PROJECT Title : Local Food Wastage Management System

A PROJECT REPORT

SUBMITTED BY

KEERTHANA R R

Institution Name : GUVI-HCL

DOMAIN : Food Management

Date Submitted : 11.05.2025

ABSTRACT

Food wastage is a growing concern globally, particularly when juxtaposed with widespread food insecurity. A significant amount of food is discarded daily by restaurants, households, and other food providers, while many individuals and communities face hunger. This project, titled **"Local Food Wastage Management System,"** aims to bridge this gap by creating a digital platform where surplus food can be listed and claimed efficiently.The system enables food providers—such as restaurants, caterers, and individuals—to register and list excess food items. On the other end, receivers—such as NGOs, volunteers, or individuals in need—can browse and claim the available food. The application is developed using **Python** and **Streamlit** for the interface, with **SQLite** as the backend database. It also includes features for **data visualization**, showing trends in food wastage, provider contributions, and claim patterns to help optimize food redistribution efforts.The primary objectives of this project are to **reduce food waste**, **combat hunger**, and promote **social good** by leveraging technology. Through simple CRUD operations, real-time listings, and analytical insights, the system provides a practical and scalable approach to food management in local communities.This project not only highlights technical competencies in **Python, SQL, and Data Analysis**, but also emphasizes the societal impact of technology in solving real-world problems.

TABLE OF CONTENTS

1.Introduction

Food wastage is a major global issue, contributing to both environmental degradation and social inequality. While large quantities of edible food are discarded daily by households, restaurants, and stores, millions of people continue to suffer from hunger and malnutrition. This imbalance highlights a critical need for systems that can bridge the gap between surplus and scarcity.

The **Local Food Wastage Management System** is a digital platform developed to reduce food waste at the community level by facilitating the redistribution of excess food to those in need. Built using Python, Streamlit, and SQLite, this system allows local food providers—such as restaurants and individuals—to list surplus food items. At the same time, NGOs and food-insecure individuals can claim this food based on their needs and location.

This project not only addresses a key social issue but also demonstrates how data-driven, technology-enabled solutions can create meaningful impact. It combines web application development, database management, and data analysis to deliver an intuitive and actionable tool for waste reduction and food access.

2. Problem Statement

Food wastage is a significant issue, with many households and restaurants discarding surplus food while numerous people struggle with food insecurity. This project aims to develop a Local Food Wastage Management System, where:

1. Restaurants and individuals can list surplus food.
2. NGOs or individuals in need can claim the food.
3. SQL stores available food details and locations.
4. A Streamlit app enables interaction, filtering, CRUD operation and visualization

Despite global food shortages and hunger, large amounts of edible food go to waste daily. There is a clear disconnect between those who have excess food and those who need it. The lack of a structured, accessible system to manage and redistribute surplus food exacerbates this problem. This project solves this by building a digital food management system for local communities.

3. Objectives

- Create a user-friendly platform to list and claim surplus food.
- Allow food providers and receivers to register update and delete records.
- Store and retrieve food data securely using SQL.
- Enable data analysis to track trends, demands, and wastage.
- Encourage social good by reducing food wastage and aiding those in need.

4. Skills Gained

- Python Programming
- SQL and Database Design
- Streamlit Web Application Development
- CRUD Operations
- Data Analysis & Visualization
- Project Structuring in VS Code
- Domain Understanding: Food Distribution and Waste Reduction

5. Technology Stack

Component	Technology
Language	Python
Web Framework	Streamlit
Database	SQLite (SQL)
Libraries	Pandas, Matplotlib, Seaborn
IDE	Visual Studio Code
File Format	CSV for Datasets

6. System Architecture

Frontend: Streamlit-based UI for user interaction.
Backend: SQLite database to store food listings, providers, claims, and receivers.
Data Layer: Python scripts for queries, analysis, and CRUD operations.

[Providers / Receivers]
↓
Streamlit UI
↓
Python Logic
↓
SQLite Database

7. Database Design

a).Providers Dataset

The providers.csv file contains details of food providers who contribute surplus food to the system.

- **Provider_ID** (Integer) – Unique identifier for each provider.
- **Name** (String) – Name of the food provider (e.g., restaurants, grocery stores, supermarkets).
- **Type** (String) – Category of provider (e.g., Restaurant, Grocery Store, Supermarket).
- **Address** (String) – Physical address of the provider.
- **City** (String) – City where the provider is located.
- **Contact** (String) – Contact information (e.g., phone number).

b).Receivers Dataset

The receivers.csv file contains details of individuals or organizations receiving food.

- **Receiver_ID** (Integer) – Unique identifier for each receiver.
- **Name** (String) – Name of the receiver (individual or organization).
- **Type** (String) – Category of receiver (e.g., NGO, Community Center, Individual).
- **City** (String) – City where the receiver is located.
- **Contact** (String) – Contact details (e.g., phone number).

c). Food Listings Dataset

The food_listings.csv file stores details of available food items that can be claimed by receivers.

- **Food_ID** (Integer) – Unique identifier for each food item.
- **Food_Name** (String) – Name of the food item.
- **Quantity** (Integer) – Quantity available for distribution.
- **Expiry_Date** (Date) – Expiry date of the food item.
- **Provider_ID** (Integer) – Reference to the provider offering the food.
- **Provider_Type** (String) – Type of provider offering the food.
- **Location** (String) – City where the food is available.

- **Food_Type** (String) – Category of food (e.g., Vegetarian, Non-Vegetarian, Vegan).
- **Meal_Type** (String) – Type of meal (e.g., Breakfast, Lunch, Dinner, Snacks).

d). Claims Dataset

The claims.csv file tracks food claims made by receivers.

- **Claim_ID** (Integer) – Unique identifier for each claim.
- **Food_ID** (Integer) – Reference to the food item being claimed.
- **Receiver_ID** (Integer) – Reference to the receiver claiming the food.
- **Status** (String) – Current status of the claim (e.g., Pending, Completed, Cancelled).

- **Timestamp** (Datetime) – Date and time when the claim was made

8. Business Use Cases

- Restaurants list extra food items at end of the day.
- NGOs or individuals search and claim food from nearby locations.
- The platform tracks who provides the most food.
- Reports help identify high-demand locations and optimal distribution strategies.

9. Implementation Overview

Directory Structure:

```
D:/Guvi_Project1/

 │

 ├── dataset/                 # Folder for datasets

 │    ├── Images/             # Folder for images

 │    │    └── image.jpg      # Thumbnail image for the project

 │    ├── claims_data.csv     # Claims data (claims related to food)

 │    ├── Food_listings_data.csv  # Food listings data (food
available for distribution)

 │    ├── providers_data.csv    # Food providers data (details of
food providers)
```

```
│     └──── Receivers_data.csv    # Receivers data (details of the
food receivers)

│

├──── env/                      # Folder for environment and scripts

│     ├──── Scripts/                        # Folder for Python
scripts

│     │     ├──── Local_food_WM.db            # SQLite database
containing the data

│     │     ├──── food_waste_management.py    # Main script for
handling food waste management logic

│     │     ├──── homepage.py                 # Homepage script for
overview, search, and filtering datasets

│     │     ├──── Providers.py                # Script for managing
food providers (CRUD operations)

│     │     ├──── Receivers.py                # Script for managing
receivers (CRUD operations)

│     │     ├──── Food_listing_datas.py       # Script for managing
food listings (view, update, delete)

│     │     ├──── claim_status.py             # Script for handling
claims (submit, cancel, complete)

│     │     ├──── Queries.py                  # Script for data
queries, insights, and generating reports

│     │     └──── about.py                    # Script providing
information about the project and how it works
```

Core Scripts:

1) homepage.py: Overview, image display, and search functionality
2) Providers.py: Manage providers Data
3) Receivers.py: Manage receivers Data
4) Food_listing_datas.py: List and update food entries Data
5) claim_status.py: Claim status and updates
6) Queries.py: QUERY, Charts and data insights
7) about.py: Project description and purpose

10. Data Analysis & Visualization

Using Pandas, Seaborn, and Matplotlib, key insights are visualized such as:

1. Food wastage trends over time
2. Top food provider contributions
3. Most claimed locations
4. Category-based wastage insights

Example visualizations include bar charts and line plots integrated into Streamlit.

11. Key Insights & Outcomes

- Certain locations have higher food demand, guiding redistribution efforts.
- A few providers consistently contribute the most surplus food.
- Food types like “VEG”, “NON-VEG” and “VEGEN”are frequently wasted.
- Claim patterns help forecast future needs.

12. Challenges Faced

- Normalizing and cleaning CSV data for database ingestion.
- Maintaining data integrity between providers, food items, and claims.
- Designing an intuitive Streamlit UI for different user roles.
- Efficiently generating dynamic charts and handling edge cases.

13. Future Enhancements

- Add real-time notifications for food availability.
- Integrate maps to show food locations based on user geolocation.
- Enable mobile app support.
- Add expiry alerts and food freshness indicators.
- Allow bulk data import/export for NGOs and restaurants.

14. Conclusion

This project successfully demonstrates a socially impactful solution by leveraging Python and Streamlit to reduce food waste and serve underprivileged communities. The Local Food Wastage Management System showcases the practical application of data, technology, and domain knowledge to solve real-world issues.

15. Appendices

1).food_waste_management.py

```
import streamlit as st
import pandas as pd
import sqlite3


st.set_page_config(page_title="LOCAL FOOD WASTE MANAGEMENT", page_icon=":material/edit:")



home = st.Page("homepage.py", title="Homepage", icon=":material/circle:")
provider = st.Page("Providers.py", title="Providers", icon=":material/circle:")
receiver = st.Page("Receivers.py", title="Receivers", icon=":material/circle:")
food = st.Page("Food_listing_datas.py", title="Food Details", icon=":material/circle:")
status = st.Page("claim_status.py", title="Claim Status", icon=":material/circle:")
query = st.Page("Queries.py", title="Queries", icon=":material/circle:")
about = st.Page("About.py", title="About", icon=":material/circle:")


pg = st.navigation([home, provider,receiver,food,status,query,about])
pg.run()


print('✅ All Done')
```

2).homepage.py

```
import streamlit as st
import pandas as pd
import sqlite3
import os

# --- App Header ---
with st.container():
    st.title("   WELCOME!!!")
    st.title("   LOCAL FOOD WASTE MANAGEMENT SYSTEM   ")
    image_path = 'D:/Guvi_Project1/dataset/Images/supwproject-210603134356-thumbnail.jpg'
    if os.path.exists(image_path):
        st.image(image_path, use_container_width=True)
    else:
        st.warning("Image not found at the specified path.")
    st.write("Connecting Food Providers with Receivers to Reduce Waste and Feed Communities")

# --- Database Connection ---
DB_PATH = 'D:/Guvi_Project1/env/Scripts/Local_food_WM.db'
if not os.path.exists(DB_PATH):
    st.error("Database file not found. Check DB path.")
    st.stop()

conn = sqlite3.connect(DB_PATH)

# --- Table selection ---
table_options = {
    "Providers": "providers",
    "Receivers": "receivers",
    "Food Listings": "food_listings",
    "Claim Status": "claims"
}
selected_table_name = st.selectbox("   Select a Dataset to View", list(table_options.keys()))
table = table_options[selected_table_name]

st.markdown(f"###    Displaying: {selected_table_name} Details")

# --- Sidebar Filters ---
st.sidebar.header("   Apply Filters")

# Providers
if selected_table_name == "Providers":
    df = pd.read_sql("SELECT * FROM providers", conn)
    df.columns = df.columns.str.strip()
    if all(col in df.columns for col in ["Provider_ID", "City", "Type"]):
        provider_ids = df["Provider_ID"].dropna().unique().tolist()
        cities = df["City"].dropna().unique().tolist()
        types = df["Type"].dropna().unique().tolist()

        provider_id = st.sidebar.selectbox("Provider ID", ["All"] + provider_ids)
        city = st.sidebar.selectbox("City", ["All"] + cities)
        provider_type = st.sidebar.selectbox("Provider Type", ["All"] + types)

        if provider_id != "All":
            df = df[df["Provider_ID"] == provider_id]
        if city != "All":
            df = df[df["City"] == city]
        if provider_type != "All":
            df = df[df["Type"] == provider_type]
    else:
        st.warning("Some expected columns are missing in Providers table.")

# Receivers
elif selected_table_name == "Receivers":
    df = pd.read_sql("SELECT * FROM receivers", conn)
    df.columns = df.columns.str.strip()
    if all(col in df.columns for col in ["Receiver_ID", "City", "Type"]):
        receiver_ids = df["Receiver_ID"].dropna().unique().tolist()
        cities = df["City"].dropna().unique().tolist()
        receiver_types = df["Type"].dropna().unique().tolist()

        receiver_id = st.sidebar.selectbox("Receiver ID", ["All"] + receiver_ids)
        city = st.sidebar.selectbox("City", ["All"] + cities)
        receiver_type = st.sidebar.selectbox("Receiver Type", ["All"] + receiver_types)

        if receiver_id != "All":
            df = df[df["Receiver_ID"] == receiver_id]
        if city != "All":
            df = df[df["City"] == city]
        if receiver_type != "All":
            df = df[df["Type"] == receiver_type]
    else:
        st.warning("Some expected columns are missing in Receivers table.")

# Food Listings (WITH JOIN for Provider Details)
elif selected_table_name == "Food Listings":
    base_query = """
        SELECT
            f.Food_ID, f.Food_Type, f.Meal_Type, f.Quantity, f.Location,
            p.Provider_ID, p.Name AS Provider_Name, p.City AS Provider_City, p.Type AS Provider_Type
        FROM food_listings f
        LEFT JOIN providers p ON f.Provider_ID = p.Provider_ID
    """
    df = pd.read_sql(base_query, conn)
    df.columns = df.columns.str.strip()

    food_ids = df["Food_ID"].dropna().unique().tolist()
    cities = df["Location"].dropna().unique().tolist()
    food_types = df["Food_Type"].dropna().unique().tolist()
    meal_types = df["Meal_Type"].dropna().unique().tolist()

    food_id = st.sidebar.selectbox("Food ID", ["All"] + food_ids)
    city = st.sidebar.selectbox("City", ["All"] + cities)
    food_type = st.sidebar.selectbox("Food Type", ["All"] + food_types)
    meal_type = st.sidebar.selectbox("Meal Type", ["All"] + meal_types)

    if food_id != "All":
        df = df[df["Food_ID"] == food_id]
    if city != "All":
        df = df[df["Location"] == city]
    if food_type != "All":
        df = df[df["Food_Type"] == food_type]
    if meal_type != "All":
        df = df[df["Meal_Type"] == meal_type]

# Claim Status (already has JOINs)
elif selected_table_name == "Claim Status":
    claim_ids = pd.read_sql("SELECT DISTINCT Claim_ID FROM claims", conn)["Claim_ID"].tolist()
    statuses = pd.read_sql("SELECT DISTINCT Status FROM claims", conn)["Status"].tolist()

    claim_id = st.sidebar.selectbox("Claim ID", ["All"] + claim_ids)
    claim_status = st.sidebar.selectbox("Claim Status", ["All"] + statuses)

    base_query = """
        SELECT
            c.Claim_ID, c.Status, c.Timestamp,
            f.Food_ID, f.Food_Type, f.Meal_Type, f.Quantity, f.Location,
            p.Provider_ID, p.Name AS Provider_Name, p.City AS Provider_City, p.Type AS Provider_Type,
            r.Receiver_ID, r.Name AS Receiver_Name, r.City AS Receiver_City, r.Type AS Receiver_Type
        FROM claims c
        LEFT JOIN food_listings f ON c.Food_ID = f.Food_ID
        LEFT JOIN providers p ON f.Provider_ID = p.Provider_ID
        LEFT JOIN receivers r ON c.Receiver_ID = r.Receiver_ID
    """
    filters = []
    params = []

    if claim_id != "All":
        filters.append("c.Claim_ID = ?")
        params.append(claim_id)
    if claim_status != "All":
        filters.append("c.Status = ?")
        params.append(claim_status)

    if filters:
        base_query += " WHERE " + " AND ".join(filters)

    df = pd.read_sql(base_query, conn, params=params)

# --- Display Final Filtered Data ---
st.dataframe(df, use_container_width=True)
st.success(f"✅ {len(df)} records found.")
conn.close()
```

3).Providers.py

```
import streamlit as st
import pandas as pd
import sqlite3
import os

# --- App Header ---
with st.container():
    st.title("   WELCOME!!!")
    st.title("   LOCAL FOOD WASTE MANAGEMENT SYSTEM   ")
    image_path = 'D:/Guvi_Project1/dataset/Images/supwproject-210603134356-thumbnail.jpg'
    if os.path.exists(image_path):
        st.image(image_path, use_container_width=True)
    else:
        st.warning("Image not found at the specified path.")
    st.write("Connecting Food Providers with Receivers to Reduce Waste and Feed Communities")

# --- Database Connection ---
DB_PATH = 'D:/Guvi_Project1/env/Scripts/Local_food_WM.db'
if not os.path.exists(DB_PATH):
    st.error("Database file not found. Check DB path.")
    st.stop()

conn = sqlite3.connect(DB_PATH)

# --- Table selection ---
table_options = {
    "Providers": "providers",
    "Receivers": "receivers",
    "Food Listings": "food_listings",
    "Claim Status": "claims"
}
selected_table_name = st.selectbox("   Select a Dataset to View", list(table_options.keys()))
table = table_options[selected_table_name]

st.markdown(f"###    Displaying: {selected_table_name} Details")

# --- Sidebar Filters ---
st.sidebar.header("   Apply Filters")

# Providers
if selected_table_name == "Providers":
    df = pd.read_sql("SELECT * FROM providers", conn)
    df.columns = df.columns.str.strip()
    if all(col in df.columns for col in ["Provider_ID", "City", "Type"]):
        provider_ids = df["Provider_ID"].dropna().unique().tolist()
        cities = df["City"].dropna().unique().tolist()
        types = df["Type"].dropna().unique().tolist()

        provider_id = st.sidebar.selectbox("Provider ID", ["All"] + provider_ids)
        city = st.sidebar.selectbox("City", ["All"] + cities)
        provider_type = st.sidebar.selectbox("Provider Type", ["All"] + types)

        if provider_id != "All":
            df = df[df["Provider_ID"] == provider_id]
        if city != "All":
            df = df[df["City"] == city]
        if provider_type != "All":
            df = df[df["Type"] == provider_type]
    else:
        st.warning("Some expected columns are missing in Providers table.")

# Receivers
elif selected_table_name == "Receivers":
    df = pd.read_sql("SELECT * FROM receivers", conn)
    df.columns = df.columns.str.strip()
    if all(col in df.columns for col in ["Receiver_ID", "City", "Type"]):
        receiver_ids = df["Receiver_ID"].dropna().unique().tolist()
        cities = df["City"].dropna().unique().tolist()
        receiver_types = df["Type"].dropna().unique().tolist()

        receiver_id = st.sidebar.selectbox("Receiver ID", ["All"] + receiver_ids)
        city = st.sidebar.selectbox("City", ["All"] + cities)
        receiver_type = st.sidebar.selectbox("Receiver Type", ["All"] + receiver_types)

        if receiver_id != "All":
            df = df[df["Receiver_ID"] == receiver_id]
        if city != "All":
            df = df[df["City"] == city]
        if receiver_type != "All":
            df = df[df["Type"] == receiver_type]
    else:
        st.warning("Some expected columns are missing in Receivers table.")
```

```
# Food Listings (WITH JOIN for Provider Details)
elif selected_table_name == "Food Listings":
    base_query = """
        SELECT
            f.Food_ID, f.Food_Type, f.Meal_Type, f.Quantity, f.Location,
            p.Provider_ID, p.Name AS Provider_Name, p.City AS Provider_City, p.Type AS Provider_Type
        FROM food_listings f
        LEFT JOIN providers p ON f.Provider_ID = p.Provider_ID
    """
    df = pd.read_sql(base_query, conn)
    df.columns = df.columns.str.strip()

    food_ids = df["Food_ID"].dropna().unique().tolist()
    cities = df["Location"].dropna().unique().tolist()
    food_types = df["Food_Type"].dropna().unique().tolist()
    meal_types = df["Meal_Type"].dropna().unique().tolist()

    food_id = st.sidebar.selectbox("Food ID", ["All"] + food_ids)
    city = st.sidebar.selectbox("City", ["All"] + cities)
    food_type = st.sidebar.selectbox("Food Type", ["All"] + food_types)
    meal_type = st.sidebar.selectbox("Meal Type", ["All"] + meal_types)

    if food_id != "All":
        df = df[df["Food_ID"] == food_id]
    if city != "All":
        df = df[df["Location"] == city]
    if food_type != "All":
        df = df[df["Food_Type"] == food_type]
    if meal_type != "All":
        df = df[df["Meal_Type"] == meal_type]

# Claim Status (already has JOINs)
elif selected_table_name == "Claim Status":
    claim_ids = pd.read_sql("SELECT DISTINCT Claim_ID FROM claims", conn)["Claim_ID"].tolist()
    statuses = pd.read_sql("SELECT DISTINCT Status FROM claims", conn)["Status"].tolist()

    claim_id = st.sidebar.selectbox("Claim ID", ["All"] + claim_ids)
    claim_status = st.sidebar.selectbox("Claim Status", ["All"] + statuses)

    base_query = """
        SELECT
            c.Claim_ID, c.Status, c.Timestamp,
            f.Food_ID, f.Food_Type, f.Meal_Type, f.Quantity, f.Location,
            p.Provider_ID, p.Name AS Provider_Name, p.City AS Provider_City, p.Type AS Provider_Type,
            r.Receiver_ID, r.Name AS Receiver_Name, r.City AS Receiver_City, r.Type AS Receiver_Type
        FROM claims c
        LEFT JOIN food_listings f ON c.Food_ID = f.Food_ID
        LEFT JOIN providers p ON f.Provider_ID = p.Provider_ID
        LEFT JOIN receivers r ON c.Receiver_ID = r.Receiver_ID
    """
    filters = []
    params = []

    if claim_id != "All":
        filters.append("c.Claim_ID = ?")
        params.append(claim_id)
    if claim_status != "All":
        filters.append("c.Status = ?")
        params.append(claim_status)

    if filters:
        base_query += " WHERE " + " AND ".join(filters)

    df = pd.read_sql(base_query, conn, params=params)

# --- Display Final Filtered Data ---
st.dataframe(df, use_container_width=True)
st.success(f"✅ {len(df)} records found.")
conn.close()
```

4).Receivers.py

```
import streamlit as st
import pandas as pd
import sqlite3
import os

st.header('   Receivers     ')

# Paths
DB_PATH = 'D:/Guvi_Project1/env/Scripts/Local_food_WM.db'
CSV_PATH = 'D:/Guvi_Project1/dataset/Receivers_data.csv'

st.subheader("   All Registered Receivers Information")
receiver_df = pd.read_csv(CSV_PATH)
st.dataframe(receiver_df)

# Initialize SQLite database and import from CSV
def initialize_db():
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS receivers (
            Receiver_ID INTEGER PRIMARY KEY AUTOINCREMENT,
            Name TEXT,
            Type TEXT,
            City TEXT,
            Contact TEXT UNIQUE
        )
    ''')
    conn.commit()

    cursor.execute("SELECT COUNT(*) FROM receivers")
    count = cursor.fetchone()[0]

    if count == 0 and os.path.exists(CSV_PATH):
        df = pd.read_csv(CSV_PATH)
        df.to_sql('receivers', conn, if_exists='append', index=False)

    conn.close()

# Get next receiver ID
def get_next_receiver_id():
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
    cursor.execute("SELECT seq FROM sqlite_sequence WHERE name='receivers'")
    row = cursor.fetchone()
    conn.close()
    return (row[0] + 1) if row else 1

# Insert a new receiver into DB
def insert_receiver(name, rtype, city, contact):
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
    cursor.execute('''
        INSERT INTO receivers (Name, Type, City, Contact)
        VALUES (?, ?, ?, ?)
    ''', (name, rtype, city, contact))
    conn.commit()
    new_id = cursor.lastrowid
    conn.close()
    return new_id

# Update Receiver
def update_receiver(rid, name, rtype, city, contact):
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
    cursor.execute('''
        UPDATE receivers
        SET Name=?, Type=?, City=?, Contact=?
        WHERE Receiver_ID=?
    ''', (name, rtype, city, contact, rid))
    conn.commit()
    conn.close()

    df = pd.read_csv(CSV_PATH)
    df.loc[df['Receiver_ID'] == rid, ['Name', 'Type', 'City', 'Contact']] = [name, rtype, city, contact]
    df.to_csv(CSV_PATH, index=False)

# Delete Receiver
def delete_receiver(rid):
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
    cursor.execute("DELETE FROM receivers WHERE Receiver_ID=?", (rid,))
    conn.commit()
    conn.close()

    df = pd.read_csv(CSV_PATH)
    df = df[df['Receiver_ID'] != rid]
    df.to_csv(CSV_PATH, index=False)

# Append to CSV file
def append_to_csv(receiver_id, name, rtype, city, contact):
    new_row = pd.DataFrame([{
        'Receiver_ID': receiver_id,
        'Name': name,
        'Type': rtype,
        'City': city,
        'Contact': contact
    }])
    if os.path.exists(CSV_PATH):
        new_row.to_csv(CSV_PATH, mode='a', header=False, index=False)
    else:
        new_row.to_csv(CSV_PATH, mode='w', index=False)

# Get the latest inserted receiver
def get_latest_receiver():
    conn = sqlite3.connect(DB_PATH)
    df = pd.read_sql_query("SELECT * FROM receivers ORDER BY Receiver_ID DESC LIMIT 1", conn)
    conn.close()
    return df

# Initialize database
initialize_db()

# Streamlit App
st.markdown("<h3 style='text-align: center;'>   Receiver Registration Form</h3>",
unsafe_allow_html=True)

with st.form("receiver_form"):
    receiver_id = get_next_receiver_id()
    st.text_input("Receiver ID", value=str(receiver_id), disabled=True)

    name = st.text_input("Name")
    rtype = st.selectbox("Receiver Type", ["Individual", "Charity", "NGO", "Shelter"])
    city = st.text_input("City")
    contact = st.text_input("Contact (must be unique)")

    submitted = st.form_submit_button("Register")
    st.subheader("Recently Registered Receiver")
    st.dataframe(get_latest_receiver(), use_container_width=True)


    if submitted:
        if name and city and contact:
            try:
                new_id = insert_receiver(name, rtype, city, contact)
                append_to_csv(new_id, name, rtype, city, contact)
                st.success(f"Receiver registered successfully with ID {new_id}")
                st.balloons()
                st.subheader("   Just Registered Register")
                registered_data = pd.DataFrame([{
                    'Receiver_ID': receiver_id,
                    'Name': name,
                    'Type': rtype,
                    'City': city,
                    'Contact': contact
                }])
                st.dataframe(registered_data, use_container_width=True)




            except sqlite3.IntegrityError:
                st.error("Contact must be unique. This receiver is already registered.")
        else:
            st.error("All fields except 'Type' are required.")
```

```
# -------------------- UPDATE & DELETE --------------------
st.markdown("---")
st.markdown("<h3 style='text-align: center;'>   Update or    Delete Receiver</h3>",
unsafe_allow_html=True)

receiver_ids = receiver_df['Receiver_ID'].tolist()
selected_id = st.selectbox("Select Receiver ID", receiver_ids)

if selected_id:
    selected_row = receiver_df[receiver_df['Receiver_ID'] == selected_id].iloc[0]

    with st.form("update_delete_form"):
        name_upd = st.text_input("Name", selected_row['Name'])
        type_upd = st.selectbox("Receiver Type", ["Individual", "Charity", "NGO", "Shelter"],
                                index=["Individual", "Charity", "NGO", "Shelter"].index(selected_row['Type']))
        city_upd = st.text_input("City", selected_row['City'])
        contact_upd = st.text_input("Contact", selected_row['Contact'])

        col1, col2 = st.columns(2)
        with col1:
            updated = st.form_submit_button("Update")
        with col2:
            deleted = st.form_submit_button("Delete")

        if updated:
            try:
                update_receiver(selected_id, name_upd, type_upd, city_upd, contact_upd)
                st.success("✅ Receiver updated successfully!")
            except sqlite3.IntegrityError:
                st.error("   Contact must be unique. Update failed.")

        if deleted:
            delete_receiver(selected_id)
            st.warning(f"   Receiver with ID {selected_id} has been deleted.")
```

5).Food_listing_datas.py

```
import streamlit as st
import pandas as pd
import sqlite3
import os
from datetime import date

st.header('     Food Listings')

# Paths
DB_PATH = 'D:/Guvi_Project1/env/Scripts/Local_food_WM.db'
CSV_PATH = 'D:/Guvi_Project1/dataset/Food_listings_data.csv'

# Show existing listings
st.subheader("   All Listed Food Items")
if os.path.exists(CSV_PATH):
    food_df = pd.read_csv(CSV_PATH)
else:
    food_df = pd.DataFrame(columns=[
        "Food_ID", "Food_Name", "Quantity", "Expiry_Date",
        "Provider_ID", "Provider_Type", "Location", "Food_Type", "Meal_Type"
    ])
st.dataframe(food_df, use_container_width=True)

# Initialize database
def initialize_db():
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS food_listings (
            Food_ID INTEGER PRIMARY KEY AUTOINCREMENT,
            Food_Name TEXT,
            Quantity INTEGER,
            Expiry_Date TEXT,
            Provider_ID INTEGER,
            Provider_Type TEXT,
            Location TEXT,
            Food_Type TEXT,
            Meal_Type TEXT
        )
    ''')
    conn.commit()

    # Load from CSV if table is empty
    cursor.execute("SELECT COUNT(*) FROM food_listings")
    count = cursor.fetchone()[0]
    if count == 0 and os.path.exists(CSV_PATH):
        df = pd.read_csv(CSV_PATH)
        df.to_sql('food_listings', conn, if_exists='append', index=False)
    conn.close()

# Get next auto-increment ID
def get_next_food_id():
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
    cursor.execute("SELECT seq FROM sqlite_sequence WHERE name='food_listings'")
    row = cursor.fetchone()
    conn.close()
    return (row[0] + 1) if row else 1

# Insert new food listing
def insert_food(name, qty, exp, pid, ptype, loc, ftype, meal):
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
    cursor.execute('''
        INSERT INTO food_listings (Food_Name, Quantity, Expiry_Date, Provider_ID, Provider_Type,
Location, Food_Type, Meal_Type)
        VALUES (?, ?, ?, ?, ?, ?, ?, ?)
    ''', (name, qty, exp, pid, ptype, loc, ftype, meal))
    conn.commit()
    new_id = cursor.lastrowid
    conn.close()
    return new_id

# Append to CSV
def append_to_csv(new_row):
    if os.path.exists(CSV_PATH):
        new_row.to_csv(CSV_PATH, mode='a', header=False, index=False)
    else:
        new_row.to_csv(CSV_PATH, mode='w', index=False)

# Get latest inserted record
def get_latest_food():
    conn = sqlite3.connect(DB_PATH)
    df = pd.read_sql_query("SELECT * FROM food_listings ORDER BY Food_ID DESC LIMIT 1", conn)
    conn.close()
    return df

# Update record
def update_food(fid, name, qty, exp, ftype, meal):
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
    cursor.execute('''
        UPDATE food_listings
        SET Food_Name=?, Quantity=?, Expiry_Date=?, Food_Type=?, Meal_Type=?
        WHERE Food_ID=?
    ''', (name, qty, exp, ftype, meal, fid))
    conn.commit()
    conn.close()

    df = pd.read_csv(CSV_PATH)
    df.loc[df['Food_ID'] == fid, ['Food_Name', 'Quantity', 'Expiry_Date', 'Food_Type', 'Meal_Type']] =
[name, qty, exp, ftype, meal]
    df.to_csv(CSV_PATH, index=False)

# Delete record
def delete_food(fid):
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
    cursor.execute("DELETE FROM food_listings WHERE Food_ID=?", (fid,))
    conn.commit()
    conn.close()

    df = pd.read_csv(CSV_PATH)
    df = df[df['Food_ID'] != fid]
    df.to_csv(CSV_PATH, index=False)

# Initialize database
initialize_db()

# Registration form
st.markdown("<h3 style='text-align: center;'>   List Surplus Food</h3>", unsafe_allow_html=True)

# Get provider options from DB
with sqlite3.connect(DB_PATH) as conn:
    providers = pd.read_sql_query("SELECT DISTINCT Provider_ID, Type, City FROM providers", conn)

provider_id = st.selectbox("Provider ID", providers["Provider_ID"].unique())
provider_type = providers.loc[providers["Provider_ID"] == provider_id, "Type"].values[0]
location = providers.loc[providers["Provider_ID"] == provider_id, "City"].values[0]

with st.form("listing_form"):
    food_id = get_next_food_id()
    st.text_input("Food ID", value=str(food_id), disabled=True)

    name = st.text_input("Food Name")
    qty = st.number_input("Quantity", min_value=1, step=1)
    exp_date = st.date_input("Expiry Date", min_value=date.today())
    ftype = st.selectbox("Food Type", ["Vegetarian", "Non-Vegetarian", "Vegan"])
    meal_type = st.selectbox("Meal Type", ["Breakfast", "Lunch", "Dinner", "Snacks"])
    submit = st.form_submit_button("List Food")
    st.subheader("Recently Provider Listing The Surplus Food ")
    st.dataframe(get_latest_food(), use_container_width=True)

    if submit:
        if name and qty and exp_date:
            new_id = insert_food(name, qty, exp_date.strftime("%Y-%m-%d"), provider_id, provider_type,
location, ftype, meal_type)
            new_row = pd.DataFrame([{
                'Food_ID': new_id,
                'Food_Name': name,
                'Quantity': qty,
                'Expiry_Date': exp_date.strftime("%Y-%m-%d"),
                'Provider_ID': provider_id,
                'Provider_Type': provider_type,
                'Location': location,
                'Food_Type': ftype,
                'Meal_Type': meal_type
            }])
            append_to_csv(new_row)
            st.success(f"✅ Food item listed with ID {new_id}")
            st.balloons()
            st.dataframe(new_row, use_container_width=True)
        else:
            st.error("Please fill in all fields.")

# Update/Delete section
st.markdown("---")
st.markdown("<h3 style='text-align: center;'>   Update or    Delete Food Listing</h3>",
unsafe_allow_html=True)

food_ids = food_df['Food_ID'].tolist()
selected_id = st.selectbox("Select Food ID", food_ids)

if selected_id:
    selected_row = food_df[food_df['Food_ID'] == selected_id].iloc[0]

    with st.form("update_delete_form"):
        name_upd = st.text_input("Food Name", selected_row['Food_Name'])
        qty_upd = st.number_input("Quantity", value=int(selected_row['Quantity']), step=1)
        exp_upd = st.date_input("Expiry Date", pd.to_datetime(selected_row['Expiry_Date']))
        ftype_upd = st.selectbox("Food Type", ["Vegetarian", "Non-Vegetarian", "Vegan"],
index=["Vegetarian", "Non-Vegetarian", "Vegan"].index(selected_row["Food_Type"]))
        meal_upd = st.selectbox("Meal Type", ["Breakfast", "Lunch", "Dinner", "Snacks"],
index=["Breakfast", "Lunch", "Dinner", "Snacks"].index(selected_row["Meal_Type"]))

        col1, col2 = st.columns(2)
        with col1:
            updated = st.form_submit_button("   Update")
        with col2:
            deleted = st.form_submit_button("   Delete")

        if updated:
            update_food(selected_id, name_upd, qty_upd, exp_upd.strftime("%Y-%m-%d"), ftype_upd,
meal_upd)
            st.success("✅ Food listing updated.")

        if deleted:
            delete_food(selected_id)
            st.warning(f"   Food listing with ID {selected_id} has been deleted.")
```

6). claim_status.py

```
import streamlit as st
import pandas as pd
import sqlite3
import os
from datetime import datetime

# Paths
DB_PATH = 'D:/Guvi_Project1/env/Scripts/Local_food_WM.db'
CSV_PATH = 'D:/Guvi_Project1/dataset/claims_data.csv'


st.header('  Claim Status⌛ ')

clm_sts = pd.read_csv(CSV_PATH )
st.dataframe(clm_sts)




# Load claims data
if os.path.exists(CSV_PATH):
    claims_df = pd.read_csv(CSV_PATH)
else:
    claims_df = pd.DataFrame(columns=["Claim_ID", "Food_ID", "Receiver_ID", "Status", "Timestamp"])



# Initialize database
def initialize_db():
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS claims (
            Claim_ID INTEGER PRIMARY KEY AUTOINCREMENT,
            Food_ID INTEGER,
            Receiver_ID INTEGER,
            Status TEXT,
            Timestamp TEXT
        )
    ''')
    conn.commit()
    if cursor.execute("SELECT COUNT(*) FROM claims").fetchone()[0] == 0 and
os.path.exists(CSV_PATH):
        df = pd.read_csv(CSV_PATH)
        df.to_sql('claims', conn, if_exists='append', index=False)
    conn.close()

# Get next Claim ID
def get_next_claim_id():
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
    cursor.execute("SELECT seq FROM sqlite_sequence WHERE name='claims'")
    row = cursor.fetchone()
    conn.close()
    return (row[0] + 1) if row else 1

# Insert claim into DB
def insert_claim(food_id, receiver_id):
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    cursor.execute('''
        INSERT INTO claims (Food_ID, Receiver_ID, Status, Timestamp)
        VALUES (?, ?, 'Pending', ?)
    ''', (food_id, receiver_id, timestamp))
    conn.commit()
    new_id = cursor.lastrowid
    conn.close()
    return new_id, timestamp

# Completed a claim
def Completed_claim(claim_id):
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
    cursor.execute("UPDATE claims SET Status='Completed' WHERE Claim_ID=?", (claim_id,))
    conn.commit()
    conn.close()

    # Also update CSV
    if os.path.exists(CSV_PATH):
        df = pd.read_csv(CSV_PATH)
        df.loc[df['Claim_ID'] == claim_id, 'Status'] = 'Completed'
        df.to_csv(CSV_PATH, index=False)


# Cancel a claim
def cancel_claim(claim_id):
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
    cursor.execute("UPDATE claims SET Status='Cancelled' WHERE Claim_ID=?", (claim_id,))
    conn.commit()
    conn.close()

    # Also update CSV
    if os.path.exists(CSV_PATH):
        df = pd.read_csv(CSV_PATH)
        df.loc[df['Claim_ID'] == claim_id, 'Status'] = 'Cancelled'
        df.to_csv(CSV_PATH, index=False)
```

```
# Append to CSV
def append_to_csv(new_row):
    if os.path.exists(CSV_PATH):
        new_row.to_csv(CSV_PATH, mode='a', header=False, index=False)
    else:
        new_row.to_csv(CSV_PATH, mode='w', index=False)

# Get latest claim
def get_latest_claim():
    conn = sqlite3.connect(DB_PATH)
    df = pd.read_sql_query("SELECT * FROM claims ORDER BY Claim_ID DESC LIMIT 1", conn)
    conn.close()
    return df

# Initialize DB
initialize_db()

# Get available Food_IDs and Receiver_IDs
with sqlite3.connect(DB_PATH) as conn:
    food_ids = pd.read_sql_query("SELECT Food_ID FROM food_listings", conn)['Food_ID'].tolist()
    receiver_ids = pd.read_sql_query("SELECT Receiver_ID FROM receivers", conn)['Receiver_ID'].tolist()

# Claim Form
st.markdown("<h3 style='text-align: center;'>   Claim Food</h3>", unsafe_allow_html=True)

with st.form("claim_form"):
    claim_id = get_next_claim_id()
    st.text_input("Claim ID", value=str(claim_id), disabled=True)
    selected_food_id = st.selectbox("Food ID", food_ids)
    selected_receiver_id = st.selectbox("Receiver ID", receiver_ids)
    submit = st.form_submit_button("   Submit Claim")

    if submit:
        new_id, timestamp = insert_claim(selected_food_id, selected_receiver_id)
        new_row = pd.DataFrame([{
            "Claim_ID": new_id,
            "Food_ID": selected_food_id,
            "Receiver_ID": selected_receiver_id,
            "Status": "Pending",
            "Timestamp": timestamp
        }])
        append_to_csv(new_row)
        st.success(f"✅ Claim submitted with ID {new_id}")
        st.dataframe(new_row, use_container_width=True)

# Recently submitted claim
st.subheader("   Recently Submitted Claim")
st.dataframe(get_latest_claim(), use_container_width=True)

st.markdown("---")

# Complete and Cancel claim section
st.markdown("<h3 style='text-align: center;'>✅ Complete Claim or ❌ Cancel a Claim</h3>",
unsafe_allow_html=True)

pending_claims = clm_sts[clm_sts['Status'] == 'Pending']
```

```
if not pending_claims.empty:
    selected_claim_id = st.selectbox("Select a Pending Claim ID", pending_claims['Claim_ID'].tolist(),
key="action_select")

    col1, col2 = st.columns(2)
    with col1:
        Complete = st.button("✅ Complete Claim", key="complete_button")
    with col2:
        Cancel = st.button("❌ Cancel Claim", key="cancel_button")

    if Complete:
        Completed_claim(selected_claim_id)
        st.success(f"✅ Claim ID {selected_claim_id} has been marked as Completed.")

    if Cancel:
        cancel_claim(selected_claim_id)
        st.warning(f"   Claim ID {selected_claim_id} has been Cancelled.")
else:
    st.info("No pending claims available to cancel.")
```

7). Queries. py

```
import streamlit as st
import sqlite3
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

st.title("   Food Waste Management - Queries")

# Connect to DB
DB_PATH = 'D:\Guvi_Project1\env\Scripts\Local_food_WM.db'
conn = sqlite3.connect(DB_PATH)

# -------------------------------
# Query Map with Descriptions
# -------------------------------
query_map = {
    # --- Providers & Receivers ---
    "1. How many food providers and receivers are there in each city?": """
        SELECT
            p.City,
            COUNT(DISTINCT p.Provider_ID) AS Providers,
            COUNT(DISTINCT r.Receiver_ID) AS Receivers
        FROM providers p
        LEFT JOIN receivers r ON p.City = r.City
        GROUP BY p.City
    """,

    "2. Which type of food provider contributes the most food?": """
        SELECT Provider_Type, COUNT(*) AS Total_Food_Items
        FROM food_listings
        GROUP BY Provider_Type
        ORDER BY Total_Food_Items DESC
        LIMIT 1
    """,

    "3. What is the contact information of food providers in a specific city?": """
        SELECT Name, Type, Address, Contact
        FROM providers
        WHERE City = ?
    """,

    "4. Which receivers have claimed the most food?": """
        SELECT r.Name, COUNT(c.Claim_ID) AS Claims
        FROM claims c
        JOIN receivers r ON c.Receiver_ID = r.Receiver_ID
        GROUP BY r.Name
        ORDER BY Claims DESC
        LIMIT 5
    """,

    # --- Food Listings ---
    "5. What is the total quantity of food available from all providers?": """
        SELECT SUM(Quantity) AS Total_Quantity FROM food_listings
    """,

    "6. Which city has the highest number of food listings?": """
        SELECT City, COUNT(*) AS Listings
        FROM food_listings f
        JOIN providers p ON f.Provider_ID = p.Provider_ID
        GROUP BY City
        ORDER BY Listings DESC
        LIMIT 1
    """,

    "7. What are the most commonly available food types?": """
        SELECT Food_Type, COUNT(*) AS Count
        FROM food_listings
        GROUP BY Food_Type
        ORDER BY Count DESC
        LIMIT 5
    """,

    # --- Claims & Distribution ---
    "8. How many food claims have been made for each food item?": """
        SELECT f.Food_Name, COUNT(c.Claim_ID) AS Total_Claims
        FROM claims c
        JOIN food_listings f ON c.Food_ID = f.Food_ID
        GROUP BY f.Food_Name
        ORDER BY Total_Claims DESC
    """,

    "9. Which provider has had the highest number of successful food claims?": """
        SELECT p.Name, COUNT(*) AS Successful_Claims
        FROM claims c
        JOIN food_listings f ON c.Food_ID = f.Food_ID
        JOIN providers p ON f.Provider_ID = p.Provider_ID
        WHERE c.Status = 'Completed'
        GROUP BY p.Name
        ORDER BY Successful_Claims DESC
        LIMIT 1
    """,

    "10. What percentage of food claims are completed vs. pending vs. canceled?": """
        SELECT Status,
               ROUND(COUNT(*) * 100.0 / (SELECT COUNT(*) FROM claims), 2) AS Percentage
        FROM claims
        GROUP BY Status
    """,

    # --- Insights & Analysis ---
    "11. What is the average quantity of food claimed per receiver?": """
        SELECT r.Name, ROUND(AVG(f.Quantity), 2) AS Avg_Quantity
        FROM claims c
        JOIN food_listings f ON c.Food_ID = f.Food_ID
        JOIN receivers r ON c.Receiver_ID = r.Receiver_ID
        GROUP BY r.Name
        ORDER BY Avg_Quantity DESC
        LIMIT 10
    """,

    "12. Which meal type is claimed the most?": """
        SELECT Meal_Type, COUNT(*) AS Claim_Count
        FROM food_listings f
        JOIN claims c ON f.Food_ID = c.Food_ID
        GROUP BY Meal_Type
        ORDER BY Claim_Count DESC
        LIMIT 1
    """,

    "13. What is the total quantity of food donated by each provider?": """
        SELECT p.Name, SUM(f.Quantity) AS Total_Donated
        FROM food_listings f
        JOIN providers p ON f.Provider_ID = p.Provider_ID
        GROUP BY p.Name
        ORDER BY Total_Donated DESC
        LIMIT 10""",
        # --- Operational / Time-based ---
    "14. What is the average time between food listing and claim?": """
        SELECT AVG(JULIANDAY(c.Timestamp) - JULIANDAY(f.Expiry_Date)) AS Avg_Days_Before_Expiry
        FROM claims c
        JOIN food_listings f ON c.Food_ID = f.Food_ID
    """,

    "15. How many expired food items are still unclaimed?": """
        SELECT COUNT(*) AS Expired_Unclaimed
        FROM food_listings f
        LEFT JOIN claims c ON f.Food_ID = c.Food_ID
        WHERE f.Expiry_Date < DATE('now') AND c.Claim_ID IS NULL
    """,

    "16. What is the average quantity of food provided by each type of provider?": """
        SELECT Provider_Type, ROUND(AVG(Quantity), 2) AS Avg_Quantity
        FROM food_listings
        GROUP BY Provider_Type
    """,
    "17. Which city has the highest amount of unclaimed food?": """
        SELECT p.City, SUM(f.Quantity) AS Unclaimed_Quantity
        FROM food_listings f
        JOIN providers p ON f.Provider_ID = p.Provider_ID
        LEFT JOIN claims c ON f.Food_ID = c.Food_ID
        WHERE c.Claim_ID IS NULL
        GROUP BY p.City
        ORDER BY Unclaimed_Quantity DESC
        LIMIT 1
    """,

    "18. List providers who haven't had any claims": """
        SELECT DISTINCT p.Name, p.City
        FROM providers p
        LEFT JOIN food_listings f ON p.Provider_ID = f.Provider_ID
        LEFT JOIN claims c ON f.Food_ID = c.Food_ID
        WHERE c.Claim_ID IS NULL
    """,

    "19. Which food types are expiring soon (next 3 days)?": """
        SELECT Food_Name, Expiry_Date, Quantity
        FROM food_listings
        WHERE DATE(Expiry_Date) <= DATE('now', '+3 days')
        ORDER BY Expiry_Date
    """,

    "20. Monthly trend of food donations": """
        SELECT strftime('%Y-%m', Expiry_Date) AS Month, SUM(Quantity) AS Total_Donated
        FROM food_listings
        GROUP BY Month
        ORDER BY Month DESC
    """,

    "21. Top 5 most donated food items": """
        SELECT Food_Name, SUM(Quantity) AS Total_Quantity
        FROM food_listings
        GROUP BY Food_Name
        ORDER BY Total_Quantity DESC
        LIMIT 5
    """,

    "22. Number of unique receivers per city": """
        SELECT City, COUNT(DISTINCT Receiver_ID) AS Unique_Receivers
        FROM receivers
        GROUP BY City
    """,

    "23. What is the total number of canceled claims per receiver?": """
        SELECT r.Name, COUNT(*) AS Canceled_Claims
        FROM claims c
        JOIN receivers r ON c.Receiver_ID = r.Receiver_ID
        WHERE c.Status = 'Canceled'
        GROUP BY r.Name
        ORDER BY Canceled_Claims DESC
        LIMIT 5
    """,
    "24. Average food quantity listed per provider per month": """
        SELECT p.Name, strftime('%Y-%m', f.Expiry_Date) AS Month, ROUND(AVG(f.Quantity), 2) AS
Avg_Quantity
        FROM food_listings f
        JOIN providers p ON f.Provider_ID = p.Provider_ID
        GROUP BY p.Name, Month
        ORDER BY Month DESC
    """,

    "25. Which day of the week has the most food donations?": """
        SELECT strftime('%w', Expiry_Date) AS Weekday, COUNT(*) AS Listings
        FROM food_listings
        GROUP BY Weekday
        ORDER BY Listings DESC

    """
}
# -------------------------------
# Dropdown Selection
# -------------------------------
selected_query = st.selectbox("   Select a query:", list(query_map.keys()))

# Get dynamic list of cities for filtering
provider_cities = pd.read_sql("SELECT DISTINCT City FROM providers", conn)['City'].tolist()
receiver_cities = pd.read_sql("SELECT DISTINCT City FROM receivers", conn)['City'].tolist()
all_cities = sorted(set(provider_cities + receiver_cities))


# Special handling for city input (query 3)
if selected_query == "3. What is the contact information of food providers in a specific city?":
     selected_city = st.selectbox("   select a city:", all_cities)
     city_input = selected_city

# Execute query
if st.button("Run Query"):
    try:
        query = query_map[selected_query]
        if selected_query == "3. What is the contact information of food providers in a specific city?":
            df = pd.read_sql_query(query, conn, params=(city_input,))
        else:
            df = pd.read_sql_query(query, conn)
        st.dataframe(df)
    except Exception as e:
        st.error(f"Error running query: {e}")

st.title("   Food Waste Management - Data Analysis & Chart")


# --- 1. Food Wastage by Category and Location ---
st.header("1   Food Wastage Trends by Category & Location")
query1 = """
    SELECT Food_Type, Location, COUNT(*) as Total_Wasted
    FROM food_listings
    GROUP BY Food_Type, Location
"""
df1 = pd.read_sql(query1, conn)
```

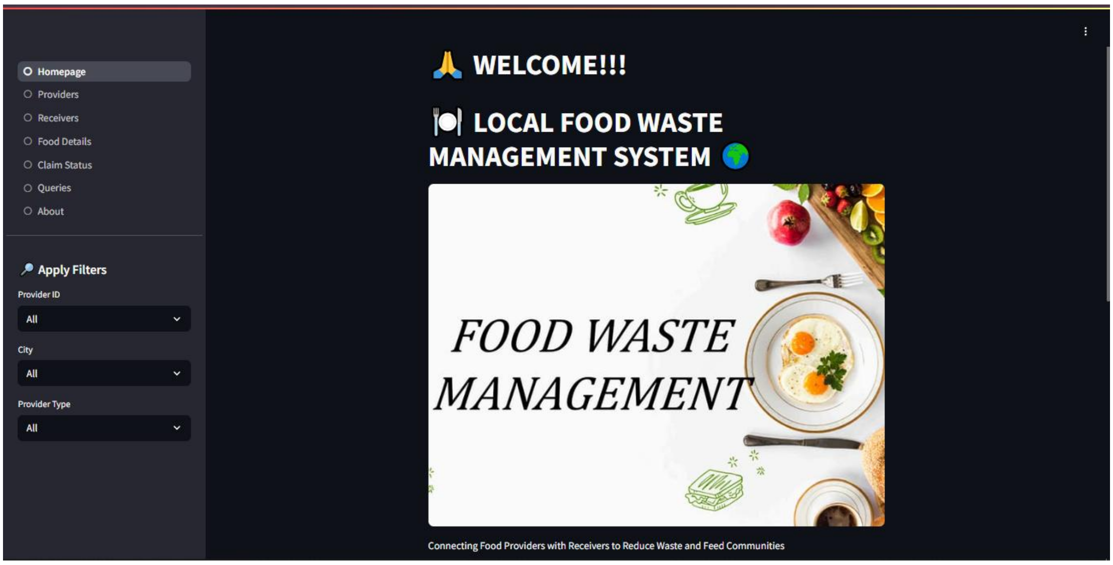
```
st.dataframe(df1)


# --- 2. Most Frequent Food Providers and Their Contributions ---
st.header("2   Top Food Providers by Contributions")
query2 = """
   SELECT p.Name AS Provider_Name, COUNT(f.Food_ID) AS Contributions
   FROM food_listings f
   JOIN providers p ON f.Provider_ID = p.Provider_ID
   GROUP BY p.Name
   ORDER BY Contributions DESC
   LIMIT 10
"""
df2 = pd.read_sql(query2, conn)
st.dataframe(df2)

fig2, ax2 = plt.subplots(figsize=(10, 4))
sns.barplot(data=df2, x='Provider_Name', y='Contributions', ax=ax2)
plt.xticks(rotation=45)
st.pyplot(fig2)

# --- 3. Highest Demand Locations Based on Food Claims ---
st.header("3   High-Demand Locations by Food Claims")
query3 = """
   SELECT f.Location, COUNT(c.Claim_ID) AS Claim_Count
   FROM claims c
   JOIN food_listings f ON c.Food_ID = f.Food_ID
   GROUP BY f.Location
   ORDER BY Claim_Count DESC
   LIMIT 10
"""
df3 = pd.read_sql(query3, conn)
st.dataframe(df3)

fig3, ax3 = plt.subplots()
sns.barplot(data=df3, x='Location', y='Claim_Count', ax=ax3)
plt.xticks(rotation=45)
st.pyplot(fig3)

# --- 4. Food Wastage Over Time (by Expiry Date) ---
st.header("4   Wastage Trend Over Time")
query4 = """
   SELECT DATE(Expiry_Date) AS Date, COUNT(*) AS Wasted_Food_Count
   FROM food_listings
   GROUP BY Date
   ORDER BY Date
"""
df4 = pd.read_sql(query4, conn)
st.line_chart(df4.set_index('Date'))

# --- Optional: Download Report ---
st.markdown("###    Download Report")
csv = df1.to_csv(index=False).encode()
st.download_button("Download Report", csv, "wastage_by_category.csv", "text/csv")

conn.close()
```

8). About. py

```
import streamlit as st
import pandas as pd


st.markdown("<h3 style='text-align: center;'>  LOCAL FOOD WASTE MANAGEMENT SYSTEM  </h3>",
unsafe_allow_html=True)

st.subheader('Introduction')
st.write('Local Food Waste Management is the process of collecting, reducing, repurposing, and
properly disposing of food waste within a specific community or region to minimize environmental
impact, reduce hunger, and promote sustainable living practices.' \
'It involves actions like food donation, composting, awareness programs, and efficient waste
collection systems operated at the community, municipal, or neighborhood level.')
st.write('Every day, food goes to waste while many go hungry. This app bridges that gap by helping
local food providers donate excess food to receivers such as NGOs, shelters, and individuals in need.')
st.write('This platform helps reduce food waste by connecting food providers (like restaurants, homes,
or stores) with receivers such as NGOs, shelters, or individuals in need.')

st.subheader('How It Works')

st.write('1.   Register as a Provider or Receiver ')

st.write('2.   Providers list food items available for donation ')

st.write('3.   Receivers browse and claim available food ')

st.write('4.✅ Status of each claim is updated')



st.subheader('Benefits')

st.write('   Reduces food waste')
st.write('   Helps feed those in need')
st.write('   Builds community responsibility')
st.write('   Promotes sustainability')
```

SCREENSHOTS:

● APP VIEW:

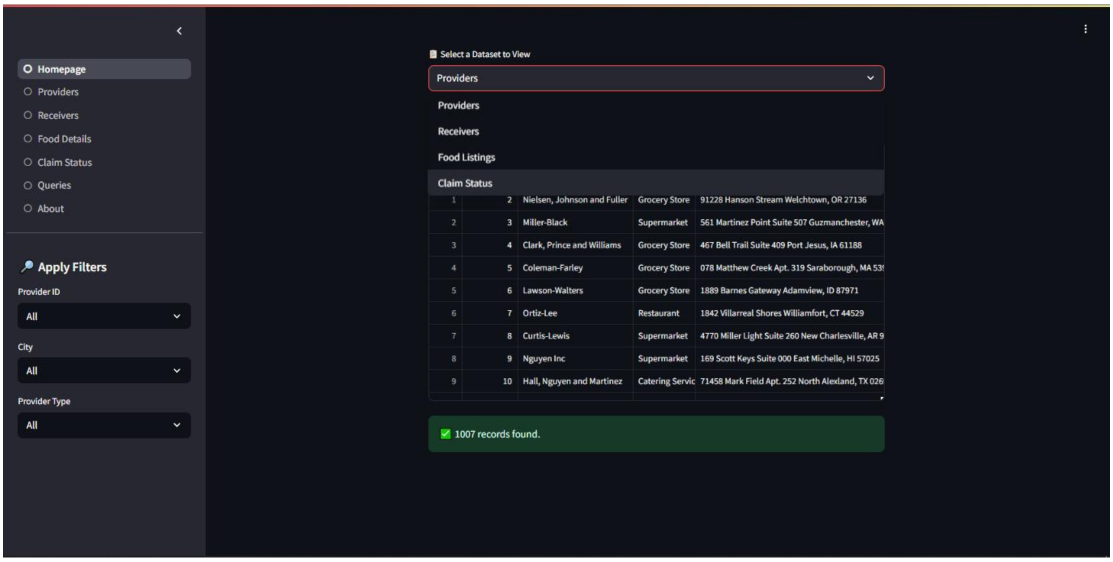


🙏 WELCOME!!!

🍽️ LOCAL FOOD WASTE MANAGEMENT SYSTEM 🌍

Connecting Food Providers with Receivers to Reduce Waste and Feed Communities

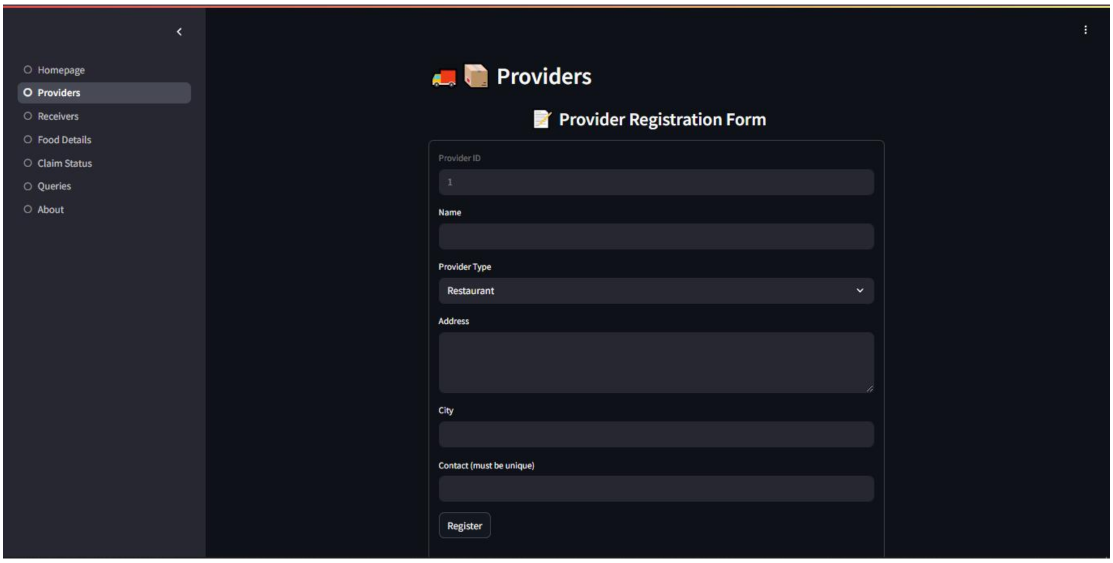
● HOMEPAGE VIEW:



1	2	Nielsen, Johnson and Fuller	Grocery Store	91228 Hanson Stream Welchtown, OR 27136
2	3	Miller-Black	Supermarket	561 Martinez Point Suite 507 Guzmanchester, WA
3	4	Clark, Prince and Williams	Grocery Store	467 Bell Trail Suite 409 Port Jesus, IA 61188
4	5	Coleman-Farley	Grocery Store	078 Matthew Creek Apt. 319 Saraborough, MA 53
5	6	Lawson-Walters	Grocery Store	1889 Barnes Gateway Adamview, ID 87971
6	7	Ortiz-Lee	Restaurant	1842 Villarreal Shores Williamfort, CT 44529
7	8	Curtis-Lewis	Supermarket	4770 Miller Light Suite 260 New Charlesville, AR 9
8	9	Nguyen Inc	Supermarket	169 Scott Keys Suite 000 East Michelle, HI 57025
9	10	Hall, Nguyen and Martinez	Catering Servic	71458 Mark Field Apt. 252 North Alexland, TX 026

✅ 1007 records found.

● PROVIDER PAGE VIEW:



🚚📦 Providers

📝 Provider Registration Form

Provider ID
1

Name

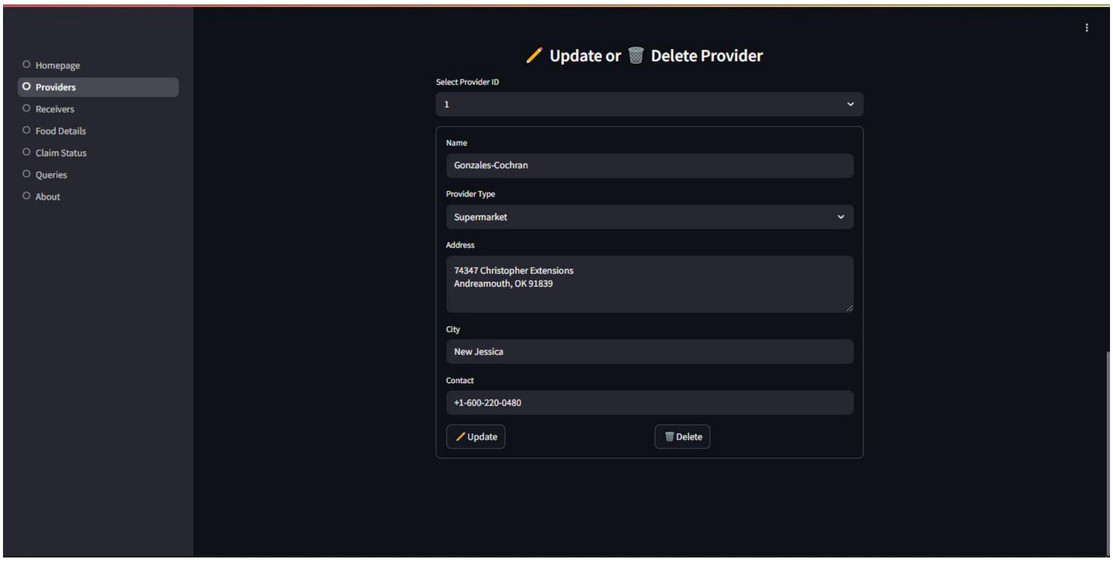
Provider Type
Restaurant

Address

City

Contact (must be unique)

Register



✏️ Update or 🗑️ Delete Provider

Select Provider ID
1

Name
Gonzales-Cochran

Provider Type
Supermarket

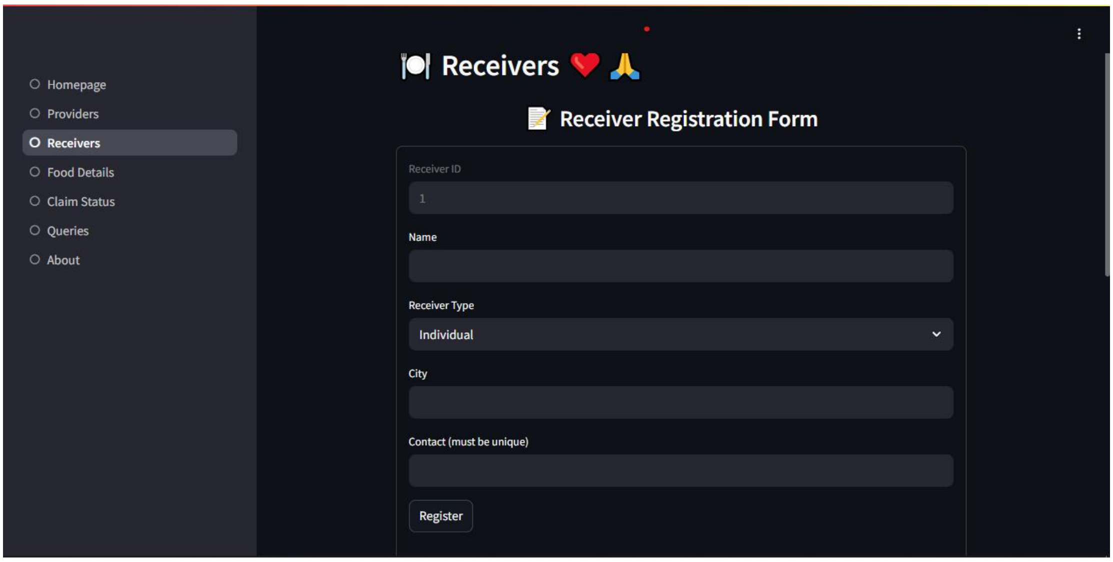
Address
74347 Christopher Extensions
Andreamouth, OK 91839

City
New Jessica

Contact
+1-600-220-0480

✏️ Update 🗑️ Delete

● RECEIVER PAGE VIEW:



🍽️ Receivers ❤️ 🙏

📝 Receiver Registration Form

Receiver ID: 1

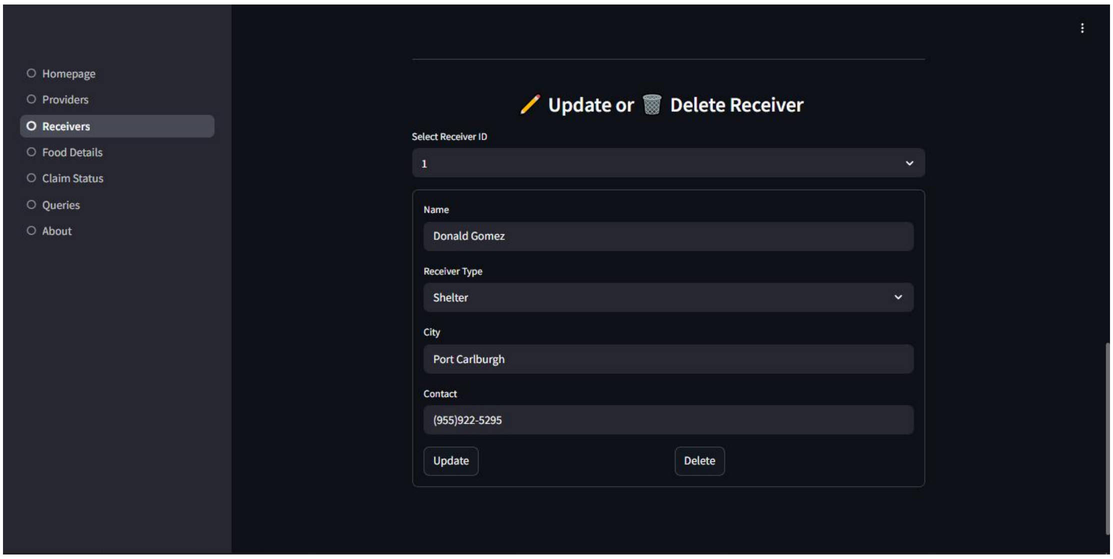
Name

Receiver Type: Individual

City

Contact (must be unique)

Register



✏️ Update or 🗑️ Delete Receiver

Select Receiver ID: 1

Name: Donald Gomez

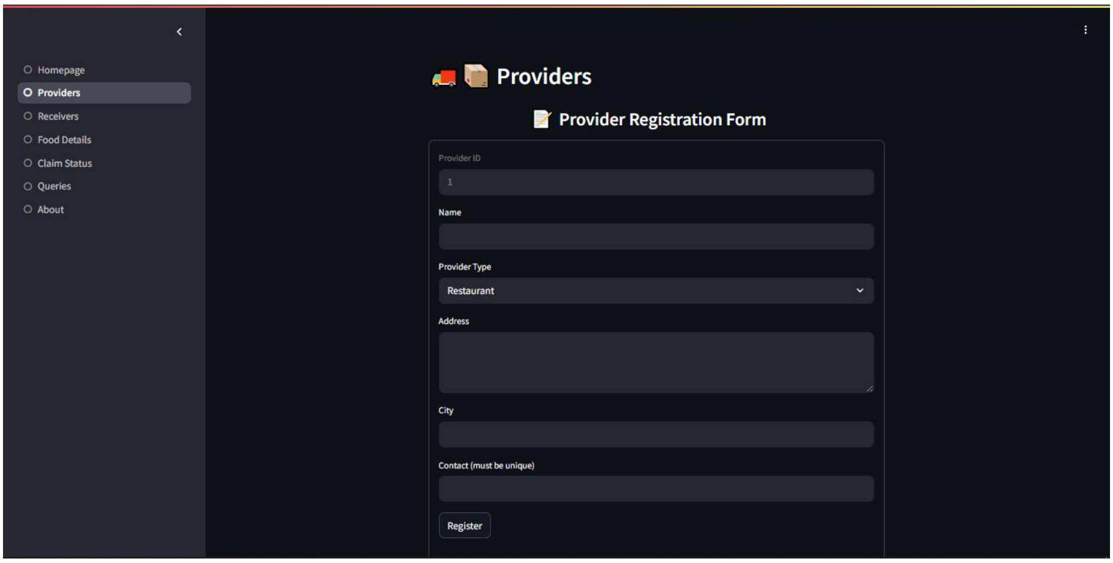
Receiver Type: Shelter

City: Port Carlburgh

Contact: (955)922-5295

Update   Delete

● FOOD LISITING PAGE VIEW:



🚚📦 Providers

📝 Provider Registration Form

Provider ID: 1

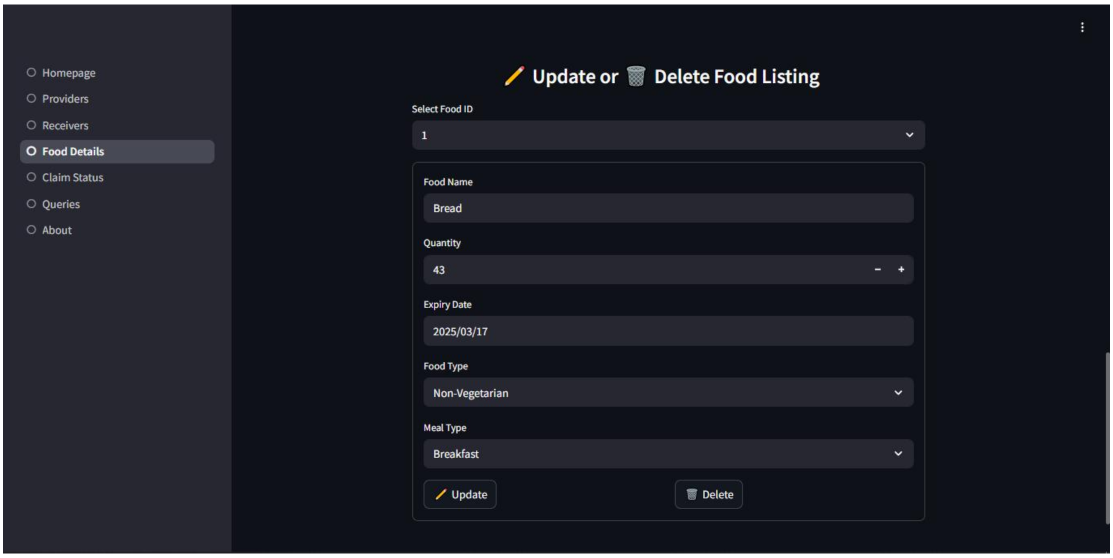
Name:

Provider Type: Restaurant

Address:

City:

Contact (must be unique):



✏️ Update or 🗑️ Delete Food Listing

Select Food ID: 1

Food Name: Bread

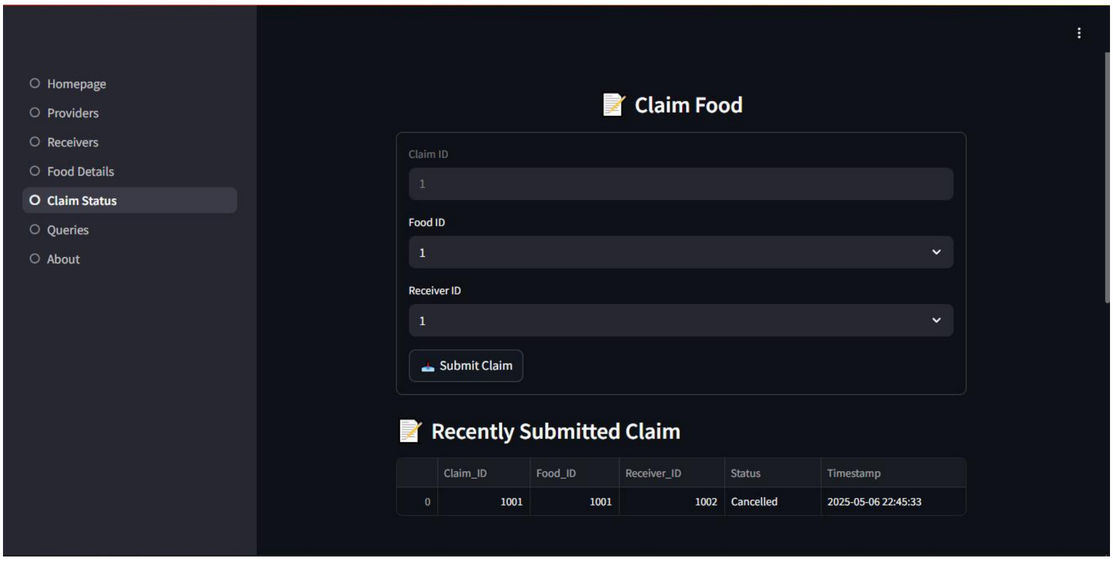
Quantity: 43

Expiry Date: 2025/03/17

Food Type: Non-Vegetarian

Meal Type: Breakfast

● CLAIM STATUS PAGE VIEW:



📝 Claim Food

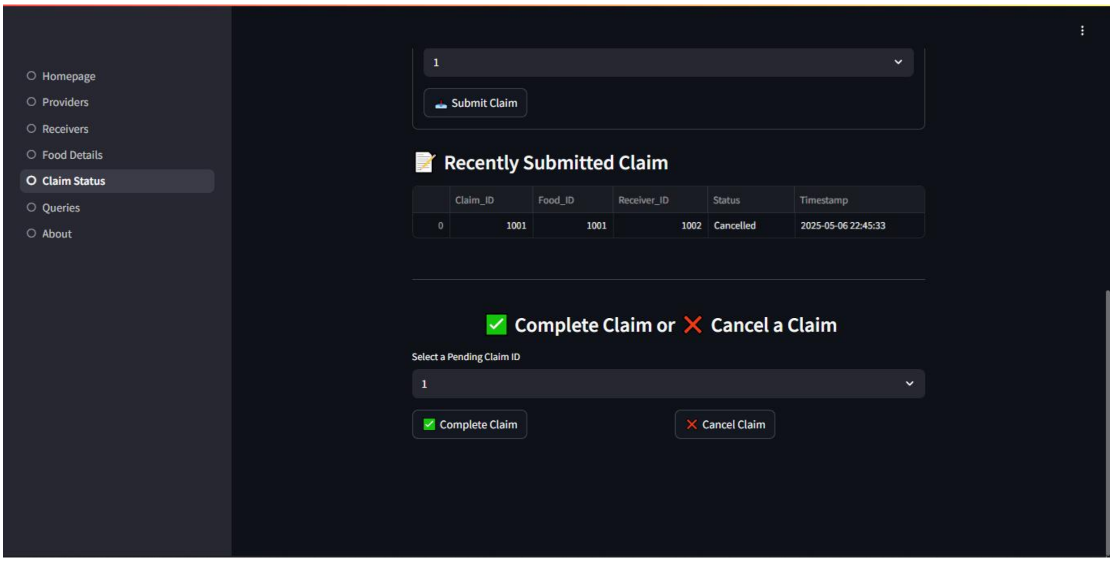
Claim ID: 1

Food ID: 1

Receiver ID: 1

📝 Recently Submitted Claim

	Claim_ID	Food_ID	Receiver_ID	Status	Timestamp
0	1001	1001	1002	Cancelled	2025-05-06 22:45:33



1

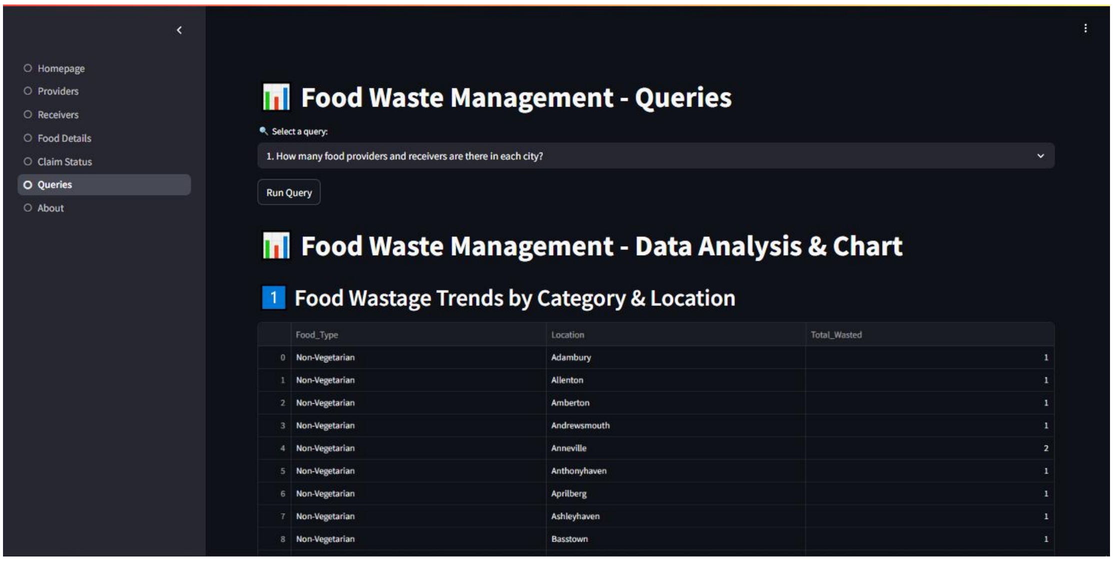
📝 Recently Submitted Claim

	Claim_ID	Food_ID	Receiver_ID	Status	Timestamp
0	1001	1001	1002	Cancelled	2025-05-06 22:45:33

✅ Complete Claim or ❌ Cancel a Claim

Select a Pending Claim ID: 1

● QUERIES AND ANALYSIS CHART VIEW:



📊 Food Waste Management - Queries

🔍 Select a query:

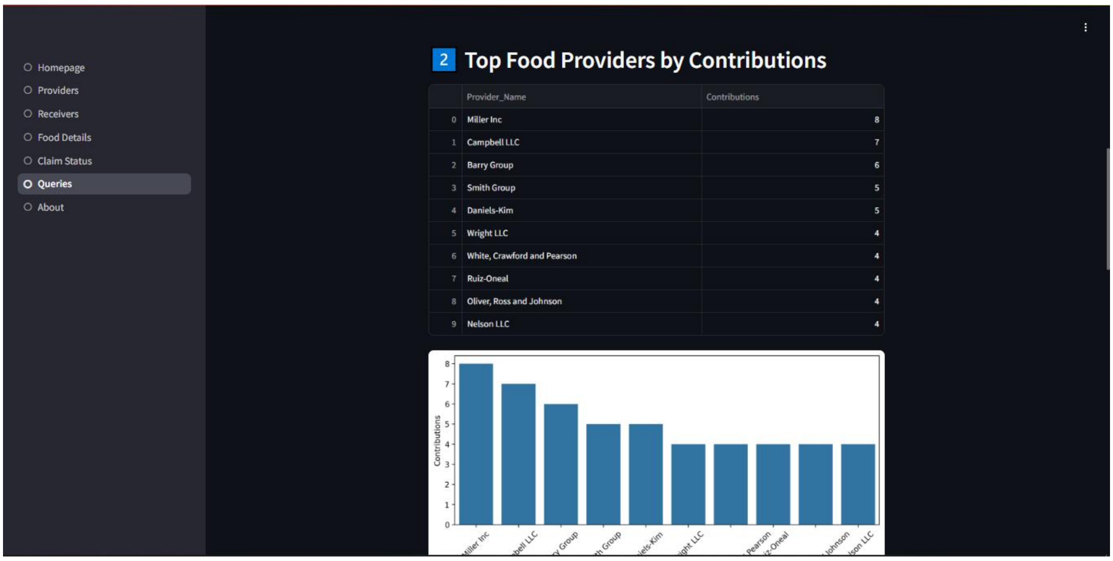
1. How many food providers and receivers are there in each city?

Run Query

📊 Food Waste Management - Data Analysis & Chart

1 Food Wastage Trends by Category & Location

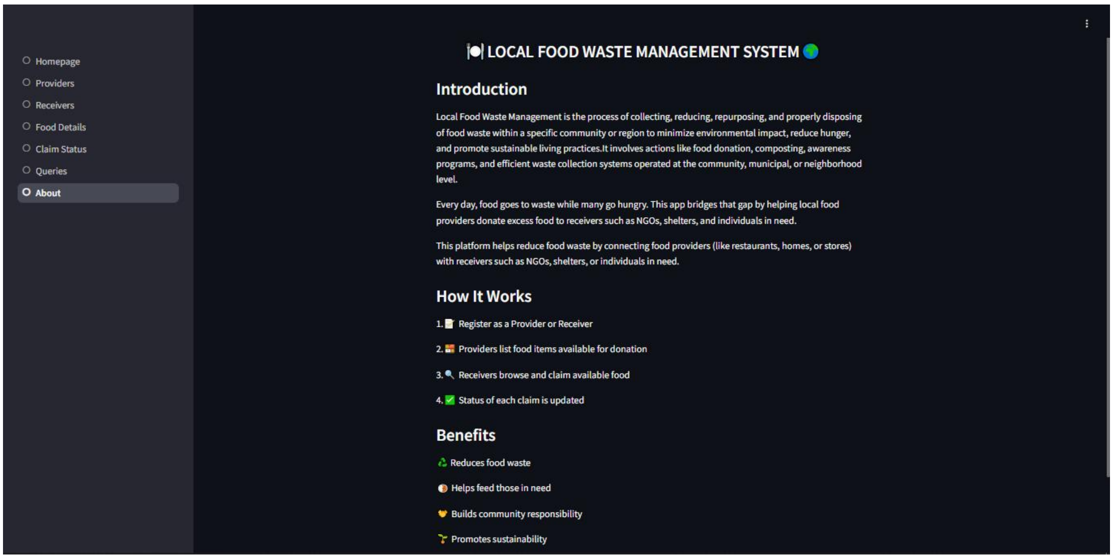
	Food_Type	Location	Total_Wasted
0	Non-Vegetarian	Adambury	1
1	Non-Vegetarian	Allenton	1
2	Non-Vegetarian	Amberton	1
3	Non-Vegetarian	Andrewsmouth	1
4	Non-Vegetarian	Anneville	2
5	Non-Vegetarian	Anthonyhaven	1
6	Non-Vegetarian	Aprilberg	1
7	Non-Vegetarian	Ashleyhaven	1
8	Non-Vegetarian	Basstown	1



2 Top Food Providers by Contributions

	Provider_Name	Contributions
0	Miller Inc	8
1	Campbell LLC	7
2	Barry Group	6
3	Smith Group	5
4	Daniels-Kim	5
5	Wright LLC	4
6	White, Crawford and Pearson	4
7	Ruiz-Oneal	4
8	Oliver, Ross and Johnson	4
9	Nelson LLC	4

- **ABOUT PAGE VIEW:**



🍽️ LOCAL FOOD WASTE MANAGEMENT SYSTEM 🌍

Introduction

Local Food Waste Management is the process of collecting, reducing, repurposing, and properly disposing of food waste within a specific community or region to minimize environmental impact, reduce hunger, and promote sustainable living practices.It involves actions like food donation, composting, awareness programs, and efficient waste collection systems operated at the community, municipal, or neighborhood level.

Every day, food goes to waste while many go hungry. This app bridges that gap by helping local food providers donate excess food to receivers such as NGOs, shelters, and individuals in need.

This platform helps reduce food waste by connecting food providers (like restaurants, homes, or stores) with receivers such as NGOs, shelters, or individuals in need.

How It Works

1. 📝 Register as a Provider or Receiver
2. 🍱 Providers list food items available for donation
3. 🔍 Receivers browse and claim available food
4. ✅ Status of each claim is updated

Benefits

♻️ Reduces food waste

🍛 Helps feed those in need

🤝 Builds community responsibility

🌱 Promotes sustainability

16. References

- Streamlit Documentation
- SQLite Documentation
- Food Waste Reduction articles from FAO
- Python libraries: Pandas, Matplotlib, Seaborn