Dou Issa Steffi E. Udasco
BSIT-3R1

Difference Between DRF and FastAPI

Feature	Django REST Framework (DRF)	FastAPI
Framework Type	Based on Django – full-stack web framework	Lightweight – built on Starlette & Pydantic
Speed/Performance	Slower due to synchronous nature	High performance (asynchronous & fast)
Ease of Use	Easy for Django users; steeper learning for beginners	Simple and beginner-friendly with modern Python
Type Checking & Validation	Uses serializers	Leverages Python type hints (Pydantic)
Documentation Generation	Manual or via third-party packages	Auto-generates Swagger & ReDoc by default
Community & Maturity	Large, mature community	Newer but growing fast
Authentication	Built-in with sessions, tokens, and third-party plugins	Requires integration or manual setup
Use Case	Ideal for complex projects with ORM and admin support	Great for APIs, microservices, and speed-focused apps

Development and Deployment Challenges
Django REST Framework (DRF)

Challenges:

- Serializer Complexity: Understanding and writing custom serializers felt verbose.
- Authentication Integration: Integrating third-party OAuth or JWT required extra configuration.
- Routing: DRF routers and views could be unintuitive initially.
- Deployment: Larger dependencies and slower performance on limited-resource platforms.

Solutions:

- Referred to official docs and used `ModelSerializer` for ease.

- Used `djoser` and `SimpleJWT` for simplified auth handling.
- Created custom API views when `ViewSet` became too limiting.
- Deployed using platforms like Heroku with `Gunicorn` and `Whitenoise`.

FastAPI

Challenges:

- CORS Issues: Encountered CORS errors when connecting frontend (React).
- Data Validation: Initially confusing to format requests using Pydantic models.
- Async/Await Usage: Had to learn and adapt to asynchronous programming.
- Deployment: Running Uvicorn reliably on Render or other hosts took tweaking.

Solutions:

Added `fastapi.middleware.cors` with `allow_origins=["*"]` for development.
Carefully matched request formats with `TodoCreate` and `TodoUpdate` models.
Studied `async def` patterns and followed best practices from docs.
Used Uvicorn with Gunicorn or Docker for stable deployment.