SustainTrack

Ameni Hsini

January 21, 2024

Abstract

The issue of food waste in Tunisia is becoming increasingly problematic, posing a significant challenge to the country's socio-economic landscape. Annually, 5 percent of food expenditures is wasted while these exist Tunisians suffering from malnutrition. This wasteful trend is stimulated by various contributors, including households, restaurants, backers, and notably retailers, who alone waste a minimum of 2.8 million dinars each year.

To address this urgent issue, the SustainTrack API offers a transformative solution that empowers retailers to efficiently manage their stock, track product validity to minimize waste, and facilitate the donation of surplus goods to local charities. By implementing SustainTrack, businesses can create a positive ripple effect that benefits both their operations and the wider community. This report provides a comprehensive exploration of the API's objectives, components, functional workflows, underlying technologies, and overall structure and operation. SustainTrack serves as a beacon of hope, promoting the creation a more sustainable and socially responsible stock management ecosystem.

Contents

1 Introduction

1.1 Background

According to the Tunisian National Institute for consumption, wasted food in Tunisia represents 5% of food expenditures per year, which is a huge 572 million dinars every year. This represents a severe problem because there are 600,000 Tunisians who suffer from malnutrition. Numerous parties contribute to this waste, including households, restaurants, backers, and notably retailers, whose contribution is estimated at a minimum of 2.8 million dinars per year.

1.2 Objective of SustainTrack

This report introduces a solution to address this issue—SustainTrack—an API created to assist retailers in managing their stock more efficiently, ensuring better tracking of product validity to reduce waste for products meant for consumption, and encouraging the donation of surplus to local charities. The goal of SustainTrack is to make a positive impact on both businesses and the community. This report will delve into the components of this API, its functional flows, the technologies used, and every aspect involved in its structure and functioning.

2 SustainTrack Overview

2.1 SustainTrack Features

2.1.1 Stock Management

- Real-time Tracking: SustainTrack ensures real-time monitoring of the quantities of products, allowing retailers to stay informed about the current status of their stock.
- Add, Update, Delete Products: Retailers can easily add, update, or delete products from their stock through interactive requests, with permanent storage ensured through the products database.

2.1.2 Expiration Date Tracking

- Automated Notifications: SustainTrack sends automated notifications to alert retailers when products are nearing their expiration dates, preventing unnecessary waste.
- Customizable Thresholds: Retailers can set customizable thresholds to tailor notifications based on their specific needs.

2.1.3 Transaction Recording

- Purchase Transactions: Retailers can record purchase transactions, maintaining an accurate record of products entering their inventory.
- Sale Transactions: The API also facilitates the recording of sale transactions, updating stock quantities accordingly.

2.1.4 Donation Facilitation

- Donation Encouragement: Through expiration notifications, the API encourages retailers to donate excess products to avoid waste.
- Charity Database Integration: SustainTrack seamlessly integrates with a local charity database, simplifying the process of identifying and connecting with nearby charities.

2.2 Models and Classes

2.2.1 Product Model

Description: The Product model represents the core entity for managing retail inventory.
Attributes:

- `product_id` (String): Unique identifier for each product.
- `name` (String): Name of the product.
- `price` (Float): Price of the product.
- `expiration_date` (Date): Date indicating the product's expiration.
- *For perishable products, the* ***`expiration_date`*** *signifies the date after which the product is not recommended for consumption. If a product is not perishable, this attribute defaults to null.*
- `quantity_in_stock` (Integer): Current quantity of the product in stock.
- `quantity_sold` (Integer, default=0): Cumulative quantity sold.
- `quantity_purchased` (Integer, default=0): Cumulative quantity purchased.

2.2.2 Transaction Model

Description: The Transaction model captures sales and purchases of products.
Attributes:

- `transaction_id` (String): Unique identifier for each transaction.
- `product_id` (String): Foreign key linking the transaction to a specific product.
- `transaction_type` (String): Indicates whether it's a purchase or sale.
- `transaction_quantity` (Integer): Quantity of products involved in the transaction.
- `timestamp` (DateTime): Date and time when the transaction occurred.

2.2.3 Charity Model

Description: The Charity model represents information about local charities.
Attributes:

- `id` (Integer): Unique identifier for each charity.
- `name` (String): Name of the charity.
- `address` (String): Address of the charity.
- `contact_info` (String): Contact information for the charity.
- `latitude` (Float): Latitude coordinate of the charity's location.
- `longitude` (Float): Longitude coordinate of the charity's location.

2.2.4 Class Diagram

Inserting a class diagram to visually represent the relationships between different classes.

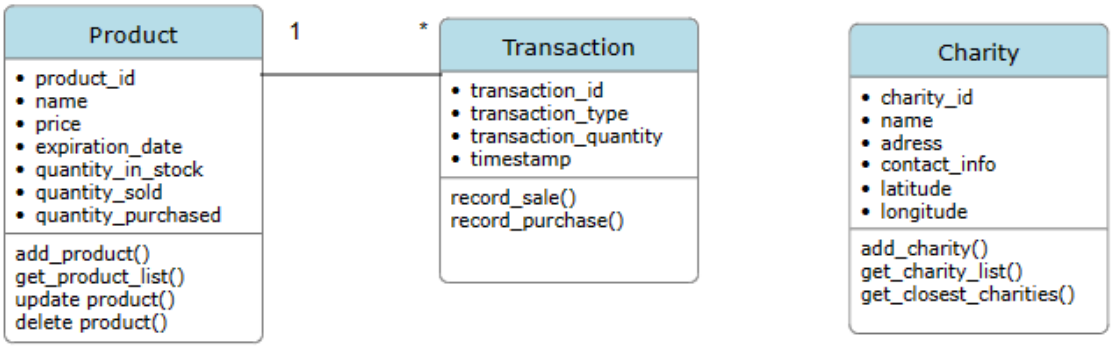

Figure 1: SustainTrack Class Diagram

3 Database Integration

3.1 Relational Database Schema and SQLAlchemy

The SustainTrack API relies on a relational database schema, managed through SQLAlchemy, to ensure efficient data organization and seamless interaction between the API and the underlying data store. We have two databases; the first one is the products database for storing products, and the second one is for storing local charities. Relational data models were deployed since the system includes transactions for structured data. It serves to maintain data integrity and dependency.

ORM Tool: SQLAlchemy: Enables interaction with the database using high-level, Pythonic abstractions. Database System: SQLite: serves as the backend database system.

3.2 Database Operations

3.2.1 Model Definition:

Data models, capturing the essence of entities, are defined within `models.py`. SQLAlchemy facilitates abstraction and interaction with the database. The foundation of the data models is established through the `Base = declarative_base()` declaration. This line signifies the creation of a base class for declarative class definitions within the data models, relationships between entities are established using the `relationship` attribute.

`models.py`

```
class Product(Base):
    __tablename__ = 'product'
    product_id = Column(String(36), primary_key=True)
    name = Column(String(50), nullable=False)
    transactions = relationship("Transaction", back_populates="product")
```

3.2.2 Database Initialization:

In `database.py`, the database is initialized using the `create_engine` function. This function, backed by SQLAlchemy and configured for SQLite, is instrumental in creating tables and structuring the database.

`database.py`

```
# Database Initialization
DATABASE_URL = "sqlite:///charities_db.db"
engine = create_engine(DATABASE_URL)
```

3.2.3 Migration:

When schema modifications are needed over time, Alembic, a powerful migration tool, is deployed. This ensures the evolution of the schema while maintaining data consistency.

```
alembic revision --autogenerate -m "current migration"
alembic upgrade head
```

3.2.4 Connection and Session Management:

This phase involves configuring the database URL, creating an engine for connection, and managing sessions. The `SessionLocal` mechanism, integrated with FastAPI's dependency injection, ensures effective session handling.

`database.py`

```
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
```

SQLAlchemy is a SQL toolkit and Object-Relational Mapping (ORM) library for Python. It provides a set of high-level API for communicating with relational databases, allowing developers to interact with databases using Python objects. SQLite is a lightweight, file-based relational database engine. It's self-contained, serverless, and requires minimal configuration. SQLite is often used for embedded databases and applications where a full-fledged database server is not necessary.

4 FastAPI Framework Integration

4.1 Overview of FastAPI

FastAPI, a cutting-edge web framework for Python 3.6+, has been strategically chosen as the foundation for the SustainTrack API given its key features and benefits.

- **High Performance:** As its name implies, FastAPI is designed for optimal speed, surpassing the performance of many traditional Python frameworks. This makes it an ideal choice for high-performance API development.
- **Automatic API Documentation:** FastAPI's standout feature is its automatic generation of interactive API documentation. This not only simplifies the understanding of API endpoints.
- **Built-in Data Validation:** With built-in support for automatic request validation and serialization, FastAPI streamlines data validation, ensuring that incoming data adheres to expected types.
- **Dependency Injection System:** FastAPI includes a robust dependency injection system, simplifying the management of components such as database sessions within the application.

Dependency injection is a design pattern where the components or services needed by a piece of code are provided to it rather than created within it.

4.2 Deployment Details in SustainTrack

The deployment of FastAPI within the SustainTrack API involves using the `uvicorn` server to run the application. The API can be initiated with the following command:

```
uvicorn app:app --reload
```

4.2.1 Handling Requests:

FastAPI provides an intuitive way to declare API routes, simplifying request handling. It also automatically validates incoming requests, enhancing reliability and endpoint security. Here is an example of defining a route for getting the product list:

```
@app.get("/products/{product_id}")
def read_product(product_id: str):
    # Logic to retrieve product information
    return {"product_id": product_id, "name": "Example Product"}
```

4.2.2 Database Interaction:

FastAPI was integrated with SQLAlchemy to efficiently manage database connections and transactions. In this following example, FastAPI is connected with SQLAlchemy models, allowing the API to interact with the database, for instance, with the products database:

```
@app.post("/products/")
def create_product(product: Product):
    # Logic to add the new product to the database
    return {"message": "Product created successfully"}
```

4.2.3 Automatic API Documentation:

FastAPI automatically generated comprehensive SustainTrack documentation during development. This documentation, available at http://localhost:8000/docs, allows users to interactively explore and understand the API's functionalities and test endpoints.

4.2.4 Testing:

FastAPI facilitated testing in SustainTrack through its built-in test client. A test case for each endpoint creation was created using `pytest` and HTTP requests. In these examples, the tests utilized the FastAPI test client to send a simulated HTTP POST request to the `/products/` endpoint, verifying that the expected response status code was received, and the response matched the anticipated result:

```
def test_create_product():
    response = client.post("/products/", json={"name": "New Product", "price": 10.99})
    assert response.status_code == 200
    assert response.json() == {"message": "Product created successfully"}
```

4.2.5 Dependency Injection:

FastAPI's dependency injection system was used to efficiently manage components like database sessions, resulting in an efficient handling of sessions within the SustainTrack API. Using dependency injection to manage a database session in SustainTrack:

```
@app.post("/products/")
def create_product(product: Product, db: Session = Depends(get_db)):
    # Logic to add the new product to the database using the injected database session
    return {"message": "Product created successfully"}
```

5 Functional Flows

5.1 Product Management

The product management functional flow represents the core operations related to handling stock. Retailers can perform the following actions:

5.1.1 View Products (`/products - GET`):

Retrieve a list of all products in stock, providing details such as product ID, name, price, expiration date, and quantity in stock. This endpoint returns a list of products from the product database.

5.1.2 Add Product (`/products - POST`):

Add a new product to the stock by providing product details, including name, price, expiration date, and initial quantity in stock.

5.1.3 Update Product (`/products/{product_id} - PUT`):

Modify the details of an existing product using its unique product ID. Retailers can update any information related to the product.

5.1.4 Delete Product (`/products/{product_id} - DELETE`):

Remove a product from the product database based on its unique product ID.

5.2 Transaction Recording (Purchase and Sale)

This functional flow captures the process of recording transactions for both purchases and sales, allowing retailers to instantly and dynamically update the quantity of products.

5.2.1 Record Purchase (`/products/{product_id}/purchase - POST`):

A purchase transaction occurs when retailers buy new products from suppliers. This API records a purchase transaction, updating the product's quantity in stock (increased by the quantity purchased) and recording details such as the transaction ID, product ID, transaction type ('purchase'), and quantity purchased.

5.2.2 Record Sale (`/products/{product_id}/sale - POST`):

A sale transaction occurs when the retailer sells products to customers. This endpoint logs a sale transaction, updating the product's quantity in stock (decreases by the sold quantity) and recording details such as the transaction ID, product ID, transaction type ('sale'), and quantity sold.

5.3 Expiration Date Tracking

The expiration date tracking flow ensures timely awareness of approaching product expirations:

5.3.1 (`/check_expiration_dates - POST`):

This functionality aims to minimize product wastage and ensures that retailers stay informed and can make informed decisions about their perishable inventory.

- Track expiration dates (from the products database) for all perishable products.
- Calculate the duration between the expiration date and the current date. If the duration is less than or equal to the specified threshold (e.g., 7 days, adjustable by the retailer), the API notifies the user.
- Facilitate proactive management of expiring products by displaying a permanent message on the interface with a list of products about to expire. Deploy notifications across all interfaces to ensure users are informed, regardless of the platform.
- Retailers can customize the threshold based on factors like demand forecasts, providing flexibility in managing inventory effectively.

5.4 Charitable Donations

The charitable donations flow facilitates the donation process to local charities.

5.4.1 Add Charity (`/charities - POST`):

Add a new charity to the charities database, providing details such as the charity name, address, and contact information. The system automatically geocodes the charity's location.

Process:

- User inputs charity details (It can be done in the API customization by importing a database of local charities, in our example we deployed).
- The API calls Nominatim API for geocoding (geopy.geocoders.Nominatim) which automatically geocodes the charity's provided address to obtain geographical coordinates (latitude and longitude).
- The charities are stored in the charities database, including their information and the automatically obtained coordinates.

5.4.2 Get Charities (`/charities - GET`):

Retrieve a list of all registered charities, including details such as the charity ID, name, address, contact information, latitude, and longitude.

5.4.3 Get Closest Charities (`/charities/closest - POST`):

Find the closest charities to a user-provided address using geolocation. This facilitates informed donation decisions based on proximity. This address might come from a user input field, a device's location data, or any other source where the user's address is obtained.

Process:

- The user-provided address is used to determine proximity to charities.
- Nominatim API is used for geocoding the user's address.
- Retrieve all registered charities from the charities database.
- For each charity, calculate the Haversine distance between its coordinates and the user's coordinates. Haversine Distance Calculation: Calculate the distance between two points on the Earth's surface using their latitudes and longitudes.
- Sort the charities based on their calculated distance and display the closest charities list to the user along with address and contact information.

6 API Dependencies and Integration

The SustainTrack API integrates the Nominatim API for geocoding, enabling the translation of addresses into geographical coordinates (latitude and longitude). This technical integration is facilitated using the geopy library, which simplifies the interaction with various geocoding services, including Nominatim. Once called, we create an instance of the Nominatim geocoder. The geocoder is then used to perform the geocoding by calling its geocode method with the relevant addresses.

6.1 Integration in `add_charity` Function

When adding a new charity, the Nominatim geocoding is used to automatically obtain the coordinates based on the provided address.

```
# Integration in add_charity function
location = geolocator.geocode(charity.address)
if location:
    latitude, longitude = location.latitude, location.longitude
else:
    latitude, longitude = None, None
```

6.2 Integration in `get_closest_charities` Function

Convert the address into coordinates.

```
# Integration in get_closest_charities function
geolocator = Nominatim(user_agent="SustainTrack")
location = geolocator.geocode(user_address)
if location:
    user_location = location.latitude, location.longitude
else:
    latitude, longitude = None, None
```

7 Security Measures

7.1 Authentication

SustainTrack can be customized and integrated into retailers' management systems and ERP, requiring a centralized authentication point for enhanced security. To achieve this, the SustainTrack API adopts JSON Web Tokens (JWT) for authentication.

7.1.1 Tools Used:

- FastAPI's OAuth2AuthorizationCodeBearer: This tool simplifies the integration of OAuth 2.0 into the FastAPI application. It is used for authorization and handling user roles. - JWT (JSON Web Token): The access token, utilized for authentication, is a JWT containing claims such as subject (`sub`), expiration (`exp`), and user role (`role`).

7.1.2 Implementation:

a. OAuth2AuthorizationCodeBearer Setup:

```
from fastapi.security import OAuth2AuthorizationCodeBearer
_scheme = OAuth2AuthorizationCodeBearer(tokenUrl="your_token_url")
```

b. JWT for Authentication: - Users are redirected to the authorization server for secure authentication. - Consent is obtained for the application to access their data. c. Access Token Issuance: - Upon successful authentication, the authorization server issues an authorization code. The `create_access_token` function is used to issue an access token. - This code is exchanged for an access token and a refresh token. d. JWT Access Token Usage: - The access token, encoded as a JWT, is included in API request headers for subsequent secure requests.

7.2 Authorization: Role-Based Access Control (RBAC)

SustainTrack API employs Role-Based Access Control (RBAC) to manage user permissions effectively, critical for data integrity, especially in ERP integration. In a supermarket context, cashiers, managers, and administrators have distinct roles; cashiers only record transactions, managers manage inventory, and administrators have full control.

7.2.1 Tools Used:

- FastAPI's Depends: Utilized to depend on the TokenData class, extracting the user's role for authorization checks.

7.2.2 Implementation:

1. Role Definition: Each user is assigned a specific role (e.g., "retailer" or "admin") during authentication.

2. Route Protection: Certain routes, such as `/protected_route`, are protected based on the user's role.

3. Access Verification: Before processing a protected request, the user's role is checked against the required role for that route.

8 Documentation and Testing

FastAPI interactively generates interactive API documentation using Swagger UI. Here are some key advantages of FastAPI's autogenerated documentation:

- **Automatic Generation of API Playground:** FastAPI generates an interactive and user-friendly API playground (Swagger UI or ReDoc) alongside the traditional OpenAPI documentation, allowing developers to interact with the API visually, making it easier to understand and simplifying the testing process.

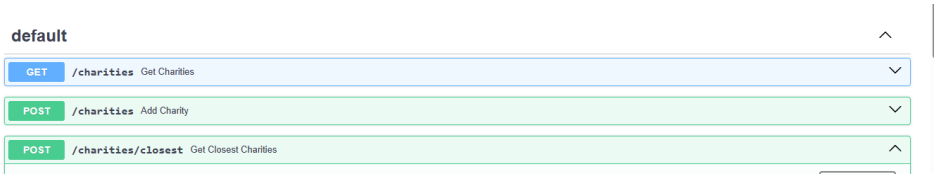

Figure 2: FastAPI Swagger UI

- **Exploration of Endpoints:** The documentation provides a comprehensive list of all available endpoints in the API. It includes all endpoint details, such as HTTP methods, path parameters, query parameters, schemas, and request/response models, which are clearly presented, aiding in exploring and understanding the functionality of each endpoint.

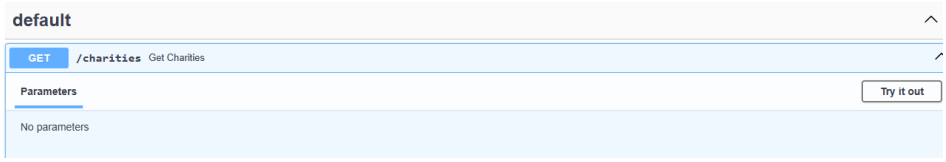

Figure 3: Endpoint Details in FastAPI Documentation

- **Real-Time Testing:** FastAPI documentation includes an interactive "Try it out" feature for each endpoint, allowing users to test API endpoints directly from the documentation interface by inputting parameters and executing requests. This real-time testing helps validate the functionality of endpoints and ensures that they behave as expected. Request parameters and their types are displayed, guiding on the correct input format.

- **Understanding Request/Response Models:** The documentation outlines the expected structure of requests and the format of response data for each endpoint. This information assists in comprehending the contract of each API endpoint, enhancing clarity and consistency in development.

- **Automatic OpenAPI and JSON Schema Generation:** The autogenerated documentation adheres to OpenAPI and JSON Schema standards, making it

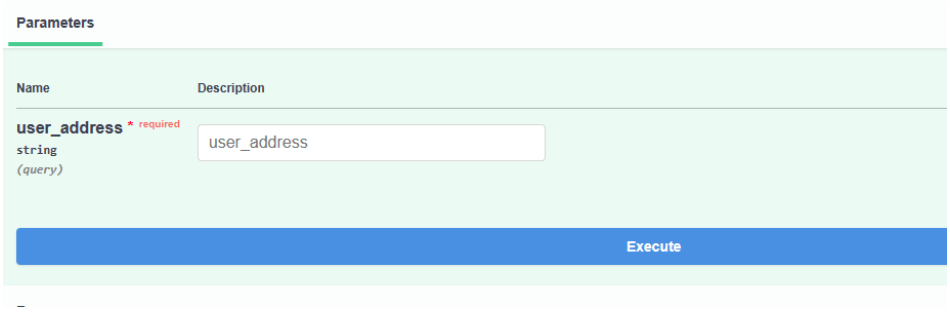

Figure 4: FastAPI Try it Out Feature

Figure 5: Request/Response Models in FastAPI Documentation

compatible with various API tools and ecosystems, providing an interoperable documentation format.

- **Automatic Validation Feedback:** Validation errors for input parameters are highlighted directly within the documentation, aiding in identifying and correcting issues.

9 Conclusion

9.1 Achievements

The development and implementation of the SustainTrack API have yielded significant achievements in meeting its intended goals of revolutionizing inventory management for sustainable and socially responsible practices. The following summarizes the key accomplishments:

- **Real-time Stock Tracking:** SustainTrack enables retailers to track stock dynamically and in real-time, minimizing waste and optimizing stock levels. This feature aids in reducing environmental impact and promoting responsible resource management.
- **Efficient Charity Integration:** The API successfully facilitates the integration of charities into the retail stock management system. Charities can be seamlessly added, providing a platform for retailers to contribute to social causes.
- **High Level of Customization and Integration:** SustainTrack features are customizable to customer businesses, existing systems, and locations. It can also be integrated to add a social approach to retailers' management systems.
- **Donation Encouragement and Facilitation:** Through notification and finding nearest charities quickly, SustainTrack facilitates the donation process, encouraging retailers to donate.

9.2 Future Developments

Looking forward, there are several potential areas for future development and improvement for the SustainTrack API, ensuring a more impactful and adaptive tool for sustainable and socially responsible stock management.

- **Integration with AI:** Artificial Intelligence deployment will make SustainTrack more dynamic and effective in donation matching. It will elevate the charity matching process and identify unique donation opportunities. By integrating with AI algorithms, SustainTrack will monitor social media platforms for charity requests and events, facilitating precise and timely matching with retailers willing to contribute. AI will also help in integration with local donation platforms, making donation more direct and easier.
- **Enhanced Analytics and Demand Forecasting:** To empower retailers with even more informed decision-making, the API will delve into advanced analytics and demand forecasting. By implementing predictive algorithms, SustainTrack aims to forecast demand trends for specific products. This innovation will not only optimize stock management but also guide retailers in making proactive

decisions regarding potential excess quantities. Additionally, the API will introduce dynamic donation recommendations, suggesting ideal donation quantities based on real-time demand and inventory data. Performance metrics, including detailed analytics on waste reduction achievements and donation contributions, will be incorporated to provide retailers with a comprehensive overview of their sustainability efforts.

- **Enhanced Customization:** As the issue of food waste is crucial, customization of SustainTrack features is needed, allowing retailers to tailor the platform to their specific requirements. This customization spans across various sectors, with a focus on addressing unique needs. For example, for restaurants, the API can be adapted to include features like perishable goods tracking, meal preparation data integration, and efficient donation scheduling.

A Appendix A: Response Examples

A.1 Product Management

A.1.1 GET /products


```
[
  {
    "product_id": "4307a68d-9e60-498a-8ab1-ec41347fd961",
    "name": "chips",
    "price": 4,
    "expiration_date": "2024-01-31",
    "quantity_in_stock": 50
  },
  {
    "product_id": "69ed00a4-dbe2-4637-8e1f-4158340c91d0",
    "name": "milk",
    "price": 1,
    "expiration_date": "2024-01-27",
    "quantity_in_stock": 10
  },
  {
    "product_id": "1fbefabc-b457-4c3f-8cc5-0bf094db7b90",
    "name": "Coke",
    "price": 3,
    "expiration_date": "2024-01-24",
    "quantity_in_stock": 15
  }
]
```

A.1.2 POST /product


```
{
  "message": "Product added successfully",
  "product_id": "53045c26-646f-4aba-aaeb-82038b919c06"
}
```


A.1.3 PUT /products/{product_id}


```
{
  "message": "Product updated successfully"
}
```


A.1.4 DELETE /products/{product_id}

A.1.5 POST /check_expiration_dates


```
{
  "log_messages": [
    "Product 'milk' is approaching its expiration date on 2024-01-27",
    "Product 'Coke' is approaching its expiration date on 2024-01-24",
    "Product 'biscuits' is approaching its expiration date on 2024-01-20"
  ]
}
```


A.1.6 POST /products/{product_id}/purchase


```
{
  "message": "Purchase recorded successfully"
}
```


A.1.7 POST /products/{product_id}/sale


```
{
  "message": "Sale recorded successfully"
}
```


A.2 Donation Facilitation

A.2.1 GET /charities


```
[
  {
    "id": 5,
    "name": "SOS Siliana",
    "address": "Siliana, Siliana, Tunisia",
```

```
    "contact_info": "71256987",
    "latitude": 36.0846651,
    "longitude": 9.3743733
  },
  {
    "id": 6,
    "name": "SOS tunis",
    "address": "Boulevard de l'Environnement, 1057 Tunis",
    "contact_info": "+216 71 919 911",
    "latitude": 36.904223,
    "longitude": 10.2974272
  }
]
```


A.2.2 POST /charities


```
{
  "id": 19,
  "name": "Test_charity",
  "address": "Tunis",
  "contact_info": "email",
  "latitude": 36.8002068,
  "longitude": 10.1857757
}
```


A.2.3 POST /charities/closest


```
[
  {
    "id": 19,
    "name": "Test_charity",
    "address": "Tunis",
    "contact_info": "email",
    "latitude": 36.8002068,
    "longitude": 10.1857757
  },
  {
    "id": 1,
    "name": "Test Charity",
    "address": "Boulevard de l'Environnement, Tunis, Tunisia 1057",
    "contact_info": "test@example.com",
    "latitude": 36.904223,
    "longitude": 10.2974272
```

```
  },
  {
    "id": 6,
    "name": "SOS tunis",
    "address": "Boulevard de l'Environnement, 1057 Tunis",
    "contact_info": "+216 71 919 911",
    "latitude": 36.904223,
    "longitude": 10.2974272
  }
]
```