# Developer’s Guide: Fund Finder Backend

## Introduction

Welcome to the Fund Finder Backend project! This guide provides a comprehensive overview of the development setup, branching strategy, testing procedures, and best practices. It is intended to help new and existing developers understand the workflow and contribute effectively to the project.

## Development Setup

The development environment is configured using Docker and follows a consistent structure to enable quick onboarding and reliable testing. Below are the details of the setup and instructions to run the project.

### Prerequisites

Ensure you have the following tools installed on your local machine:

**- Git  
- Docker & Docker Compose  
- Python (optional, for debugging and running Django commands manually)**

### Steps to Run the Setup

1. Clone the repository:

```bash  
 git clone <repository-url>  
 cd fund\_finder\_backend  
 ```

2. Create an environment file from the example template:

```bash  
 cp envs/.env.example envs/.env.dev  
 ```

3. Update the `.env.dev` file with the necessary values.

4. Build and run the Docker containers:

```bash  
 docker-compose -f docker/compose.dev.yml up --build  
 ```

5. Access the application:

- Backend: [http://localhost:8080](http://localhost:8080)

- Admin Panel: [http://localhost:8080/admin](http://localhost:8080/admin)

## Branching Strategy

We follow the Git Flow branching model to ensure smooth collaboration and streamlined deployments. Below is an overview of the branching strategy:

- \*\*dev\*\*: Active development happens here. Developers create feature branches off this branch.  
- \*\*staging\*\*: Pre-production testing environment. Features from `dev` are merged here after review.  
- \*\*prod\*\*: Live production code. Only thoroughly tested code from `staging` is merged here.  
- \*\*Feature Branches\*\*: For individual features or fixes. E.g., `feature/add-grant-api`.  
- \*\*Release Branches\*\*: For preparing releases. E.g., `release/v1.0`.  
- \*\*Hotfix Branches\*\*: For critical production fixes. E.g., `hotfix/fix-crash-bug`.

## Testing Procedures

Testing is an integral part of the development workflow. Below are the types of tests and their purposes:

### Types of Tests

- \*\*Unit Tests\*\*: Validate individual components (e.g., functions, models).  
- \*\*Integration Tests\*\*: Ensure components work together (e.g., API endpoints).  
- \*\*System Tests\*\*: Test the application end-to-end.  
- \*\*User Acceptance Tests (UAT)\*\*: Verify that the application meets business requirements.  
- \*\*Performance Tests\*\*: Test speed and scalability under load.  
- \*\*Security Tests\*\*: Check for vulnerabilities like SQL injection and XSS.

## Best Practices

- \*\*Write Unit Tests\*\*: Ensure at least 80% code coverage for critical modules.  
- \*\*Automate Testing\*\*: Use CI/CD pipelines to automate unit and integration tests.  
- \*\*Code Reviews\*\*: Submit pull requests (PRs) for peer review before merging.  
- \*\*Environment Variables\*\*: Do not hardcode sensitive values; use `.env` files.  
- \*\*Database Migrations\*\*: Always run migrations when modifying the database schema.  
- \*\*Follow Code Standards\*\*: Adhere to PEP 8 and project-specific guidelines.

## Code Review Guidelines

Code reviews are mandatory for every pull request. Reviewers should:

- Check for code readability and adherence to standards.  
- Verify adequate test coverage.  
- Identify potential performance or security issues.  
- Ensure proper documentation of new functionality.

## Workflow Summary

1. Create a feature branch from `dev`.  
2. Write code and tests.  
3. Submit a pull request to `dev` for review.  
4. Merge into `staging` after successful review and testing.  
5. Deploy to production after thorough testing in `staging`.

## Closing Note

This guide serves as a comprehensive reference for the development workflow. If you encounter any issues or have questions, consult the team lead or refer to the project's README for additional details.