Lumay, Alyza Shane C.

IT3R5

Performance Innovative Task:

Building a Full-Stack To-Do List Application with FastAPI Documentation

DRF vs FastAPI: Comparison

Feature	**Django REST Framework (DRF)**	**FastAPI**
Framework Base	Built on Django (batteries-included web framework)	Built on Starlette (lightweight ASGI framework)
Speed	Slower (sync-based by default)	Very fast (async support out of the box)
Type Hints / Validation	Optional, uses Django forms/serializers	Mandatory, uses Python type hints & Pydantic
Learning Curve	Steeper, especially for beginners	Easier for those familiar with Python type hints
Admin Panel	Built-in Django admin	No built-in admin panel
ORM	Django ORM (mature, feature-rich)	Can use SQLAlchemy or others (flexible but manual)
Auto Docs	Basic (via third-party like drf-yasg or coreapi)	Swagger UI & Redoc auto generated by default
Community & Ecosystem	Large and mature	Growing rapidly, but smaller than DRF
Best Use Cases	Enterprise-level projects, CMS, complex admin apps	Fast APIs, microservices, modern async apps

Advantages

DRF

- Mature ecosystem with lots of extensions
- Integrated with Django's ORM and admin
- Great for projects with complex models & authentication needs

FastAPI

- Much faster performance (async I/O)
- Cleaner code with automatic data validation (Pydantic)
- Instant interactive API docs (Swagger & ReDoc)
- Ideal for microservices and modern web apps

Disadvantages

DRF

- Slower response time due to synchronous nature
- Verbose code with serializers and views
- Less intuitive docs compared to FastAPI

FastAPI

- No built-in admin or user system
- Smaller ecosystem for advanced features
- Requires manual setup for things DRF does out of the box (auth, admin, etc.)

CHALLENGES:

1. Backend and Frontend Connection Issues

- **Challenge:** Integration problems between your backend (DRF/FastAPI) and frontend can lead to miscommunication or errors in API consumption.
- **Solution:**
    - **Validate API Endpoints**: Use tools like Postman or Swagger to ensure your backend APIs are working as intended.
    - **CORS Configuration**: Enable Cross-Origin Resource Sharing (CORS) in your backend to ensure the frontend can access the APIs. Both DRF and FastAPI have easy-to-use libraries for this.
    - **Frontend Debugging**: On the frontend side, inspect network requests in the browser's DevTools to ensure correct URLs and payloads are sent.

2. Error Handling (Many Errors)

- **Challenge:** Errors could arise due to mismatched configurations, faulty logic, or dependency issues.
- **Solution:**
    - **Log Details**: Enable verbose logging in both DRF and FastAPI to track errors systematically.
    - **Error Tracking Tools**: Use tools like Sentry to monitor and identify recurring errors.
    - **Unit Tests**: Write tests for both backend and frontend to catch issues early in the development cycle.

3. Accidental Deletion of Backend Repository

- **Challenge:** Losing your backend repository can be a major setback.
- **Solution:**
    - **Version Control**: Always use Git for version control and frequently push to a remote repository (e.g., GitHub, GitLab).
    - **Backup**: Enable repository backup services or keep local copies in a safe location.

    - **Recovery**: If deletion occurs, check if the repository hosting service has recovery options (e.g., GitHub’s Recycle Bin feature).

4. GitHub Pages Deployment Issues

- **Challenge:** Missing gh-pages branch and incorrect page display during deployment.
- **Solution:**
    - **Manually Create** gh-pages **Branch**: As you've already done, create the branch manually if automatic deployment isn't set up.
    - **GitHub Actions for Automation**: Use GitHub Actions workflows to automate the deployment process. Here's an example workflow:
    - **Double-Check Repository Settings**: Ensure correct branch is selected in the GitHub Pages settings under "Source."