Matheo Jay U. Fabre BSIT-3R1
Applications Development and Emerging Technologies

Difference between Django REST Framework (DRF) and FastAPI (highlight key features, advantages, disadvantages)

Overview

- **Django REST Framework (DRF)**: A powerful and flexible toolkit for building Web APIs on top of the Django framework. It's ideal for projects already using Django for the backend.
- **FastAPI:** A modern, high-performance web framework for building APIs with Python 3.7+ based on standard Python type hints. It's built for speed and developer productivity.

Django REST Framework (DRF)

Key Features

- Built atop Django, offering seamless integration with its ORM and admin interface.
- Provides robust tools for serialization, authentication, and permissions.
- Offers a web-browsable interface for easy testing and debugging.
- Backed by Django's extensive community and third-party packages.

Advantages

- Offers a wide range of built-in features, reducing the need for third-party packages.
- Excellent for building large-scale, full-stack web applications.
- Tight integration with Django’s ORM and admin interface.
- Includes protections against common web vulnerabilities out of the box.

Disadvantages

- Slower performance compared to FastAPI.
- Synchronous nature may lead to slower performance under heavy load compared to asynchronous frameworks.
- May be heavyweight for simple API services or microservices architectures.

FastAPI

Key Features

- Modern, high-performance web framework built on Starlette and Pydantic.
- Asynchronous support using async/await.
- Automatic generation of interactive API docs (Swagger/OpenAPI)
- Type hints used for validation and documentation.

Advantages

- Extremely fast, on par with Node.js and Go.
- Automatic validation and serialization using Python type hints.
- Easy to write and maintain clean, self-documenting code.
- Ideal for microservices and real-time applications

Disadvantages

- Lacks some out-of-the-box functionalities like an admin interface, requiring additional setup.
- Smaller ecosystem and younger community compared to Django.
- Less suitable for large monolithic applications out of the box.

Challenges Encountered Using Django REST Framework (DRF) and How They Were Solved

1. **CORS Issues**

    - Problem: React frontend couldn’t communicate with the DRF backend due to CORS policy restrictions.

    - Solution: Installed and configured django-cors-headers in the Django settings

2. **POST/PUT Requests Failing**

    - Problem: Frontend POST and PATCH requests failed due to CSRF token issues or improper payloads.

- Solution: Used @csrf_exempt on views when necessary (for testing/local dev).
- Used DRF's APIView or ViewSet which handle CSRF more smoothly.

3. **Deployment on Render**

   - Problem: Deployment to Render.com failed due to environment variable misconfigurations or missing build steps.

   - Used WhiteNoise for static file handling
   - Set ALLOWED_HOSTS, DEBUG=False, and proper database URL in settings.py

4. **Gunicorn/WSGI + Nginx Setup**

   - Problem: App failed to run on server after deployment due to misconfigured WSGI/Gunicorn.

   - Solution: Carefully followed Gunicorn setup, ensured wsgi.py was correctly pointing to Django app

5. **Static Files Not Loading**

   - Problem: Static files (e.g., admin styles) were not showing in production.

   - Solution: Configured STATIC_ROOT and ran collectstatic. Updated Nginx/Apache to serve static files correctly.

Challenges Encountered Using FastAPI and How They Were Solved

1. **Setting Up the FastAPI Backend Structure**

   - Problem: FastAPI is flexible and doesn’t enforce a specific project structure.

   - Solution: Followed a modular structure by separating routers, models, schemas, and database logic.

2. **Connecting FastAPI to a Database**

- Problem: Connecting to SQLite and PostgreSQL with SQLAlchemy and managing sessions wasn't straightforward at first.

- Solution: Set up SQLAlchemy engine and session using database.py.
- Created models and linked them to the database tables.

3. **CORS Errors from Frontend**

- Problem: When integrating with your React frontend, you faced Cross-Origin Resource Sharing (CORS) issues.

- Solution: Installed fastapi.middleware.cors.CORSMiddleware
- from fastapi.middleware.cors import CORSMiddleware
- app.add_middleware(
-     CORSMiddleware,
-     allow_origins=["*"],
-     allow_credentials=True,
-     allow_methods=["*"],
-     allow_headers=["*"],
-)

4. **FastAPI Deployment**

- Problem: After deploying my React frontend along with the FastAPI backend, the user interface (UI) didn't render as expected. Instead of loading the React app, visiting the URL showed only JSON data from the backend or a "404 Not Found" page when refreshing routes handled by React Router.

- Solution**:** I created a _redirects file in the public folder of the React app
- /*   /index.html   200
- to redirect all paths to index.html, allowing React Router to handle them properly