Note 41

maximkazakov2005@gmail.com

Contents

1 Introduction

In today's fast-paced digital landscape, developing a backend that is both efficient and secure is paramount, especially for applications in sectors like food and beverage. This document outlines a comprehensive approach to creating a coffee shop application using **FastAPI** and **Firebase**. Through this guide, you will learn not only how to set up your backend but also how to implement key features like menu management and user authorization effectively.

Join us as we delve into the steps necessary to create a fully functional and secure backend for a coffee shop application. You'll see how to manage your drinks menu, implement user roles, and ensure a seamless experience for both your front-end developers and end-users.

2 Overview of FastAPI and Firebase

2.1 FastAPI

- **High Performance**: Built on Starlette for the web parts and Pydantic for the data parts, FastAPI is designed for high performance and fast responses.
- **Ease of Use**: It simplifies the development process with automatic generation of OpenAPI documentation.
- **Type Hints**: Leverages Python type hints to provide better editor support and type validation.

2.2 Firebase

- **Real-time Database**: Offers a NoSQL cloud database that allows for real-time data syncing.
- **Authentication**: Provides robust user authentication systems with custom claims for role management.
- **Hosting**: Easily deploy your applications on Firebase with built-in hosting solutions.

3 Setting Up Your Coffee Shop Application

3.1 Step 1: Seeding Your Database

To start populating your Firebase database with initial data, you need to run a seed script. This is crucial for setting up a foundational menu for your coffee shop application.

Run the Seed Script:

```
python scripts/seed.py
```

Upon running the script, you will be greeted with an engaging message signaling that you're ready to implement the killer feature.

3.2 Initial Menu Data

The following drinks are included in the initial seed data:

```
drinks = [
    {"id": "latte", "name": "Classic Latte", "category": "espresso"},
    {"id": "cappuccino", "name": "Cappuccino", "category": "espresso"},
    {"id": "americano", "name": "Americano", "category": "espresso"},
    {"id": "mocha", "name": "Mocha", "category": "espresso"},
    {"id": "espresso", "name": "Espresso", "category": "espresso"},
    {"id": "cold_brew", "name": "Cold Brew", "category": "brew"},
    {"id": "flat_white", "name": "Flat White", "category": "espresso"},
    {"id": "macchiato", "name": "Macchiato", "category": "espresso"},
    {"id": "irish_coffee", "name": "Irish Coffee", "category": "brew"},
    {"id": "frappuccino", "name": "Frappuccino", "category": "blended"},
]
```

3.3 Database Setup Script

This script populates the `drinks` collection in your Firebase database:

```
for drink in drinks:
    db.collection("drinks").document(drink["id"]).set(drink)

db.collection("config").document("loyalty").set({
    "points_per_dollar": 10,
    "rewards": {"Free Espresso": 200, "Free Any Drink": 500}
})
print("Seeded!")
```

Note: Ensure that the Firebase SDK is properly set up in your project environment.

4 Robust Authorization: Securing Your Application

4.1 Why Authorization Matters

Before you expand your application with additional endpoints, it is critical to establish a solid authorization mechanism. This ensures that sensitive operations are restricted to authorized users only, safeguarding your application against unauthorized access.

4.2 Steps to Implement Strong Authorization

1. **Fix & Perfect the Auth Dependency**

```
from fastapi import Depends, HTTPException, Header, status
from firebase_admin import auth
from typing import Dict, Optional

async def get_current_user(authorization: Optional[str] = Header(None)):
    if not authorization or not authorization.startswith("Bearer "):
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Missing or invalid Authorization header",
            headers={"WWW-Authenticate": "Bearer"},
        )
    token = authorization.split(" ")[1]
    try:
        decoded_token = auth.verify_id_token(token)
        return decoded_token
    except Exception:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Invalid or expired token"
        )
```

2. **Set Yourself as Admin**

   (a) In the Firebase Console:

   - Navigate to **Authentication** → **Users**.
   - Find your test user (the one you log in with).
   - Click the pencil icon → **Add custom claims**.
   - Add the following JSON:

     ```
     {
       "admin": true
     }
     ```

   (b) This sets you as the only admin, allowing you to manage the application effectively.

3. **Implement Dev-Only Endpoint for Admin Rights**

```
@router.post("/dev/make-me-admin")
async def make_me_admin(user = Depends(get_current_user)):
    if settings.ENVIRONMENT != "development":
        raise HTTPException(status_code=404)

    firebase_auth.set_custom_user_claims(user["uid"], {"admin": True})
    return {"message": "You are now admin everywhere"}
```

4. **Protect Every Endpoint**

- **Public Endpoint**:

  ```
  @router.get("/menu/drinks")  # Anyone can see the menu
  ```

- **User Endpoint**:

  ```
  @router.get("/me", dependencies=[Depends(get_current_user)])
  ```

- **Admin Endpoint**:

  ```
  @router.post("/admin/drinks", dependencies=[Depends(get_current_admin)])
  ```

5. **Testing Authorization**:
   - Run your backend and navigate to Swagger UI (`/docs`).
   - Click the **Authorize** button and test your endpoints:
     - Ensure that accessing `/admin/secret` returns a 200 status for admins and a 403 for non-admin users.
     - Verify that unauthenticated requests receive a 401 status.

5 Conclusion

Congratulations on setting up a robust backend for your coffee shop application with FastAPI and Firebase! By following the steps outlined above, you have effectively seeded your database with essential drink information and implemented a strong authorization framework to secure your application.

As you continue to expand your application, keep in mind the importance of maintaining secure coding practices and regular testing to ensure your backend remains a solid foundation for your coffee shop's digital presence.

You are now just a few steps away from having a fully functional coffee shop backend that is both robust and secure. Keep building, and may your coffee shop application thrive! $\cup$