SoukSpot API Platform

Revolutionizing Tunisian Souks Trading

Presented by: Roua Othmani

Supervised by: Pr.Montassar Ben Messaoud

Tunis Business School
Major: Business Analytics
Minor: Information Technology

2024-2025

Abstract

The traditional Tunisian souks present a fragmented trading environment, which creates difficulties for both vendors and buyers, limiting their access to each other, and creating an overall lack of transparency in the system. This project addresses this problem by proposing the development of a unified API platform, which acts as a central hub connecting local vendors and customers. By providing a structured method for vendors to list and manage their products and for customers to find products based on a personalized recommendations, and also improving the communication between them, the platform aims to revolutionize the traditional souk system.

For local Tunisian vendors, the platform offers an opportunity to expand their reach, present their unique local products to a new demographic of customers, and improve their trading experience by having a well structured system for their products and offers.

This project presents a way to transform the mobile marketplace landscape by creating a more efficient, transparent, and empowering system for both sides of the marketplace.

Contents

List of Figures

1 Introduction

The Tunisian Souk landscape, characterized by the vast number of vendors offering unique traditional products and services, presents both opportunities for growth and complex problems for the community. These problems arise from lack of methods for the user to discover all of the vendors and products in the souks, and a lack of an online structure that creates new opportunities for vendors to showcase their products and services to a wider audience, reaching new types of users, including tourists and also local clients that otherwise are not able to access the physical souks.

This project proposes a groundbreaking solution: a unified API platform, called the SoukSpot API, which is a central place that connects vendors and their products to a wider audience. By providing new tools to manage their products, offers, and also by giving customers a new and easier way to discover new products and local vendors, it aims to transform the mobile landscape in Tunisia.

Through its innovative approach, this project has the potential to revolutionize the customer experience in the souk by enhancing their ability to explore all products and services in the market, while giving local vendors a system to manage their presence and reach their clients in a more effective way. The platform has a robust technical architecture, that facilitates the communication between vendors, customers, and also with the database. This will also be used to enable new methods of price negotiation and transparent trading, making the process a lot easier.

This project delves into the design and development of this unified API platform, exploring its technical architecture and key functionalities.The subsequent sections will provide a deeper understanding of the platform's functionalities, its impact on users and vendors, and its potential to usher in a new era of connectivity and innovation in the Tunisian Souk landscape, by showcasing local artisan and craft products to both locals and tourists.

2 Motivation

2.1 Problem Statement

Traditional Tunisian souks, while rich in cultural heritage and diverse local products, face significant hurdles in the modern digital landscape. These challenges impact both vendors and customers, hindering economic and cultural growth. Key problems include:

- **Limited Online Presence for Local Vendors:** Most vendors in the souks operate exclusively offline, lacking the tools and skills to reach a wider audience. This significantly limits their customer base, particularly for those unable to attend the physical markets, affecting their visibility and potential revenue from both tourists and locals.
- **Inconsistent and Inaccessible Information:** Crucial information such as vendor offerings, opening hours, pricing, and product details are often unavailable online, making it hard for potential customers to browse their options and understand what is being offered.
- **Limited Opportunities for Price Negotiation:** The traditional bargaining process can be difficult and daunting for many customers to engage with, requiring a more formal and transparent method for handling offers and negotiations.
- **Difficulties in Promoting Unique Traditional Products:** Local artisans and craft vendors often struggle to promote their unique traditional products to a broader audience beyond the physical confines of the souks, especially with the lack of an established method for promotion.
- **Inefficient Catalog Management**: Vendors struggle with keeping track of their available products, update the information, manage prices, and other details due to the lack of access to management tools, leading to problems and making the information less reliable.

Important Note

Thus,here comes the motivation behind **the SoukSpot Platform: Revolutionizing the Tunisian Souks Trading**

2.2 Solution

The SoukSpot API Platform offers a suite of features designed to address the challenges faced by Tunisian souks, empowering vendors and improving the overall experience for customers, also enhancing the experience for the tourists and the locals.

2.2.1 Vendor Management

- **Vendor Profile Creation:** Enables vendors to create profiles with details about their offerings, location, and contact information.
- **Vendor Discovery and Filtering:** Allows users to easily browse and find vendors by category or Souk name, or see all vendors in the platform.

2.2.2 Product Catalog Management

- **Product Listing Management:** Enables vendors to add, manage, and remove their product listings with details such as price, description, and discounts.
- **Product Discovery and Search:** Allows users to browse and find products by categories, names, or to see all products in the system.
- **Product Detail Retrieval**: Gives access to more details about the products in your platform.

2.2.3 Review Management

- **Review Posting and Creation:** Empowers users to post new reviews on the platform, providing feedback on products and vendors.
- **Review Discovery and Viewing:** Allows users to list the reviews or to check one.

2.2.4 Price Negotiation through Bargaining Sessions

- **Offer Creation and Acceptance:** Allows users to make offers on the products they want to buy and vendors to respond to the offers
- **Offer Status Management:** Allows users to retrieve the current status of their bargaining sessions.

2.2.5 Order Product Management

- **Manage Items in Orders:** Allows users to see all products in a specific order and update the details of their orders.
- **Order Placement and Tracking:** Allows users to create orders, retrieve information about them, or view their previous orders.

2.2.6 Personalized Product Recommendations

- **Tailored Product Discovery:** The platform offers product recommendations based on your purchase history and their product category preferences, making it easier for you to find new products that might interest you.

3 Technologies Used

This section outlines the key tools and their specific utility in the development, the deployment and operations of the system

3.1 FastAPI

- **Description:** A modern, high-performance, web framework built with Python that serves as the core of the API.

3.2 FastAPI Swagger UI

- **Description:** An interactive API documentation interface, automatically generated by FastAPI. It allows developers to see how the methods work, and explore your API endpoints/docs endpoint'.

3.3 Postman

- **Description:** It is used as a popular tool for API testing and debugging by creating and sending HTTP requests and viewing their responses.

3.4 SQLite

- **Description:** It's a lightweight, file-based, relational database engine. Used to store all the data, in an easier way to handle, compared to other more complex databases.

3.5 SQLAlchemy

- **Description:** It's used as a Python SQL toolkit and Object-Relational Mapper (ORM).It provides an abstraction layer that lets you interact with the database using Python objects.

3.6 JSON Web Tokens (JWT)

- **Description**: JSON Web Tokens are a standard method for securely sending and validating data, used for authentication.

3.7 Podman

- **Description**: Podman is used as a containerization platform that packages applications into standardized units for easier development and deployment. Podman provides a daemonless and rootless alternative to Docker, offering enhanced security and efficiency."

4 System Architecture and Design

The SoukSpot API platform is built as a REST API and the architecture follows a layered structure that allows for separation of concerns. This helps in making the codebase more scalable, easy to maintain and also extensible for future additions.

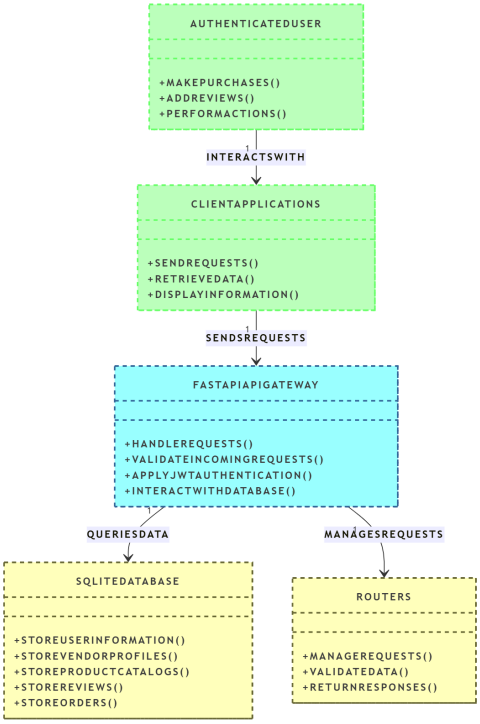

Figure 4.1: High-Level System Architecture for the SoukSpot Platform

4.1 Database Usage

The SoukSpot API Platform stores and manages data using an SQLite database and the SQLAlchemy ORM. The database structure is created dynamically using the SQLAlchemy declarative base, and the database sessions are controlled to ensure that all the data is being inserted and updated correctly. The platform automatically creates the database when running the main file. The initial database content is created and populated by a ‘seed-database()‘ function. This function adds initial data for users, vendors, products, reviews, and other tables.

4.2 Models

The project uses several models, which are representations of tables and their attributes in the database, they are:

- **User**: Represents a user account within the SoukSpot platform.
- **Vendor**: Represents a seller profile within the SoukSpot marketplace.
- **Product**: Represents a product available for purchase.
- **Review**: Represents a user's feedback on a vendor or product.
- **OrderProduct**: Represents a purchase made by a user.
- **BargainingSession**: Represents a price negotiation process between a user and a vendor.
- **Purchase**: Represents a record of a user's purchase of a product.
- **Souk**: Represents a marketplace in the platform.

4.3 Class Diagrams

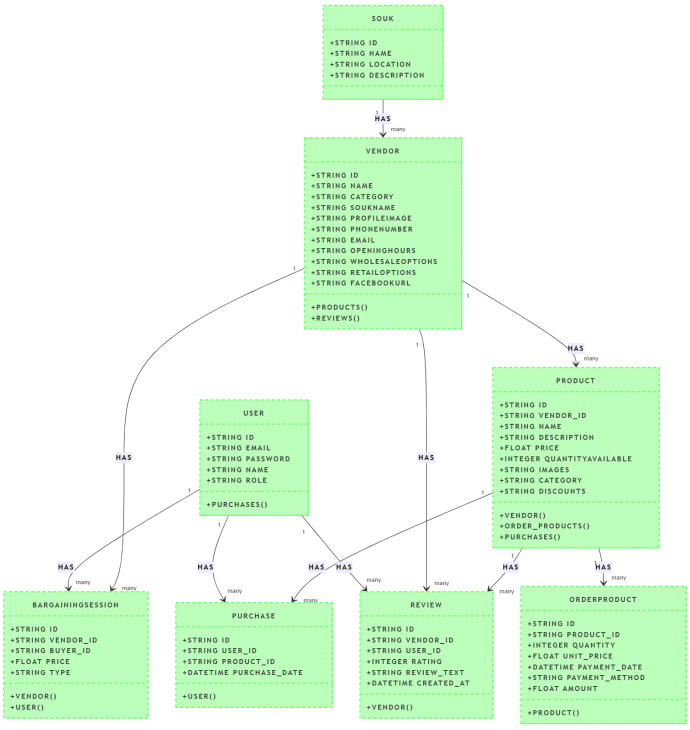

Figure 4.2: Class Diagram of SoukSpot Platform

5 Security Implementation

The SoukSpot API platform implements several security measures to protect user data and ensure secure access to various functionalities.

5.1 Password Hashing

To ensure that the passwords are not exposed in plain text in case of a security breach, the passwords are hashed using **bcrypt library**. This one-way hashing function uses a unique salt for each password, and it's not reversible. This method provides a more secure way of storing and handling user passwords.

The hashing is implemented in the `/users/signup` route. When a new user is registered the following process happens:

- The password is encoded into bytes, a unique salt is generated using **bcrypt.gensalt()** method.
- the **bcrypt.hashpw()** is then used to generate the hash with the password and the salt.
- The hashed password is then saved in the database using the user model.

```python
@router.post("/signup", response_model=UserSchema, status_code=status.HTTP_201_CREATED)
def signup_user(user: UserCreate, db: Session = Depends(get_db)):
    # Check if the user already exists
    existing_user = db.query(UserModel).filter(UserModel.email == user.email).first()
    if existing_user:
        raise HTTPException(status_code=400, detail="Email already registered")

    # Hash the password
    hashed_password = bcrypt.hashpw(user.password.encode('utf-8'), bcrypt.gensalt())
    db_user = UserModel(
        email=user.email,
        name=user.name,
        role=user.role,
        password=hashed_password.decode('utf-8')  # Store the hashed password
    )
    db.add(db_user)
    db.commit()
    db.refresh(db_user)
    return db_user
```

Figure 5.3: Password Hashing

5.2 JWT (JSON Web Tokens) for Authentication

JSON Web Tokens (JWT) are used as a standard method for securely exchanging tokens to authenticate requests to the API endpoints. When a user logs in, a unique token is generated and it's being used to authorize the rest of the requests to the API.

- The JWT tokens are stored in the `authorization header`, prefixed by extt-tBearer.
- The tokens have an expiration time to ensure that they cannot be used for an indefinite time, and the time of expiration is customizable with an environment variable.
- If the token is not valid, or it is expired the request will return an error of unauthorized.

```python
@router.post("/login")
def login_user(user: UserLogin, db: Session = Depends(get_db)):
    # Retrieve user from the database
    db_user = db.query(UserModel).filter(UserModel.email == user.email).first()
    if not db_user:
        raise HTTPException(status_code=401, detail="Invalid email or password")

    # Verify the password
    if not bcrypt.checkpw(user.password.encode('utf-8'), db_user.password.encode('utf-8')):
        raise HTTPException(status_code=401, detail="Invalid email or password")

    access_token = create_access_token({"sub": user.email}, expires_delta=timedelta(minutes=15))
    return {"access_token": access_token, "token_type": "bearer"}
```

Figure 5.4: User authentication

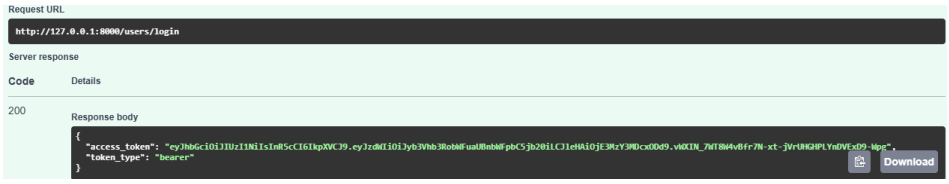

Figure 5.5: Token generation

6 API Endpoints

The SoukSpot API Platform exposes a set of endpoints designed to offer functionalities for users and vendors.

6.1 Vendor Management Endpoints

- **GET /vendors**: retrieves a list of all vendors and their details registered in the platform in a structured JSON format.
- **GET /vendors/category/{category}**:The user retrieves a list of all vendors that match the specified category, or returns a 404 if no vendors match that category
- **GET /vendors/souk/{souk_name}**: The user retrieves all vendors that are located in a specific Souk (market), by using the Souk name.
- **POST /vendors**: The user can create a new vendor profile by making a 'POST' request to this endpoint with valid data.
- **DELETE /vendors/{vendor_id}**: Admin can delete vendor profile from the platform using his specific id.

```python
@router.post("/", response_model=VendorSchema, status_code=status.HTTP_201_CREATED)
def create_vendor(vendor: VendorCreate, db: Session = Depends(get_db), current_user:
    TokenData = Depends(decode_access_token)):
    db_vendor = Vendor(**vendor.model_dump())
    db.add(db_vendor)
    db.commit()
    db.refresh(db_vendor)
    return db_vendor

@router.get("/", response_model=List[VendorSchema])
def get_all_vendors(db: Session = Depends(get_db), current_user: TokenData = Depends(decode_access_token)):
    return db.query(Vendor).all()

@router.get("/category/{category}", response_model=List[VendorSchema])
def get_vendor_by_category(category: str, db: Session = Depends(get_db), current_user:
    TokenData = Depends(decode_access_token)):
    vendors = db.query(Vendor).filter(Vendor.category == category).all()
    if not vendors:
        raise HTTPException(status_code=404, detail="Vendor not found in this category")
    return vendors
```

Figure 6.6: Vendor Endpoints

6.2 Product Catalog Management Endpoints

- **GET /products**: Users can access all the products, by accessing this endpoint.
- **GET /products/category/{category}**: Users can access and filter the products by category,if the category does not exist it will return an error.

- **GET /products/name/{product_name}**: user can use this method for finding all the products by a specific name or a part of the name.
- **POST /products**: An authenticated user can create a new product by using a 'POST' request, and providing the product details in the body.

```python
@router.get("/", response_model=List[ProductSchema])
def get_products(db: Session = Depends(get_db), current_user: TokenData = Depends(decode_access_token)):
    products = db.query(ProductModel).all()
    return products

@router.get("/category/{category}", response_model=List[ProductSchema])
def get_products_by_category(category: str, db: Session = Depends(get_db), current_user:
                             TokenData = Depends(decode_access_token)):
    products = db.query(ProductModel).filter(ProductModel.category == category).all()
    return products

@router.post("/", response_model=ProductSchema, status_code=status.HTTP_201_CREATED)
def create_product(product: ProductCreate, db: Session = Depends(get_db), current_user:
                   TokenData = Depends(decode_access_token)):
    db_product = ProductModel(**product.model_dump())
    db.add(db_product)
    db.commit()
    db.refresh(db_product)
    return db_product
```

Figure 6.7: Product Endpoints

6.3 Review Management Endpoints

- **GET /reviews**: It can be used to see how the users rate the products.
- **POST /reviews**: Creates a new review for a vendor or a product with a specific rating and text.

```python
@router.get("/", response_model=List[ReviewSchema])
def get_reviews(db: Session = Depends(get_db), current_user: TokenData = Depends(decode_access_token)):
    reviews = db.query(ReviewModel).all()
    return reviews

@router.post("/", response_model=ReviewSchema, status_code=status.HTTP_201_CREATED)
def create_review(review: ReviewCreate, db: Session = Depends(get_db), current_user: TokenData = Depends(decode_access_toke
    db_review = ReviewModel(**review.model_dump())
    db.add(db_review)
    db.commit()
    db.refresh(db_review)
    return db_review
```

Figure 6.8: Review Endpoints

6.4 Bargaining Session Management Endpoints

- **POST /bargaining/offer/{vendor_id}**: Allows the user to create a new offer for a specific vendor.
- **GET /bargaining/status/{vendor_id}**: the admin can retrieve the status of a bargaining session between a user and a vendor.
- **POST /bargaining/accept/{vendor_id}**: Allows a vendor to accept a specific offer.

```python
@router.post("/offer/{vendor_id}", response_model=BargainingSchema, status_code=status.HTTP_201_CREATED)
def make_offer(vendor_id: str, offer: BargainingSchema, db: Session = Depends(get_db), current_user: TokenData = Depends(de
    #verify the user
    vendor = db.query(Vendor).filter(Vendor.id == vendor_id).first()
    if not vendor:
        raise HTTPException(status_code=404, detail="Vendor not found")
    buyer = db.query(User).filter(User.email == current_user.username).first()
    if not buyer:
        raise HTTPException(status_code=404, detail="Buyer not found")

    db_bargain = db.query(BargainingSession).filter(BargainingSession.vendor_id == vendor_id, BargainingSession.buyer_id ==

    if not db_bargain:
        db_bargain = BargainingSession(vendor_id=vendor_id, buyer_id = current_user.username, price = offer.price, type = "
        db.add(db_bargain)
    else:
        db_bargain.price = offer.price
        db_bargain.type = "offer"
    db.commit()
    db.refresh(db_bargain)
    return db_bargain

@router.get("/status/{vendor_id}", response_model=BargainingSchema)
def get_bargaining_status(vendor_id: str, db: Session = Depends(get_db), current_user: TokenData = Depends(decode_access_to
    db_bargain = db.query(BargainingSession).filter(BargainingSession.vendor_id == vendor_id, BargainingSession.buyer_id ==
    if not db_bargain:
        raise HTTPException(status_code=404, detail="Bargaining session not found")
    return db_bargain
```

Figure 6.9: Bargaining Endpoints

6.5 Personalized Recommendation Endpoints

- **POST/recommendations/purchases**: Allows the user to create a new offer for a specific vendor.
- **GET /bargaining/status/{vendor_id}**:This method creates a new record of a user's purchase with the product id and the date, this will be used in a future process to generate recommendations.
- **GET /recommendations/user/{user_id}**: This method retrieves the purchase history of a user to show what they might want to buy, by listing the most frequently bought products in the database. If no products are available from their history it will retrieve 3 random products from the platform.

```python
@router.post("/purchases", response_model=PurchaseSchema, status_code=status.HTTP_201_CREATED)
def record_purchase(purchase: PurchaseCreate, db: Session = Depends(get_db), current_user:
                    TokenData = Depends(decode_access_token)):
    db_purchase = PurchaseModel(**purchase.model_dump())
    db.add(db_purchase)
    db.commit()
    db.refresh(db_purchase)
    return db_purchase

@router.get("/user/{user_id}", response_model=List[ProductSchema])
def get_recommendations_for_user(user_id: str, db: Session = Depends(get_db), current_user:
    TokenData = Depends(decode_access_token)):
    purchases = db.query(PurchaseModel).filter(PurchaseModel.user_id == user_id).all()

    if not purchases:
        return []
    product_counts = defaultdict(int)
    for purchase in purchases:
        product_counts[purchase.product_id] += 1

    top_product_ids = sorted(product_counts, key=product_counts.get, reverse=True)
    all_products = db.query(ProductModel).all()

    products_bought_ids = [purchase.product_id for purchase in purchases]
    available_products = [product for product in all_products if product.id not in products_bought_ids]
    if not available_products:
        return []
    return available_products[:3]
```

Figure 6.10: Recommendation Endpoints

6.6 Order Management Endpoints

- **GET /orders**: This endpoint lists all existing orders in the platform, offering a comprehensive view of purchase activity.
- **POST /orders**: Allows a user to create a new order in the system.
- **PUT /order_products/{order_product_id}**: This allows updating details on an existing order product using its id.

```python
@router.get("/", response_model=List[OrderProductSchema])
def get_order_products(db: Session = Depends(get_db), current_user: TokenData = Depends(decode_access_token)):
    order_products = db.query(OrderProductModel).all()
    return order_products

@router.post("/", response_model=OrderProductSchema, status_code=status.HTTP_201_CREATED, summary="Create a new order produ
def create_order_product(order_product: OrderProductCreate, db: Session = Depends(get_db), current_user: TokenData = Depend
    try:
        db_order_product = OrderProductModel(**order_product.model_dump())
        db.add(db_order_product)
        db.commit()
        db.refresh(db_order_product)
        return db_order_product
    except Exception as e:
        logging.error(f"Error creating order product: {e}", exc_info=True)
        raise HTTPException(status_code=500, detail="Internal Server Error")
```

Figure 6.11: Order Management Endpoints

7 Deployment Overview

The SoukSpot API is designed to be deployed using container technology for maximum flexibility and scalability. The container image, built with **Podman**, is stored in a container registry and deployed to a cloud service or a local server, depending on the use case. The deployment process involves these main steps:

- **Build:** The first step is to create the container image using the ‘podman build‘ command and a Dockerfile.
- **Push:** The Docker image will then be uploaded to a Docker registry, such as Docker Hub, using the ‘podman push‘ command.
- **Pull:** The image will then be pulled from the registry in the server using ‘podman pull‘.
- **Run:** Finally, the container will be started in the server using ‘podman run‘, which will make your application accessible through the specified port.

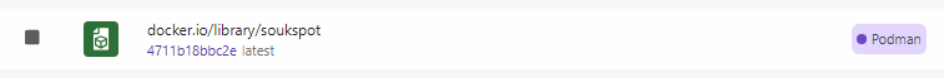

Figure 7.12: Container Image

8 Results

The SoukSpot API platform achieves several key outcomes that demonstrate its effectiveness in creating a modern and usable trading platform, with its main accomplishments including:

- **Empowered Vendors Through Efficient Management:** The platform has demonstrated a capacity to handle vendors dynamically by enabling them to create, modify, and remove their profiles. The different routes provide a better way for all users to discover vendors using several methods such as their category or the souk where they are located.
- **Dynamic Product Catalogs with Enhanced Visibility**: The API makes it easier for vendors to create and update product listings with rich details, and the users can access, search and filter them easily, allowing a broader audience to discover their products, and also providing transparency to the products in your marketplace.
- **Enhanced Trust with Review System:** The application enables users to post, explore, and check reviews on both vendors and products which builds transparency and trust into your platform.
- **Transparent Price Negotiation**: The platform enables a clear way for users to make offers to vendors, and also a way for vendors to accept or decline these offers based on their preferences.
- **Personalized Shopping Experiences**: The platform offers personalized recommendations to the users to provide more products that they might like, based on their purchase history and product preferences, leading to a much more effective product discovery.

The API has been tested in a local environment using tools such as Swagger UI and Postman, and it can provide all of the core functionalities that were designed.

9 Conclusion

Our unified API platform, built on FastAPI with a focus on security, successfully bridges the gap in Tunisia's Souk trading landscape. The platform, with an emphasis in empowering both customers and vendors, allows users to discover local products, while enabling vendors to manage their product offerings and connect with customers. The implementation of password hashing and JWT authentication provides a secure and robust authentication layer.

This project empowers customers, fosters growth for providers, and paves the way for a more connected and user-centric mobile ecosystem in Tunisia. I am committed to expanding functionalities, fostering wider adoption, and continuously enhancing security and performance. This platform stands as a testament to the power of innovation, shaping the future of traditional Tunisian markets.

Prospects

The SoukSpot API Platform provides a foundation for future development and expansion. The platform could be extended in many ways such as:

- **Advanced Recommendation Algorithms:** Implement machine learning algorithms to provide more personalized and context-aware product recommendations.
- **Real-time Communication:** Create a method for vendors and customers to have real time chats to discuss offers.
- **Payment Processing:** Include payment processing methods within the platform for a more complete e-commerce functionality.
- **Advanced Search**: Implement more robust search methods to allow more complex queries.

With these methods, the platform will have more features, and also it will be better suited for the current and future market needs.

References

[1] Jeffrey K. Pinto, *Project Management: Achieving Competitive Advantage (5th Edition)*, 2020.

[2] FastAPI Official Documentation. https://fastapi.tiangolo.com/

[3] SQLAlchemy Official Documentation. https://www.sqlalchemy.org/

[4] Python documentation. https://www.python.org

[5] SQLite Official Documentation. https://www.sqlite.org/

[6] Postman Official Website. https://www.postman.com/

[7] Swagger UI Documentation. https://swagger.io/tools/swagger-ui/