 BusinessProducts 25
4.31 ReturnProduct 25

4.32 UserTransactions 26
5. Implementation Details **27**
Frontend Implementation 27
Backend Implementation 28
6. User Manual **28**
6.1 Build Instructions 28
6.2 Login / Sign Up 29
6.3 Customer Homepage 30
6.4 Wishlist 31
6.5 Shopping Cart 31
6.6 Customer Profile 32
6.7 Edit Customer Profile 33
6.8 Past Purchases 33
6.9 Small Business Profile 33
6.10 Create Product 34
6.14 Admin Page 36

1. Project Description

CraftValley operates as an online marketplace for buying and selling handcrafted goods. Small businesses can create profiles, list products, and manage inventory. They can showcase their work with images, write detailed descriptions, and set prices. Additionally, they gain valuable insights into customer preferences through ratings, allowing them to refine their offerings and build trust within the community.

Customers enjoy a user-friendly experience and discover products. Advanced search filters based on various criteria help them find the exact items they're looking for, while they can track their favorite products. The platform streamlines the buying process with wish lists, shopping carts, and transaction options. Customers can also contribute to the community by leaving honest feedback, fostering trust and transparency within the marketplace.

Administrators are equipped with comprehensive reporting tools that provide valuable insights into popular products, user trends, and overall platform performance. They can manage user accounts to ensure platform integrity and security, taking necessary actions to maintain a safe and enjoyable environment for all users.

CraftValley's robust database architecture guarantees efficient data retrieval and manipulation. Users can leverage advanced search filters, and the system seamlessly handles user ratings, transactions, and image uploads. By empowering all user groups and fostering a collaborative community, CraftValley aims to be a destination for both artisans and customers loving the beauty and value of handcrafted creativity.

2. Individual Contributions

2.1 Project Proposal

2.1.1 **Berkay Gündüz**

Introduction
Project Description
E/R Design

2.1.2 **Öykü Elis Türegün**

Database Usage
E/R Design

2.1.3 **Eren Aslan**

Functional Requirements
Pseudo Requirements
E/R Design

2.1.4 **Ege Şirvan**

Non-functional Requirements
E/R Design

2.1.5 **Kemal Sarper Şahin**

Limitations
E/R Design

2.2 Project Design

2.2.1 **Berkay Gündüz**

E/R Design

2.2.2 **Öykü Elis Türegün**

Table Schemas
E/R Design

2.2.3 **Eren Aslan**

E/R Design

2.2.4 **Ege Şirvan**

E/R Design

2.2.5 **Kemal Sarper Şahin**

E/R Design

2.3 Project Final Report

2.3.1 **Berkay Gündüz**

2.3.2 **Öykü Elis Türegün**

Final Tables, Views, and Procedures
Project Description
User Manual of Small Business
Implementation Plan

2.3.3 **Eren Aslan**

E/R Design

2.3.4 **Ege Şirvan**

2.3.5 **Kemal Sarper Şahin**

User Manual of Build Instructions, Login/Signup, Customer-related pages, Admin page.

2.4 Project Implementation

2.4.1 **Berkay Gündüz**

SQL Table Definitions
Fullstack development:

2.4.2 **Öykü Elis Türegün**

SQL Table Definitions
Frontend-Backend of Small Business Operations:

– create_product
– list_products
– edit_product
– update_product_amount
– delete_product
– small_business_profile main: showing business details

General UI Design Fix

2.4.3 **Eren Aslan**

SQL Procedure Definitions (Product Filter, Rating, Wish) and View Definitions
Frontend-Backend of Customer Operations:

-mainPageUser
-transaction

Backend for Product related operations (Wish, Return, Rate, etc.)

2.4.4 **Ege Şirvan**

SQL Procedure Definitions
Frontend-Backend of Customer Operations:

-categoryPage
-navigation bar / userBase

Frontend of some Product/User related operations
General debug

2.4.5 **Kemal Sarper Şahin**

Frontend-Backend of Authorization

- Login/Register

Frontend-Backend of Customer Profile
- View current information
- edit profile information

A part of Small-Business backend
- edit profile information

3. Final E/R Model

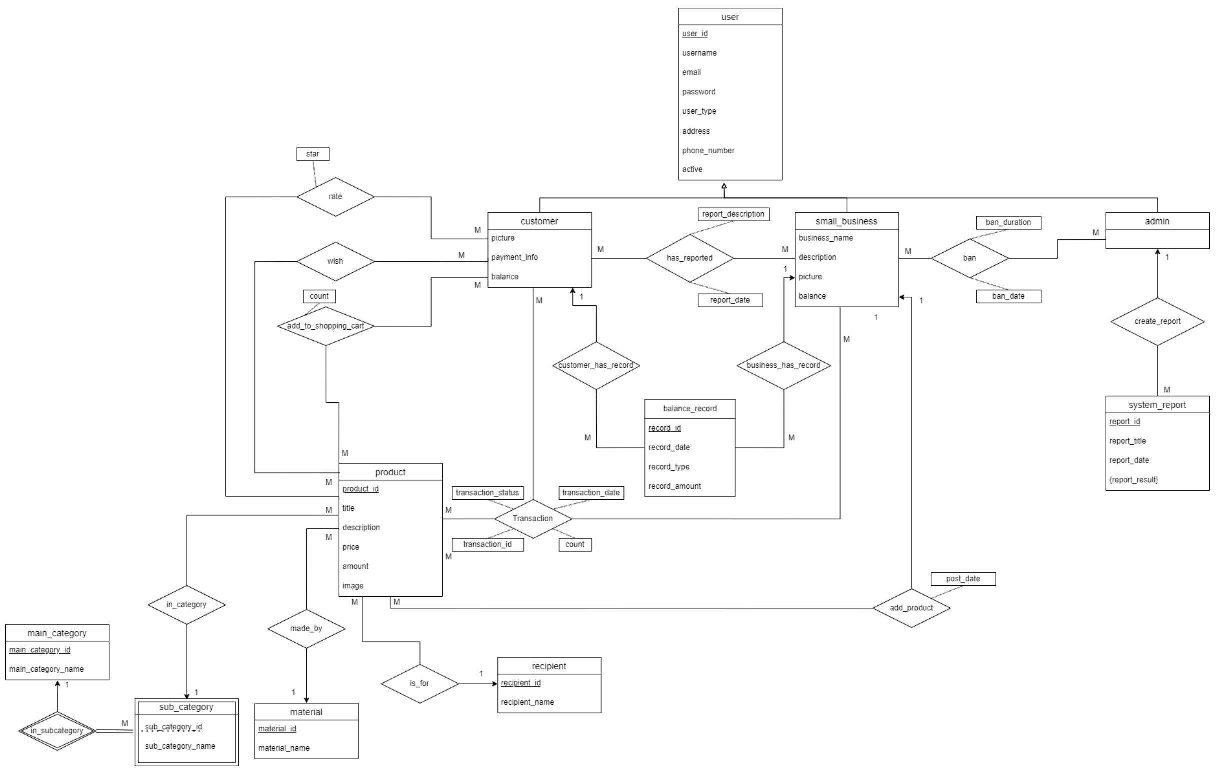


Figure 1

4. Final List of Tables, Views, Procedures

4.1 User

Relational Model:

```
User(user_id, user_name, email, password, user_type, address,
phone_number, active)
```

Functional Dependencies:

```
user_id -> user_name, email, password, user_type, address,
phone_number, active
email -> user_id
phone_number -> user_id
```

Candidate Keys:

```
{user_id}, {email}, {phone_number}
```

Primary Key:

```
user_id
```
Foreign Keys:
```
None
```
Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS User (
   user_id INT NOT NULL AUTO_INCREMENT,
   user_name VARCHAR(255) NOT NULL,
   email VARCHAR(255) NOT NULL,
   password VARCHAR(255) NOT NULL,
   user_type VARCHAR(255) NOT NULL,
   address VARCHAR(255),
   phone_number VARCHAR(20) NOT NULL,
   active INT NOT NULL CHECK (active IN (0, 1)),
   PRIMARY KEY (user_id),
   UNIQUE KEY (email),
   UNIQUE KEY (phone_number)
);
```

4.2 Small_Business

Relational Model:
```
Small_Business(user_id, business_name, title, description, picture,
balance)
```
Functional Dependencies:
```
user_id -> business_name, title, description, picture, balance
```
Candidate Keys:
```
{user_id}
```
Primary Key:
```
user_id
```
Foreign Keys:
```
user_id -> User(user_id)
```
Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Small_Business (
   user_id          INT NOT NULL,
   business_name    VARCHAR(255) NOT NULL,
   title        VARCHAR(255) NOT NULL,
   description  VARCHAR(255),
   picture          LONGBLOB,
   balance          DECIMAL(10,2),
   PRIMARY KEY(user_id),
   FOREIGN KEY(user_id) REFERENCES User(user_id) ON DELETE CASCADE
);
```

4.3 Customer

Relational Model:

```
Customer(user_id, picture, payment_info, balance)
```

Functional Dependencies:

```
user_id -> picture, payment_info, balance
```

Candidate Keys:

```
{user_id}
```

Primary Key:

```
user_id
```

Foreign Keys:

```
user_id -> User(user_id)
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Customer(
   user_id              INT NOT NULL,
   picture              LONGBLOB,
   payment_info         VARCHAR(255) NOT NULL,
   balance              DECIMAL(10,2) NOT NULL,
   PRIMARY KEY(user_id),
   FOREIGN KEY(user_id) REFERENCES User(user_id) ON DELETE CASCADE
);
```

4.4 Admin

Relational Model:

```
Admin(user_id)
```

Functional Dependencies:

```
user_id -> (No other attributes)
```

Candidate Keys:

```
{user_id}
```

Primary Key:

```
user_id
```

Foreign Keys:

```
user_id -> User(user_id)
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Admin (
   user_id INT NOT NULL,
   PRIMARY KEY (user_id),
   FOREIGN KEY (user_id) REFERENCES User(user_id) ON DELETE CASCADE
```

);

4.5 Product

Relational Model:

`Product(product_id, title, description, price, amount, images)`

Functional Dependencies:

```
product_id -> title, description, price, amount, images
```

Candidate Keys:

```
{product_id}
```

Primary Key:

```
product_id
```

Foreign Keys:

```
None
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Product(
   product_id   INT NOT NULL AUTO_INCREMENT,
   title        VARCHAR(255) NOT NULL,
   description  VARCHAR(255),
   price               DECIMAL(10,2) NOT NULL,
   amount              INT NOT NULL,
   images              LONGBLOB,
   PRIMARY KEY(product_id)
);
```

4.6 Balance_Record

Relational Model:

```
Balance_Record(record_id, record_date, record_type, record_amount)
```

Functional Dependencies:

```
record_id -> record_date, record_type, record_amount
```

Candidate Keys:

```
{record_id}
```

Primary Key:

```
record_id
```

Foreign Keys:

```
None
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Balance_Record(
    record_id              INT NOT NULL AUTO_INCREMENT,
    record_date            DATE NOT NULL,
    record_type            VARCHAR(255) NOT NULL,
    record_amount          DECIMAL(10,2) NOT NULL,
    PRIMARY KEY(record_id)
);
```

4.7 Recipient

Relational Model:

    Recipient(recipient_id, recipient_name)

Functional Dependencies:

    recipient_id -> recipient_name

Candidate Keys:

    {recipient_id}, {recipient_name}

Primary Key:

    recipient_id

Foreign Keys:

    None

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Recipient(
    recipient_id           INT NOT NULL AUTO_INCREMENT,
    recipient_name                  VARCHAR(255) NOT NULL,
    PRIMARY KEY(recipient_id),
        UNIQUE KEY(recipient_name)
);
```

4.8 Material

Relational Model:

    Material(material_id, material_name)

Functional Dependencies:

    material_id -> material_name

Candidate Keys:

    {material_id},{material_name}

Primary Key:

    material_id

Foreign Keys:

    None

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Material(
   material_id  INT NOT NULL AUTO_INCREMENT,
   material_name       VARCHAR(255) NOT NULL,
   PRIMARY KEY(material_id),
   UNIQUE KEY(material_name)
);
```

4.9 Main_Category

Relational Model:

`Main_Category(main_category_id, main_category_name)`

Functional Dependencies:

`main_category_id -> main_category_name`

Candidate Keys:

`{main_category_id}, {main_category_name}`

Primary Key:

`main_category_id`

Foreign Keys:

`None`

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Main_Category(
   main_category_id   INT NOT NULL AUTO_INCREMENT,
   main_category_name        VARCHAR(255) NOT NULL,
   PRIMARY KEY(main_category_id)
);
```

4.10 Sub_Category

Relational Model:

`Sub_Category(sub_category_id, main_category_id, sub_category_name)`

Functional Dependencies:

`sub_category_id, main_category_id -> subcategory_name`

Candidate Keys:

`{sub_category_id, main_category_id}, {sub_category_name, main_category_name}`

Primary Key:

`(sub_category_id, main_category_id)`

Foreign Keys:

```
main_category_id -> Main_Category(main_category_id)
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Sub_Category(
   sub_category_id     INT NOT NULL AUTO_INCREMENT,
   main_category_id    INT NOT NULL,
   sub_category_name          VARCHAR(255) NOT NULL,
   PRIMARY KEY(sub_category_id, main_category_id),
   FOREIGN KEY(main_category_id) REFERENCES
Main_Category(main_category_id) ON DELETE CASCADE
);
```

4.11 In_Category

Relational Model:

```
In_Category(sub_category_id, main_category_id, product_id)
```

Functional Dependencies:

```
sub_category_id, main_category_id, product_id → {no other attributes}
```

Candidate Keys:

```
{sub_category_id, main_category_id, product_id}
```

Primary Key:

```
(sub_category_id, main_category_id, product_id)
```

Foreign Keys:

```
main_category_id -> Sub_Category(main_category_id) ON DELETE CASCADE
sub_category_id -> Sub_Category(sub_category_id) ON DELETE CASCADE
product_id -> Product(product_id) ON DELETE CASCADE
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS In_Category (
   sub_category_id INT NOT NULL,
   main_category_id INT NOT NULL,
   product_id INT NOT NULL,
   PRIMARY KEY (sub_category_id, main_category_id, product_id),
   FOREIGN KEY (main_category_id) REFERENCES
Main_Category(main_category_id) ON DELETE CASCADE,
   FOREIGN KEY (sub_category_id) REFERENCES
Sub_Category(sub_category_id) ON DELETE CASCADE,
   FOREIGN KEY (product_id) REFERENCES Product(product_id) ON DELETE
CASCADE
);
```

4.12 Has_Reported

Relational Model:

```
Has_Reported(customer_id, small_business_id,
             report_description, report_date)
```

Functional Dependencies:

```
customer_id, small_business_id -> report_description, report_date
```

Candidate Keys:

```
{customer_id, small_business_id}
```

Primary Key:

```
(customer_id, small_business_id)
```

Foreign Keys

```
customer_id -> Customer(user_id)
small_business_id -> Small_Business(user_id)
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Has_Reported (
   customer_id INT NOT NULL,
   small_business_id INT NOT NULL,
   report_description VARCHAR(255),
   report_date DATE,
   PRIMARY KEY (customer_id, small_business_id),
   FOREIGN KEY (customer_id) REFERENCES Customer(user_id) ON DELETE
CASCADE,
   FOREIGN KEY (small_business_id) REFERENCES Small_Business(user_id) ON
DELETE CASCADE
);
```

4.13 Ban

Relational Model:

```
Ban(admin_id, small_business_id, ban_duration, ban_date)
```

Functional Dependencies:

```
admin_id, small_business_id -> ban_duration, ban_date
```

Candidate Keys:

```
{admin_id, small_business_id}
```

Primary Key:

```
(admin_id, small_business_id)
```

Foreign Keys:

```
admin_id -> Admin(user_id)
small_business_id -> Small_Business(user_id)
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Ban (
   admin_id INT NOT NULL,
   small_business_id INT NOT NULL,
   ban_duration VARCHAR(255),
   ban_date DATE,
   PRIMARY KEY (admin_id, small_business_id),
   FOREIGN KEY (admin_id) REFERENCES Admin(user_id) ON DELETE
CASCADE,
   FOREIGN KEY (small_business_id) REFERENCES Small_Business(user_id) ON
DELETE CASCADE
);
```

4.14 System_Report

Relational Model:

System_Report(<u>report_id</u>, report_title, report_date, report_results)

Functional Dependencies:

report_id → report_title, report_date, report_results

Candidate Keys:

{report_id}

Primary Key:

(report_id)

Foreign Keys:

None

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS System_Report(
   report_id           INT NOT NULL AUTO_INCREMENT,
   report_title        VARCHAR(255) NOT NULL,
   report_date         DATE NOT NULL,
   report_results              VARCHAR(255),
   PRIMARY KEY(report_id)
   );
```

4.15 Create_Report

Relational Model:

```
Create_Report(admin_id, report_id)
```

Functional Dependencies:

```
admin_id, report_id → {no other attributes}
```

Candidate Keys:

```
{admin_id, report_id}
```

Primary Key:

```
(admin_id, report_id)
```

Foreign Keys:

```
admin_id -> Admin(user_id)
report_id -> System_Report(report_id)
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Create_Report (
   admin_id INT NOT NULL,
   report_id INT NOT NULL,
   PRIMARY KEY (admin_id, report_id),
   FOREIGN KEY (admin_id) REFERENCES Admin(user_id) ON DELETE
CASCADE,
   FOREIGN KEY (report_id) REFERENCES System_Report(report_id) ON DELETE
CASCADE
);
```

4.16 Rate

Relational Model:

```
Rate(customer_id, product_id, star)
```

Functional Dependencies:

```
customer_id, product_id -> star
```

Candidate Keys:

```
{customer_id, product_id}
```

Primary Key:

```
(customer_id, product_id)
```

Foreign Keys:

```
customer_id -> Customer(user_id)
product_id -> Product(product_id)
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Rate (
   customer_id INT NOT NULL,
   product_id INT NOT NULL,
   star DECIMAL(2, 1) NOT NULL,
   PRIMARY KEY (customer_id, product_id),
   FOREIGN KEY (customer_id) REFERENCES Customer(user_id) ON DELETE
CASCADE,
   FOREIGN KEY (product_id) REFERENCES Product(product_id) ON DELETE
CASCADE
);
```

4.17 Wish

Relational Model:

```
Wish(customer_id, product_id)
```

Functional Dependencies:

```
customer_id, product_id -> (No other attributes)
```

Candidate Keys:

```
{customer_id, product_id}
```

Primary Key:

```
(customer_id, product_id)
```

Foreign Keys:

```
customer_id -> Customer(user_id) ON DELETE CASCADE
product_id -> Product(product_id) ON DELETE CASCADE
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Wish (
    customer_id INT NOT NULL,
    product_id INT NOT NULL,
    PRIMARY KEY (customer_id, product_id),
    FOREIGN KEY (customer_id) REFERENCES Customer(user_id) ON DELETE
CASCADE,
    FOREIGN KEY (product_id) REFERENCES Product(product_id) ON DELETE
CASCADE
);
```

4.18 Add_To_Shopping_Cart

Relational Model:

```
Add_To_Shopping_Cart(customer_id, product_id, count)
```

Functional Dependencies:

```
customer_id, product_id -> count
```

Candidate Keys:

```
{customer_id, product_id}
```

Primary Key:

```
(customer_id, product_id)
```

Foreign Keys:

```
customer_id -> Customer(user_id)
product_id -> Product(product_id)
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Add_To_Shopping_Cart (
   customer_id INT NOT NULL,
   product_id INT NOT NULL,
   count INT NOT NULL,
   PRIMARY KEY (customer_id, product_id),
   FOREIGN KEY (customer_id) REFERENCES Customer(user_id) ON DELETE
CASCADE,
   FOREIGN KEY (product_id) REFERENCES Product(product_id) ON DELETE
CASCADE
);
```

4.19 Business_Has_Record

Relational Model:

```
Business_Has_Record(small_business_id, record_id)
```

Functional Dependencies:

```
small_business_id, record_id -> (No other attributes)
```

Candidate Keys:

```
{small_business_id, record_id}
```

Primary Key:

```
(small_business_id, record_id)
```

Foreign Keys:

```
small_business_id -> Small_Business(user_id)
record_id -> Balance_record(record_id)
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Business_Has_Record (
   small_business_id INT NOT NULL,
   record_id INT NOT NULL,
   PRIMARY KEY (record_id, small_business_id),
   FOREIGN KEY (small_business_id) REFERENCES Small_Business(user_id) ON
DELETE CASCADE,
   FOREIGN KEY (record_id) REFERENCES Balance_Record(record_id) ON
DELETE CASCADE
);
```

4.20 Customer_Has_Record

Relational Model:

```
Customer_Has_Record(customer_id, record_id)
```

Functional Dependencies:

```
customer_id, record_id -> (No other attributes)
```

Candidate Keys:

```
{customer_id, record_id}
```

Primary Key:

```
(customer_id, record_id)
```

Foreign Keys:

```
customer_id -> Customer(user_id)
record_id -> Balance_Record(record_id)
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Customer_Has_Record (
   customer_id INT NOT NULL,
   record_id INT NOT NULL,
   PRIMARY KEY (record_id, customer_id),
   FOREIGN KEY (customer_id) REFERENCES Customer(user_id) ON DELETE
CASCADE,
   FOREIGN KEY (record_id) REFERENCES Balance_Record(record_id) ON
DELETE CASCADE
);
```

4.21 Transaction

Relational Model:

```
Transaction(product_id, customer_id, small_business_id,
transaction_status, transaction_date, count)
```

Functional Dependencies:

```
product_id, customer_id, small_business_id -> transaction_date, count,
transaction_status
```

Candidate Keys:

```
{product_id, customer_id, small_business_id}
```

Primary Key:

```
(product_id, customer_id, small_business_id)
```

Foreign Keys:

```
product_id -> Product(product_id)
customer_id -> Customer(user_id)
small_business_id -> Small_Business(user_id)
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Transaction (
   transaction_id INT NOT NULL AUTO_INCREMENT,
   product_id INT NOT NULL,
   customer_id INT NOT NULL,
   small_business_id INT NOT NULL,
   transaction_date DATE NOT NULL,
   count INT NOT NULL,
   transaction_status VARCHAR(255) NOT NULL,
   PRIMARY KEY (transaction_id),
   FOREIGN KEY (product_id) REFERENCES Product(product_id) ON DELETE
CASCADE,
   FOREIGN KEY (small_business_id) REFERENCES Small_Business(user_id) ON
DELETE CASCADE,
   FOREIGN KEY (customer_id) REFERENCES Customer(user_id) ON DELETE
CASCADE
);
```

4.22 Add_Product

Relational Model:

Add_Product(<u>product_id</u>, <u>small_business_id</u>, post_date)

Functional Dependencies:

product_id, small_business_id -> post_date

Candidate Keys:

{product_id, small_business_id}

Primary Key:

(product_id, small_business_id)

Foreign Keys:

product_id -> Product(product_id)
small_business_id -> Small_Business(user_id)

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Add_Product (
   product_id INT NOT NULL,
   small_business_id INT NOT NULL,
   post_date DATE NOT NULL,
   PRIMARY KEY (product_id, small_business_id),
   FOREIGN KEY (product_id) REFERENCES Product(product_id) ON DELETE
CASCADE,
   FOREIGN KEY (small_business_id) REFERENCES Small_Business(user_id) ON
DELETE CASCADE
);
```

4.23 Made_By

Relational Model:

```
Made_By(product_id, material_id)
```

Functional Dependencies:

```
product_id, material_id -> (No other attributes)
```

Candidate Keys:

```
{product_id, material_id}
```

Primary Key:

```
(product_id, material_id)
```

Foreign Keys:

```
product_id -> Product(product_id)
material_id -> Material(material_id)
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Made_By (
   product_id INT NOT NULL,
   material_id INT NOT NULL,
   PRIMARY KEY (product_id, material_id),
   FOREIGN KEY (product_id) REFERENCES Product(product_id) ON DELETE
CASCADE,
   FOREIGN KEY (material_id) REFERENCES Material(material_id) ON DELETE
CASCADE
);
```

4.24 Is_For

Relational Model:

```
Is_For(product_id, recipient_id)
```

Functional Dependencies:

```
product_id, recipient_id -> (No other attributes)
```

Candidate Keys:

```
{product_id, recipient_id}
```

Primary Key:

```
(product_id, recipient_id)
```

Foreign Keys:

```
product_id -> Product(product_id)
recipient_id -> Recipient(recipient_id)
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Is_For (
   product_id INT NOT NULL,
   recipient_id INT NOT NULL,
   PRIMARY KEY (product_id, recipient_id),
   FOREIGN KEY (product_id) REFERENCES Product(product_id) ON DELETE
CASCADE,
   FOREIGN KEY (recipient_id) REFERENCES Recipient(recipient_id) ON DELETE
CASCADE
);
```

PROCEDURES

4.25 CartAdder

This procedure is used to add a specified amount of a product to a customer's shopping cart. If the product is already in the cart, it updates the count; otherwise, it inserts a new record. It also updates the product inventory accordingly.

```
CREATE PROCEDURE CartAdder(IN in_customer_id INT, IN in_product_id INT, IN
product_amount INT)
BEGIN
   DECLARE product_amount_in_cart INT;

   SELECT SUM(count) INTO product_amount_in_cart
   FROM Add_To_Shopping_Cart AS A
   WHERE A.product_id = in_product_id AND A.customer_id = in_customer_id;

   IF product_amount_in_cart > 0 THEN
      UPDATE Add_To_Shopping_Cart
      SET count = count + product_amount
      WHERE product_id = in_product_id AND customer_id = in_customer_id;

      UPDATE Product
      SET amount = amount - product_amount
      WHERE product_id = in_product_id;
   ELSE
      INSERT INTO Add_To_Shopping_Cart (customer_id, product_id, count)
      VALUES (in_customer_id, in_product_id, product_amount);

      UPDATE Product
      SET amount = amount - product_amount
      WHERE product_id = in_product_id;
   END IF;
END;
```

4.26 Rate Product

This procedure allows a customer to rate a product. If the customer has already rated the product, it updates the rating. If not, it inserts a new rating.

```
CREATE PROCEDURE RateProduct(IN customer_id INT, IN product_id INT, IN
rate_amount INT)
BEGIN
   DECLARE old_rate_exist INT;

   SELECT COUNT(*) INTO old_rate_exist
   FROM Rate AS R
   WHERE R.product_id = product_id AND R.customer_id = customer_id;

   IF old_rate_exist > 0 THEN
      UPDATE Rate
      SET star = rate_amount
      WHERE product_id = product_id AND customer_id = customer_id;
   ELSE
      INSERT INTO Rate (customer_id, product_id, star)
      VALUES (customer_id, product_id, rate_amount);
   END IF;
END;
```

4.27 ProductPrinter

This procedure retrieves a paginated list of products with details such as title, description, price, available amount, average rating, number of ratings, images, business name, and wishlist status. It also supports pagination.

```
CREATE PROCEDURE ProductPrinter(IN per_page INT, IN start_index INT, IN
wish_user_id INT)
BEGIN
   SELECT P.product_id, P.title, P.description, P.price, P.amount,
        ROUND(COALESCE(R.avg_rating, 0), 1) AS average_rating,
        COALESCE(R.num_rating, 0) AS number_of_rating, P.images,
        SB.business_name,
        IF(W.product_id IS NULL, 0, 1) AS is_in_wishlist
   FROM Product P
   LEFT JOIN (
      SELECT product_id, AVG(star) AS avg_rating, COUNT(*) AS num_rating
      FROM Rate
      GROUP BY product_id
   ) R ON P.product_id = R.product_id
   LEFT JOIN (
      SELECT product_id
```

```
        FROM Wish
        WHERE customer_id = wish_user_id
    ) W ON P.product_id = W.product_id
    JOIN Add_Product AP ON P.product_id = AP.product_id
    JOIN Small_Business SB ON AP.small_business_id = SB.user_id
    ORDER BY P.product_id DESC
    LIMIT per_page OFFSET start_index;
END;
```

4.28 ProductFilter

The ProductFilter stored procedure retrieves a paginated list of products based on various filters and sorting criteria. It accepts parameters for pagination (per_page, start_index), filtering by business name, price range, product name, recipient name, material name, and wishlist status. The products are joined with related tables to gather business names, ratings, and wishlist status. The results are sorted according to the specified sort_method and limited by the specified page size and offset.

```
CREATE PROCEDURE ProductFilter(
    IN per_page INT,
    IN start_index INT,
    IN filter_business_name VARCHAR(255),
    IN filter_min_price DECIMAL(10,2),
    IN filter_max_price DECIMAL(10,2),
    IN sort_method INT,
    IN wish_user_id INT,
    IN search_product_name VARCHAR(255),
    IN filter_recipient_name VARCHAR(255),
    IN filter_material_name VARCHAR(255)
)
BEGIN
    SELECT
        P.product_id,
        P.title,
        P.description,
        P.price,
        P.amount,
        ROUND(COALESCE(R.avg_rating, 0), 1) AS average_rating,
        COALESCE(R.num_rating, 0) AS number_of_rating,
        P.images,
        SB.business_name,
        IF(W.product_id IS NULL, 0, 1) AS is_in_wishlist
    FROM Product P
    LEFT JOIN (
        SELECT product_id, AVG(star) AS avg_rating, COUNT(*) AS num_rating
        FROM Rate
        GROUP BY product_id
    ) R ON P.product_id = R.product_id
    JOIN Add_Product AP ON P.product_id = AP.product_id
    JOIN Small_Business SB ON AP.small_business_id = SB.user_id
    LEFT JOIN (
        SELECT product_id
        FROM Wish
        WHERE customer_id = wish_user_id
    ) W ON P.product_id = W.product_id
    LEFT JOIN Made_By MB ON P.product_id = MB.product_id
    LEFT JOIN Is_For IFOR ON P.product_id = IFOR.product_id
    LEFT JOIN Material M ON MB.material_id = M.material_id
    LEFT JOIN Recipient R ON IFOR.recipient_id = R.recipient_id
    LEFT JOIN Ban B ON SB.user_id = B.small_business_id
    WHERE SB.business_name LIKE CONCAT('%', filter_business_name, '%')
    AND P.price BETWEEN filter_min_price AND filter_max_price
    AND P.title LIKE CONCAT('%', search_product_name, '%')
    AND (filter_recipient_name = '' OR R.recipient_name = filter_recipient_name)
    AND (filter_material_name = '' OR M.material_name = filter_material_name)
    AND B.small_business_id IS NULL
    ORDER BY
        CASE
            WHEN sort_method = 0 THEN P.product_id
            WHEN sort_method = 1 THEN P.price
            WHEN sort_method = 3 THEN P.product_id
        END DESC,
        CASE
            WHEN sort_method = 2 THEN P.price
            WHEN sort_method = 4 THEN P.product_id
        END ASC
    LIMIT per_page OFFSET start_index;
END;
```

4.29 CategoryProductFilter

The CategoryProductFilter stored procedure retrieves products based on category, subcategory, and other filters, with sorting and pagination. It accepts parameters for page size, start index, business name, price range, sort method, wishlist status, category name, and subcategory name. The query joins the necessary tables to gather product details, ratings, business information, and wishlist status. It filters results by the specified criteria and sorts them according to the sort_method, limiting the output to the specified number of products per page starting at the given index.

```
CREATE PROCEDURE CategoryProductFilter(
    IN per_page INT,
    IN start_index INT,
        IN filter_business_name VARCHAR(255),
        IN filter_min_price DECIMAL(10,2),
        IN filter_max_price DECIMAL(10,2),
        IN sort_method INT,
        IN wish_user_id INT,
        IN category_name VARCHAR(255),
        IN subcategory_name VARCHAR(255)
    )
    BEGIN
        SELECT P.product_id, P.title, P.description, P.price, P.amount,
               ROUND(COALESCE(R.avg_rating, 0), 1) AS average_rating,
               COALESCE(R.num_rating, 0) AS number_of_rating, P.images,
               SB.business_name,
               IF(W.product_id IS NULL, 0, 1) AS is_in_wishlist
        FROM Product P
        JOIN In_Category IC ON P.product_id = IC.product_id
        JOIN Main_Category MC ON IC.main_category_id =
MC.main_category_id
        JOIN Sub_Category SC ON IC.sub_category_id =
SC.sub_category_id
        LEFT JOIN (
           SELECT product_id, AVG(star) AS avg_rating, COUNT(*) AS
num_rating
           FROM Rate
           GROUP BY product_id
        ) R ON P.product_id = R.product_id
        JOIN Add_Product AP ON P.product_id = AP.product_id
        JOIN Small_Business SB ON AP.small_business_id = SB.user_id
        LEFT JOIN (
           SELECT product_id
           FROM Wish
           WHERE customer_id = wish_user_id
        ) W ON P.product_id = W.product_id
        WHERE SB.business_name LIKE CONCAT('%',
filter_business_name, '%')
        AND P.price BETWEEN filter_min_price AND filter_max_price
        AND MC.main_category_name = category_name AND
SC.sub_category_name = subcategory_name
        ORDER BY
           CASE
              WHEN sort_method = 0 THEN P.product_id
              WHEN sort_method = 1 THEN P.price
              WHEN sort_method = 3 THEN P.product_id
           END DESC,
            CASE
                WHEN sort_method = 2 THEN P.price
                WHEN sort_method = 4 THEN P.product_id
            END ASC
        LIMIT per_page OFFSET start_index;
    END;
```

4.30 BusinessProducts

This procedure retrieves all products a specific business offers based on the business name. It includes product details and average ratings.

```
CREATE PROCEDURE BusinessProducts(
   IN filter_business_name VARCHAR(255)
)
BEGIN
   SELECT P.product_id, P.title, P.description, P.price, P.amount,
         ROUND(COALESCE(R.avg_rating, 0), 1) AS average_rating,
         COALESCE(R.num_rating, 0) AS number_of_rating, P.images,
         SB.business_name
   FROM Product P
   LEFT JOIN (
      SELECT product_id, AVG(star) AS avg_rating, COUNT(*) AS num_rating
      FROM Rate
      GROUP BY product_id
   ) R ON P.product_id = R.product_id
   JOIN Add_Product AP ON P.product_id = AP.product_id
   JOIN Small_Business SB ON AP.small_business_id = SB.user_id
   WHERE SB.business_name LIKE CONCAT('%', filter_business_name, '%');
END
```

4.31 ReturnProduct

This procedure handles the return of a product. It updates the transaction status, restores the product inventory, adjusts the customer's balance, and deducts the corresponding amount from the small business's balance.

```
CREATE PROCEDURE ReturnProduct(IN return_customer_id INT, IN
return_product_id INT, IN return_transaction_date DATE, IN return_transaction_id
INT, IN small_business_id INT)
BEGIN
   DECLARE product_amount_transaction INT;
   DECLARE product_price DECIMAL(10,2);

   SELECT SUM(count) INTO product_amount_transaction
```

```
    FROM Transaction AS T
    WHERE T.transaction_id = return_transaction_id;

    SELECT price INTO product_price
    FROM Product AS P
    WHERE P.product_id = return_product_id;

    UPDATE Transaction
    SET transaction_status = 'Returned'
    WHERE transaction_id = return_transaction_id;

    UPDATE Product
    SET amount = amount + product_amount_transaction
    WHERE product_id = return_product_id;

    UPDATE Customer
    SET balance = balance + (product_amount_transaction * product_price)
    WHERE user_id = return_customer_id;

    UPDATE Small_Business
    SET balance = balance - (product_amount_transaction * product_price)
    WHERE user_id = small_business_id;
END;
```

VIEWS

4.32 UserTransactions

This view provides detailed information about a user's transactions, including product details, business name, transaction date, count, transaction status, and user rating. It consolidates data from multiple tables to offer a comprehensive view of user transactions.

```
CREATE VIEW UserTransactions AS
SELECT
    T.product_id,
    P.title AS product_title,
    P.images AS product_image,
    P.description AS product_description,
    P.price AS product_price,
    T.small_business_id,
    SB.business_name,
    DATE_FORMAT(T.transaction_date, '%m-%d-%Y') AS transaction_date,
    T.count,
    T.transaction_status,
    R.star AS user_rating,
    U.user_id AS customer_id,
   T.transaction_id AS transaction_id
FROM
   Transaction T
JOIN
   Product P ON T.product_id = P.product_id
JOIN
   Customer C ON T.customer_id = C.user_id
JOIN
   User U ON C.user_id = U.user_id
JOIN
   Small_Business SB ON T.small_business_id = SB.user_id
LEFT JOIN
   Rate R ON T.customer_id = R.customer_id AND T.product_id = R.product_id;
```

5. Implementation Details

We crafted our platform with Python Django at the backend, orchestrating user management, purchase processing, and data storage with MySQL. On the front end, we leveraged Jinja templates along with jQuery for interactivity and Bootstrap for a polished, responsive layout suitable for all screen sizes. Wel handled all interactions between our application and database directly through SQL queries rather than automation tools or libraries like ORM.

Frontend Implementation

1. Framework: The frontend is built using Django for server-side rendering, combined with JavaScript and jQuery for dynamic interactions.
2. UI Components: Element UI components are used to create a responsive and user-friendly interface.
3. Styling: Bootstrap is used for consistent styling and layout across the application.
4. Pages and Components:
    - Homepage: Displays featured products and categories.
    - Login: User authentication page.
    - Register as Business: Registration page for businesses.
    - Register as Customer: Registration page for customers.
    - User Profile: Displays user information and allows profile editing.
    - Business Balance History View: Displays the transaction history for businesses.
    - Create Product: Interface for adding new products.
    - Edit Product: Interface for editing existing products.
    - Edit Business Profile: Interface for businesses to update their profile.
    - Edit User Profile: Interface for users to update their profile.
    - List Products: Displays a list of products with filters.

- Main Page for Users: Main landing page for logged-in users.
- User Wishlist: Displays products the user has added to their wishlist.
- User Shopping Cart: Displays the user's shopping cart.
- Admin Pages: For administrative tasks such as banning businesses and viewing system reports.

Backend Implementation

1. Framework: Django is used to handle backend logic and routing.
2. Database: MySQL is used as the database system, with raw SQL queries for database interactions.
3. SQL Schemas
4. Advanced SQL Features: Procedures and Views
5. Docker
    - Containerization: Docker is used to containerize the application, ensuring consistent environments for development and deployment.
    - Docker Compose: Used to manage multi-container applications, including web and database services.

6. User Manual

6.1 Build Instructions

- This project uses Docker, so the first thing to do is to download Docker. For more information, please use the official documentation.
- Similarly, you need to clone the project's repository from GitHub to your system using "git clone".
- Then, using your operating system's shell, navigate to the directory where you cloned the repository.
- To run the project, execute the following commands in your operating system's shell.
    - docker build -t craftvalley .
    - docker compose up -d
- Once your terminal indicates that containers are up and functional, you can use Craftvalley at http://localhost:3000/auth/login/
    - If the app is not working properly, go to the docker-containers, run the migration-1 container, and restart the web-1 container.
- You can close the application from the terminal with the "docker compose down" command.

6.2 Login / Sign Up

Customers or small businesses have to log in first to use the app. The app will redirect unlogged users to the login page. All types of users use a single login page. Users can sign up by following the register as a customer or register as a small business button to sign up. Customers must fill in username, e-mail, password, and phone number. Note that existing usernames, e-mails, and phone numbers can not be used again. Small businesses must fill in username, password, phone number, business name, title, and description. These fields are restricted similarly to customer registration.

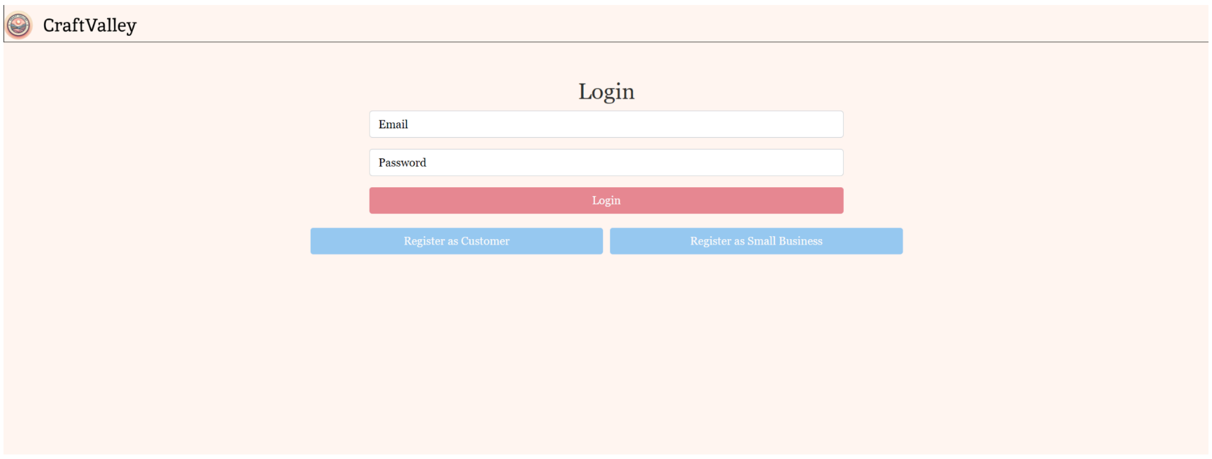


Figure 2

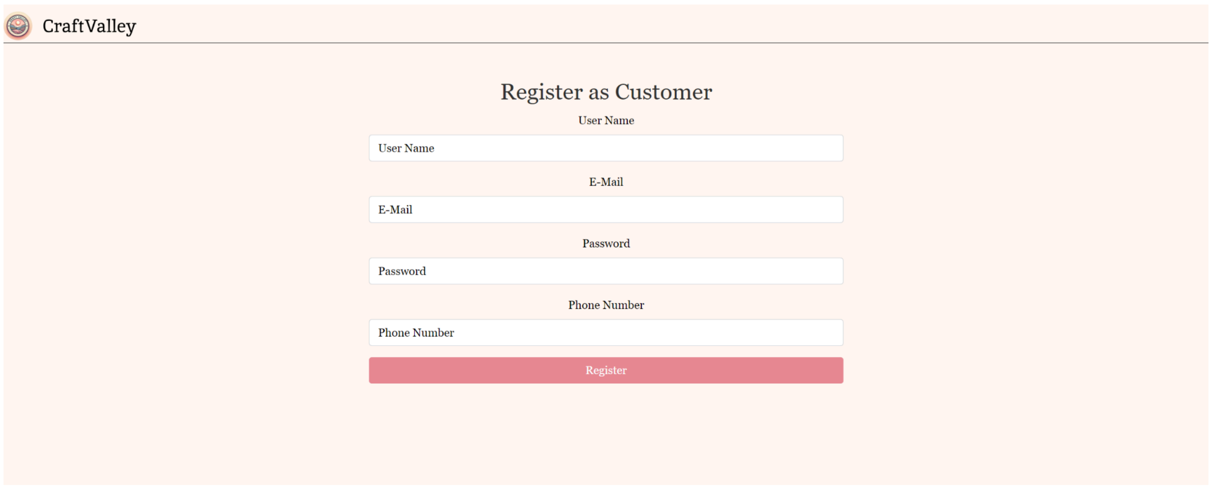


Figure3

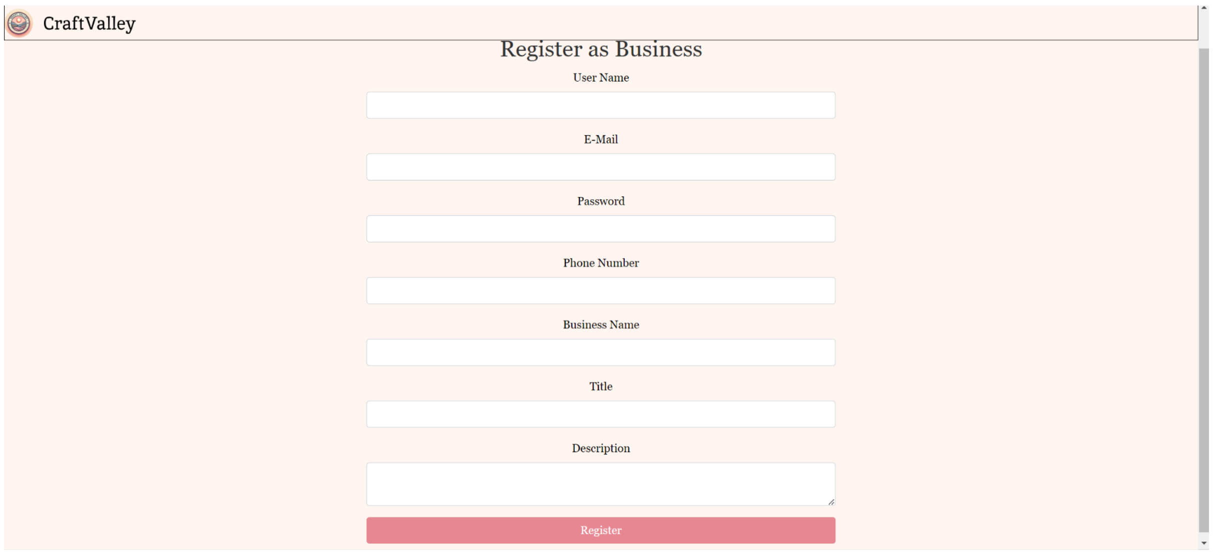


Figure4

6.3 Customer Homepage

After logging in, customers are met with the homepage. On the navigation bar, they have the logo which redirects to the homepage, categories, and a category's sub-categories. When clicking on a sub-category below only said products are shown. From the user icon, they can go to their profile. From the heart icon, they can see their wishlist; the shopping cart icon takes them to checkout; finally, they can log out with the logout icon. Below are the products. Here they can see some details about products, however to see more details they need to click on the desired product. They can wishlist products with the wishlist button. The pop-up window with the details of a product also allows the customer to pick a quantity and add it to the cart. Note that the system does not allow customers to add more than there is available. Customers can also report businesses by using the report button.

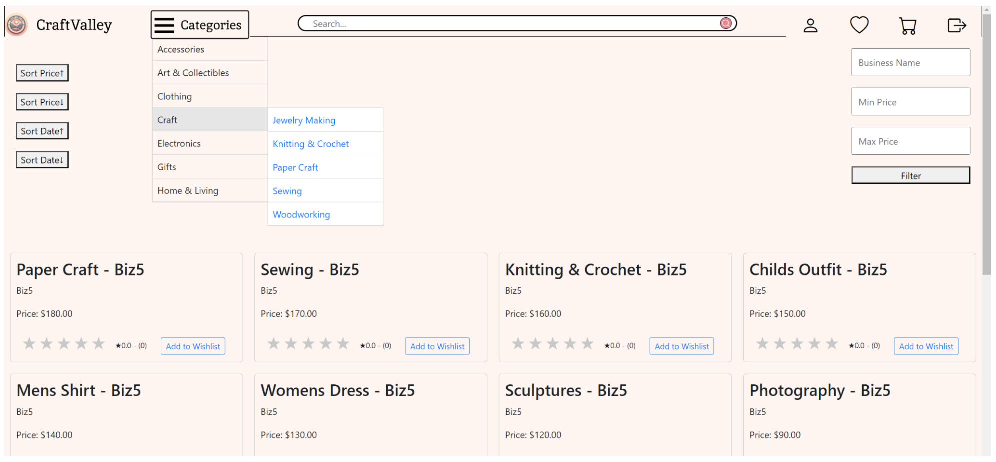
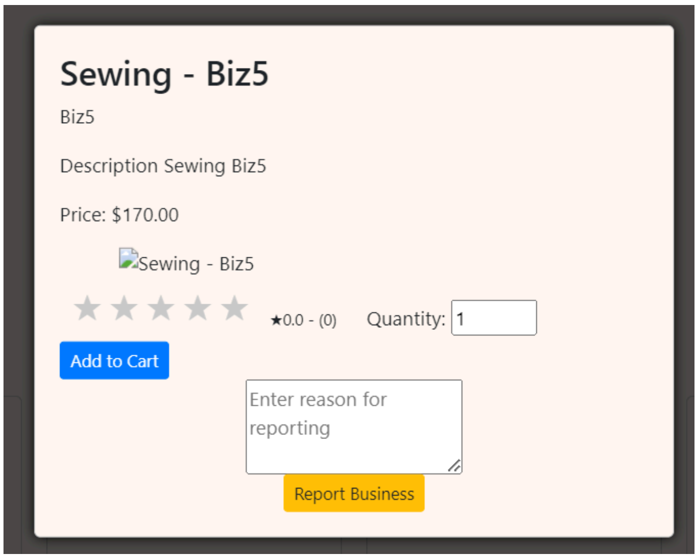


Figure 5


Sewing - Biz5

Biz5

Description Sewing Biz5

Price: $170.00

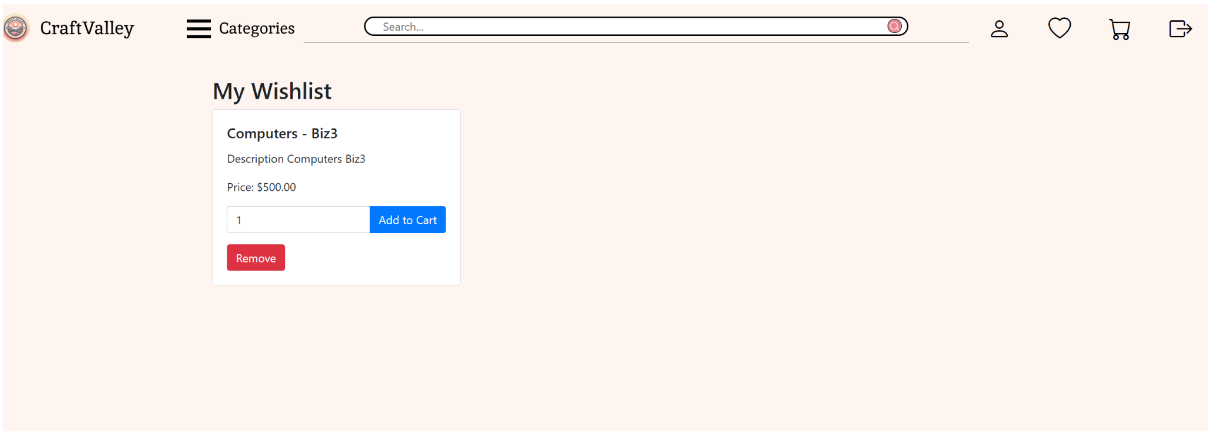
Sewing - Biz5

★0.0 - (0) Quantity: 1

Figure 6

6.4 Wishlist

Customers can view their wishlisted items on the wishlist page. Here they can remove items from their wishlist or choose an amount and add the to the cart. Note that users cannot add more than available.



My Wishlist

Computers - Biz3

Description Computers Biz3

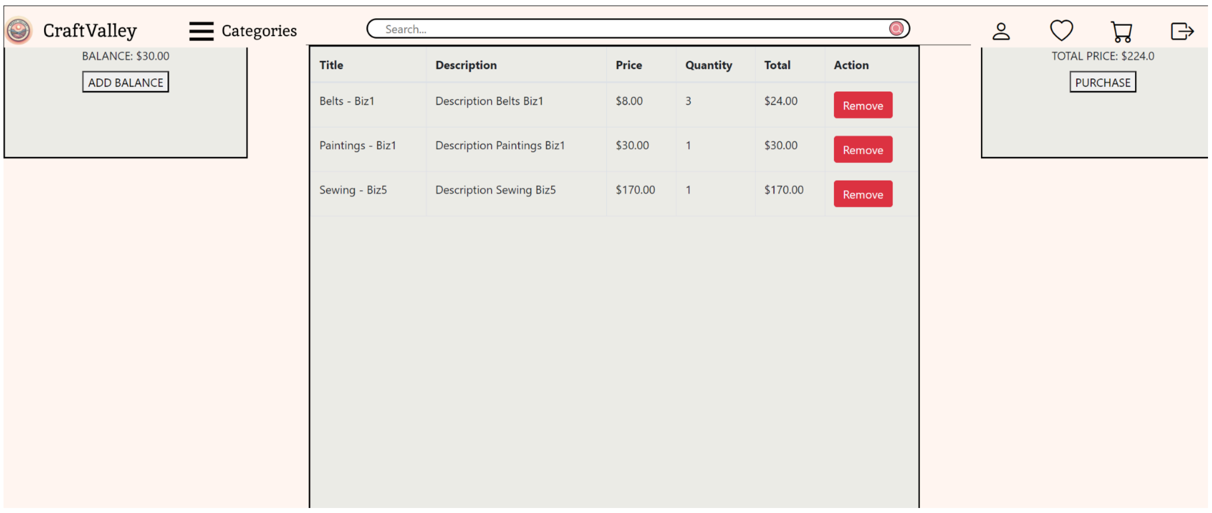
Price: $500.00

1

Figure 7

6.5 Shopping Cart

Customers can see the items they have added to their cart and their prices on the shopping cart page. They can also see their balance and the total cost of the products. Note that the system gives a pop-up error message when purchase button is pressed and the balance is lower than total price. Customers can also add balance using the add balance button. After a purchase is completed, the total price amount is deducted from the customer, and the money is added to the balance of the small businesses.


CraftValley

Categories

Search...

BALANCE: $30.00

ADD BALANCE

TOTAL PRICE: $224.0

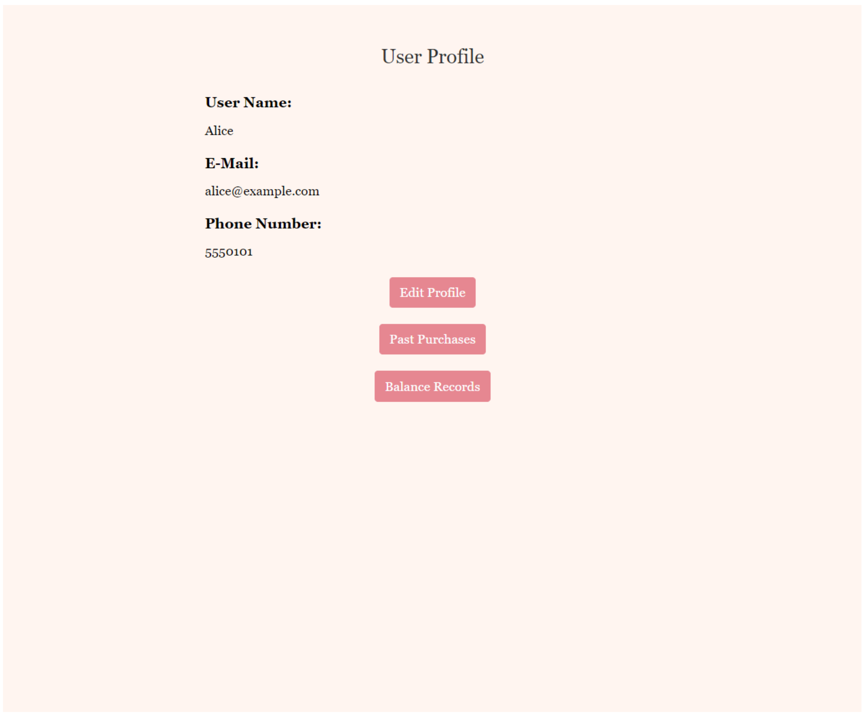
PURCHASE

Title	Description	Price	Quantity	Total	Action
Belts - Biz1	Description Belts Biz1	$8.00	3	$24.00	Remove
Paintings - Biz1	Description Paintings Biz1	$30.00	1	$30.00	Remove
Sewing - Biz5	Description Sewing Biz5	$170.00	1	$170.00	Remove

Figure 8

6.6 Customer Profile

Users can see their information on Customer Profile. Here there are two buttons. Edit profile takes you to the edit page where they can update their information, and Past Purchases takes them to Past Purchases page. Balance Records shows every purchase, refund, and add balance that the customer did.


User Profile

User Name:

Alice

E-Mail:

alice@example.com

Phone Number:

5550101

Edit Profile

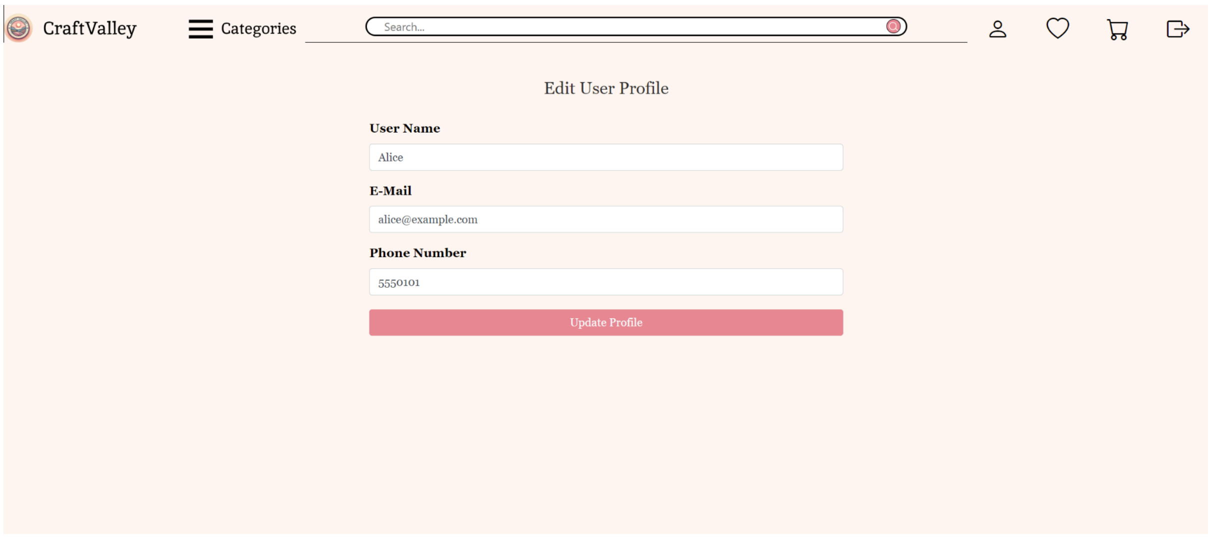
Past Purchases

Balance Records

Figure 9

6.7 Edit Customer Profile

Here, user can update their username, e-mail, and phone number. After they are done, they can click on the update profile, which updates the information and takes them back to their profile. Note that when updating information, there are similar restrictions to registering.



Edit User Profile

User Name

Alice

E-Mail

alice@example.com

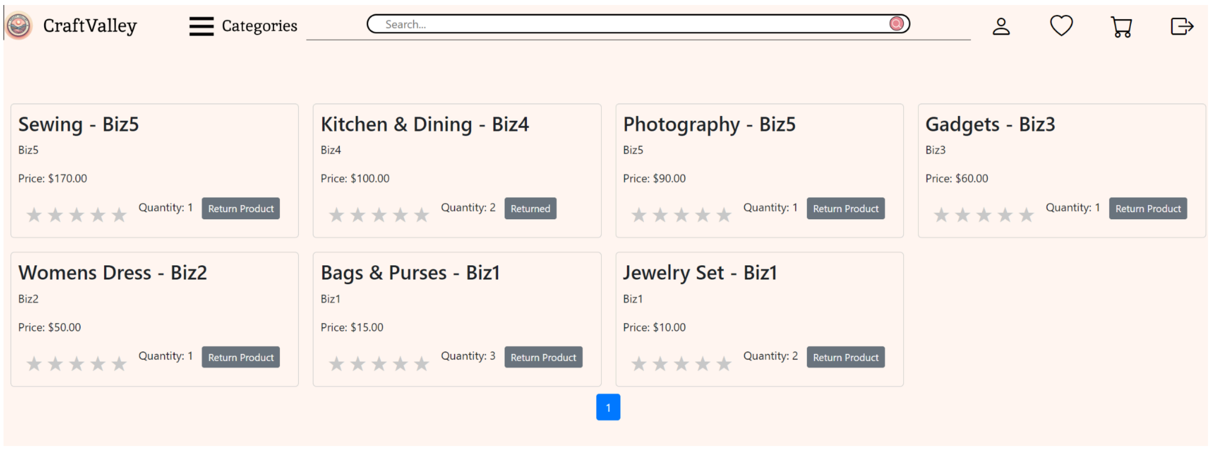
Phone Number

5550101

Figure 10

6.8 Past Purchases

On this page, customers can see their past purchases and return them. In order to return the sale, it must have happened within the last 30 days. If a product is returned, the customer and business balances are updated accordingly, and the product's available count is reverted.



Sewing - Biz5
Biz5
Price: $170.00
Quantity: 1 Return Product

Kitchen & Dining - Biz4
Biz4
Price: $100.00
Quantity: 2 Returned

Photography - Biz5
Biz5
Price: $90.00
Quantity: 1 Return Product

Gadgets - Biz3
Biz3
Price: $60.00
Quantity: 1 Return Product

Womens Dress - Biz2
Biz2
Price: $50.00
Quantity: 1 Return Product

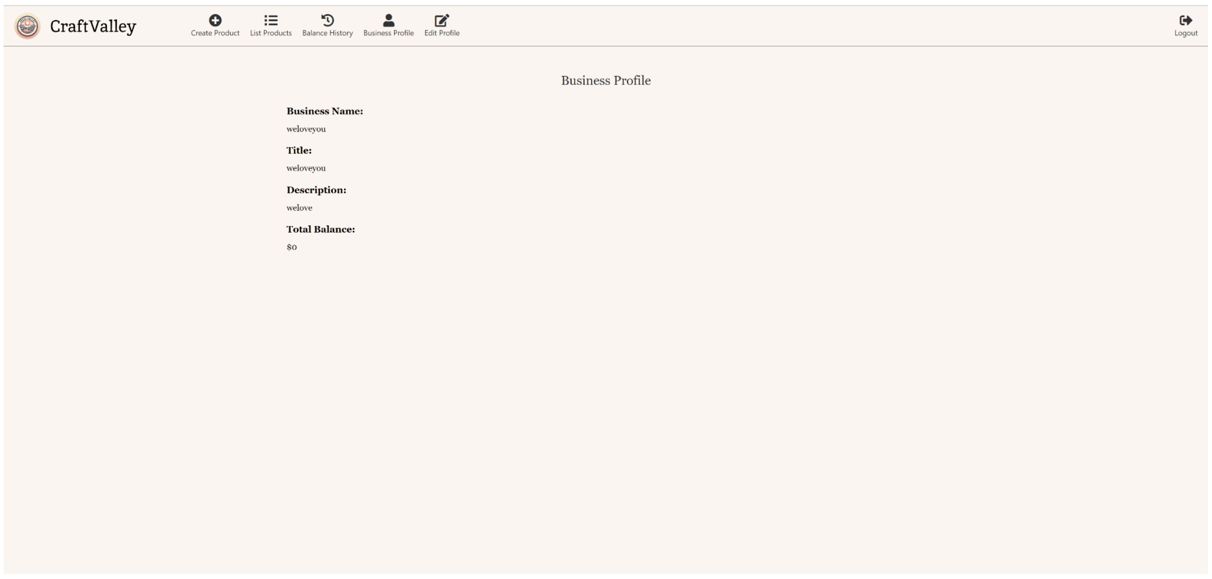
Bags & Purses - Biz1
Biz1
Price: $15.00
Quantity: 3 Return Product

Jewelry Set - Biz1
Biz1
Price: $10.00
Quantity: 2 Return Product

Figure 11

6.9 Small Business Profile

After small businesses register on the app and then log in, they are redirected to the small business profile page. On this page, you will see the business name, title, description, and total balance. By clicking on the Create Product, List Products, Balance History, and Edit Profile buttons in the bar above, you are directed to perform the relevant actions.



Business Profile

Business Name:
weloveyou

Title:
weloveyou

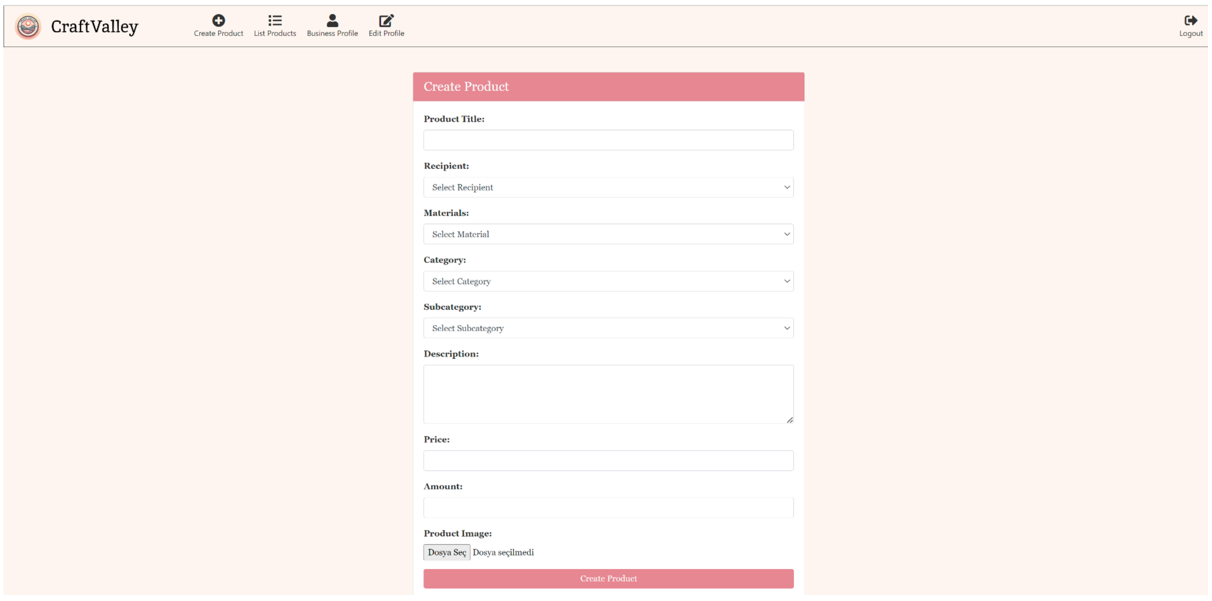
Description:
welove

Total Balance:
$0

Figure 12

6.10 Create Product

After filling in the product information, the business owner can create a product by pressing the create product button.



Create Product

Product Title:

Recipient:
Select Recipient

Materials:
Select Material

Category:
Select Category

Subcategory:
Select Subcategory

Description:

Price:

Amount:

Product Image:
Dosya Seç Dosya seçilmedi

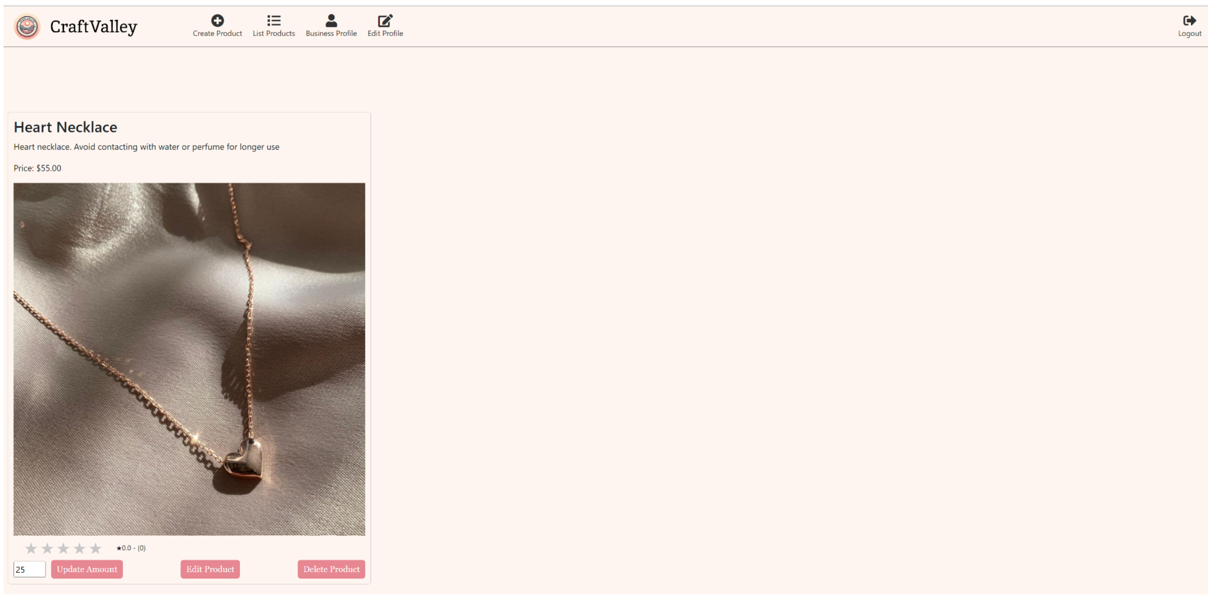
Create Product

Figure 13

6.11 List Products

On the List products page, the business displays the products it has created. Sees product name, price, rating, and stock information. Stock information can be changed using the Update Amount button. Businesses can delete the product by clicking on the Delete Product

Button. Using the Edit Product button directs the business to the pages where it can perform product-related operations.



Heart Necklace

Heart necklace. Avoid contacting with water or perfume for longer use

Price: $55.00

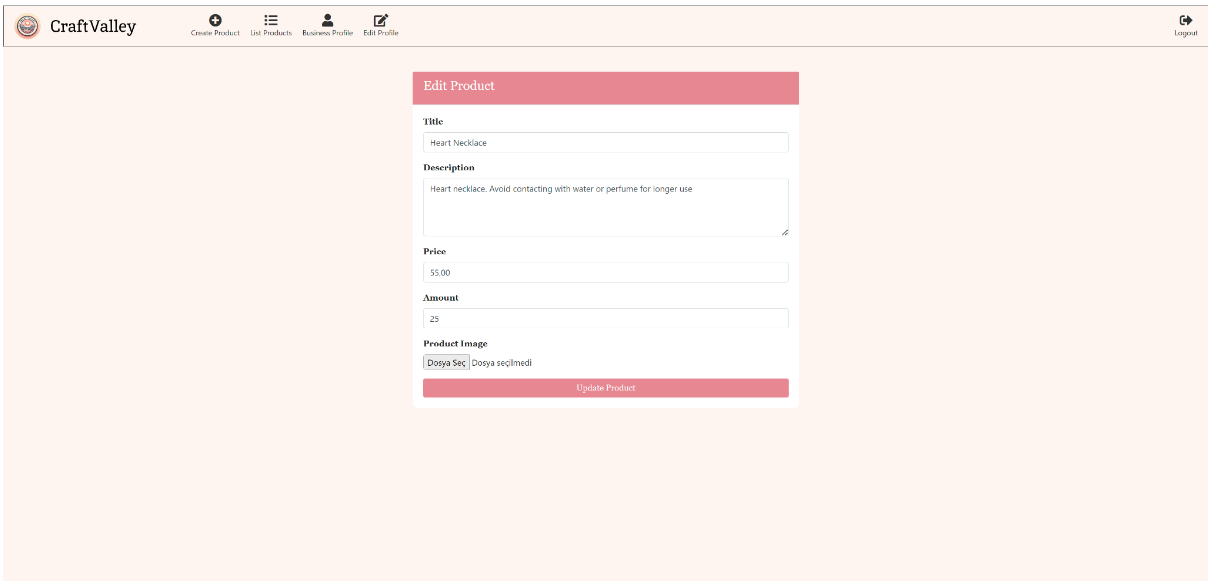
★0.0 - (0)

25 Update Amount | Edit Product | Delete Product

Figure 14

6.12 Edit Product Details

The business can update its information about the product on this page.



Edit Product

Title: Heart Necklace

Description: Heart necklace. Avoid contacting with water or perfume for longer use

Price: 55,00

Amount: 25

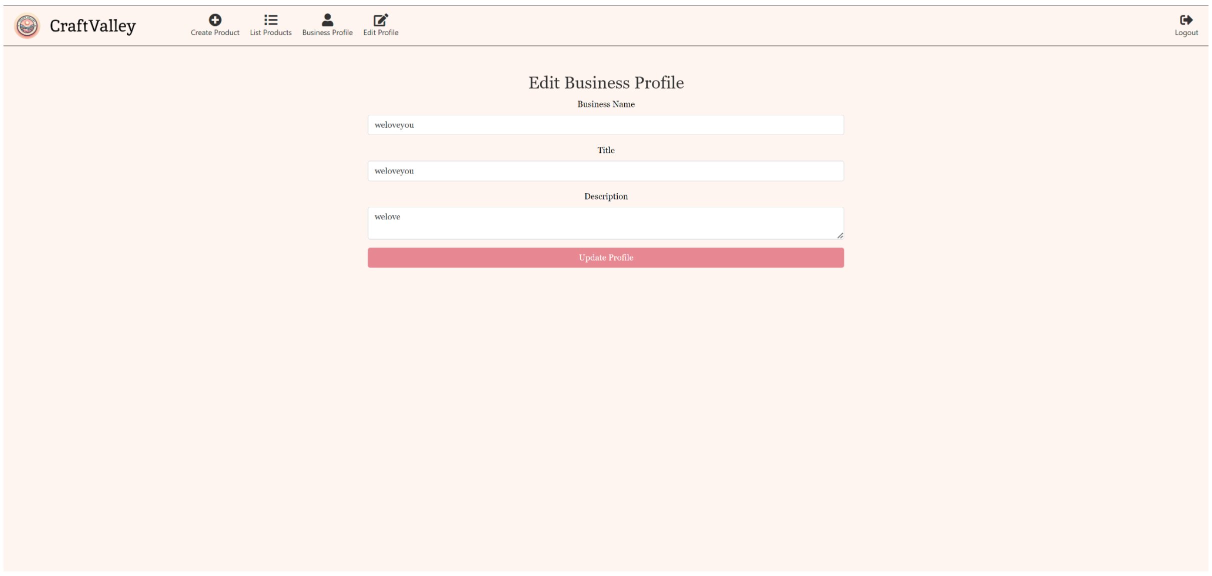
Product Image: Dosya Seç Dosya seçilmedi

Update Product

Figure 15

6.13 Edit Business Profile Details

The business can update its information about the business on this page.



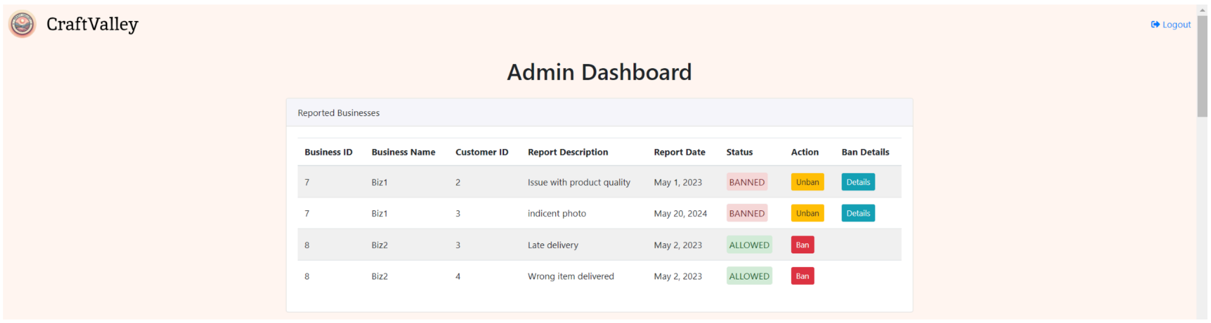
Edit Business Profile

Business Name	weloveyou
Title	weloveyou
Description	welove

Figure 16

6.14 Admin Page

Here is the Admin Dashboard where admins can view reports about small businesses made by the customers. They can ban small businesses or unban them. Banned businesses cannot login and their products are not shown on the main page.



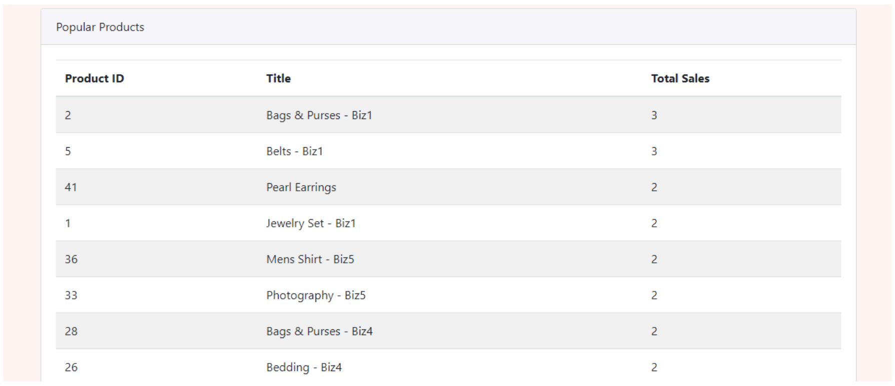
Admin Dashboard

Reported Businesses

Business ID	Business Name	Customer ID	Report Description	Report Date	Status	Action	Ban Details
7	Biz1	2	Issue with product quality	May 1, 2023	BANNED	Unban	Details
7	Biz1	3	indicent photo	May 20, 2024	BANNED	Unban	Details
8	Biz2	3	Late delivery	May 2, 2023	ALLOWED	Ban	
8	Biz2	4	Wrong item delivered	May 2, 2023	ALLOWED	Ban	

Figure 17

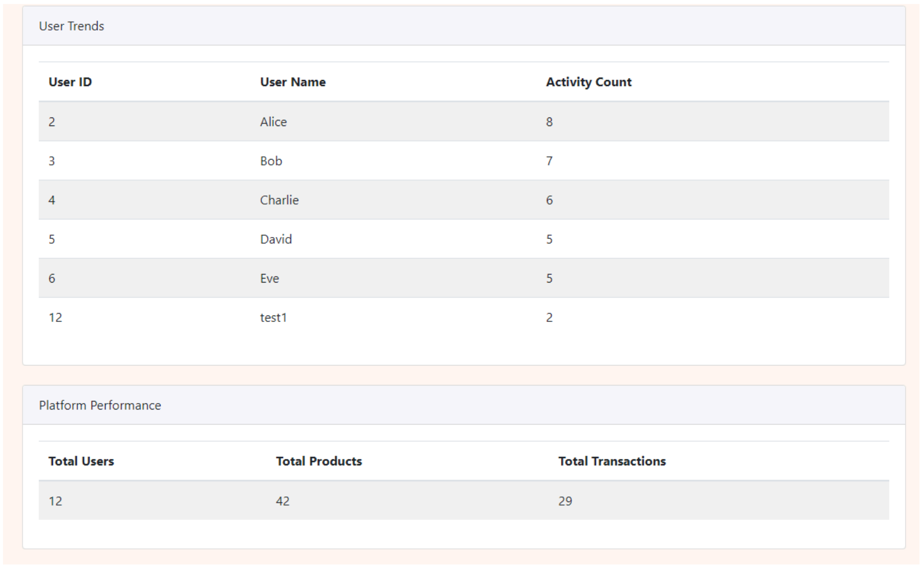
Admins can also see information like most selling products, most active users, total user count, and total product count.



Popular Products

Product ID	Title	Total Sales
2	Bags & Purses - Biz1	3
5	Belts - Biz1	3
41	Pearl Earrings	2
1	Jewelry Set - Biz1	2
36	Mens Shirt - Biz5	2
33	Photography - Biz5	2
28	Bags & Purses - Biz4	2
26	Bedding - Biz4	2

Figure 18



User Trends

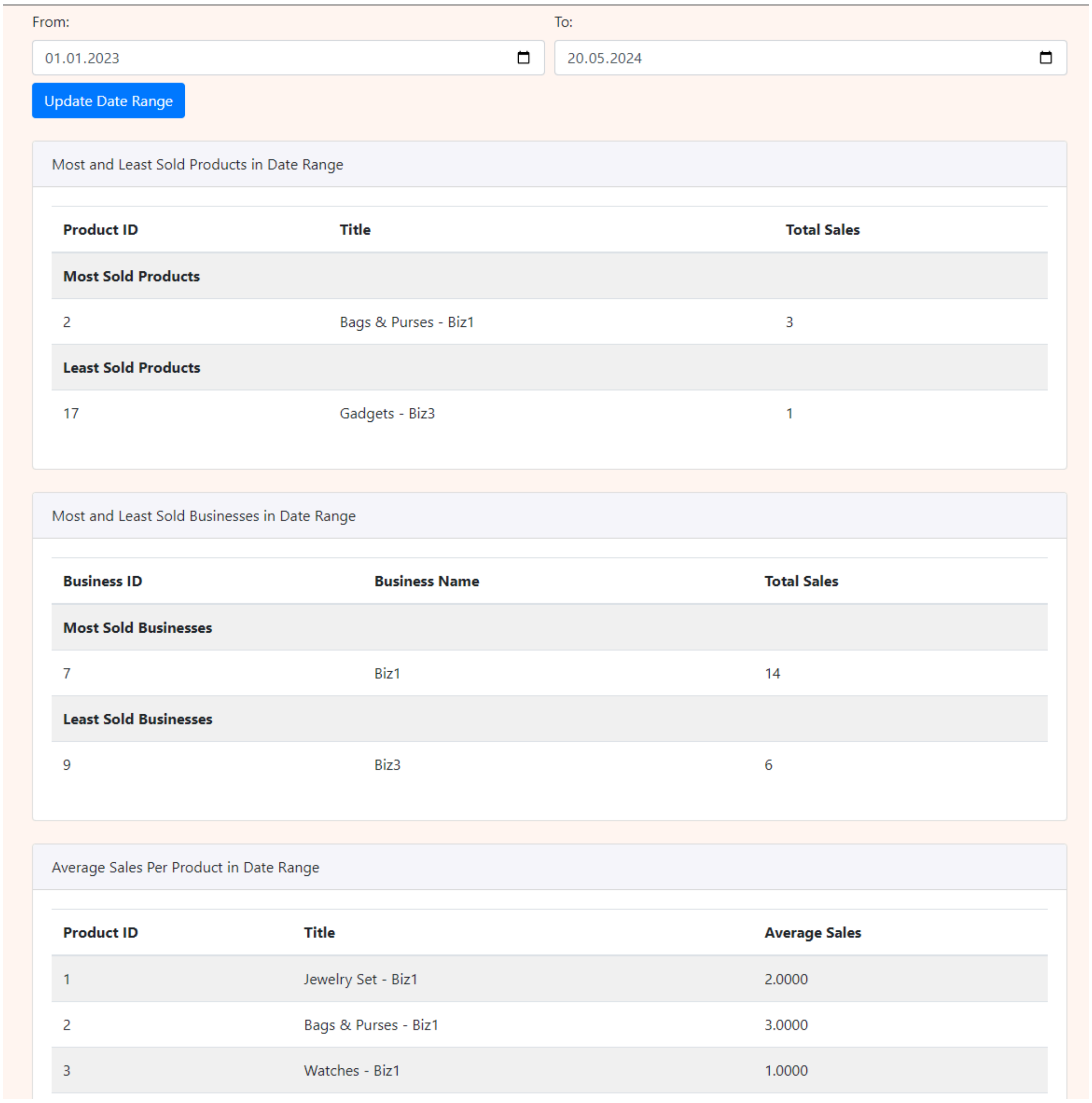
User ID	User Name	Activity Count
2	Alice	8
3	Bob	7
4	Charlie	6
5	David	5
6	Eve	5
12	test1	2

Platform Performance

Total Users	Total Products	Total Transactions
12	42	29

Figure 19

Admins can also pick a date range and see the most selling and least selling products and businesses, as well as average sales per product.


From: 01.01.2023

To: 20.05.2024

Update Date Range

Most and Least Sold Products in Date Range

Product ID	Title	Total Sales
Most Sold Products		
2	Bags & Purses - Biz1	3
Least Sold Products		
17	Gadgets - Biz3	1

Most and Least Sold Businesses in Date Range

Business ID	Business Name	Total Sales
Most Sold Businesses		
7	Biz1	14
Least Sold Businesses		
9	Biz3	6

Average Sales Per Product in Date Range

Product ID	Title	Average Sales
1	Jewelry Set - Biz1	2.0000
2	Bags & Purses - Biz1	3.0000
3	Watches - Biz1	1.0000

Figure 20