## *Backend Documentation*

The **Emissions backend** is a *Django-based application* designed to manage, store, and expose emission data for various countries, activities, years, and emission types. The project is structured around a **clean architecture** that separates concerns into **domain**, **infrastructure**, and **use cases** layers. The **domain layer** contains core business logic and entities, such as the `Emission` class, and abstract repository interfaces that define how data should be accessed without tying the code to a specific database implementation. The **infrastructure layer** includes Django models, serializers, views, and repository implementations that interact directly with the database and the REST API. The **use cases layer** orchestrates business rules, handling complex queries and filtering operations. This separation ensures *modularity*, *testability*, and *maintainability*, allowing developers to add features or swap implementations without breaking the core logic.

The project follows a **well-organized folder structure** to reinforce this separation. At the top level, `manage.py` handles Django management commands, while `docker-compose.yml` and `Dockerfile` manage containerization. The `emissions_project` folder contains the core Django app: **domain/** defines entities and repository interfaces, **infrastructure/** includes models (`models.py`), repository implementations (`repository_impl.py`), serializers (`serializers.py`), views (`views.py`), migrations, and management commands such as `seed_emissions.py`. The **usecases/** folder contains application logic like `get_emissions.py`, and the **tests/** folder contains unit tests for domain entities, API endpoints, repository functionality, and use cases. Supporting files like `settings.py`, `urls.py`, and `wsgi.py` configure Django settings, routing, and server interfaces. This organization allows developers to navigate the codebase intuitively and locate responsibilities according to the architecture.

The project is built with **Python 3.8**, **Django 4.2**, and **SQLite3**, and it uses **Docker** and **Docker Compose** to provide a consistent development environment. Docker containers allow running the backend, seeding the database with sample emissions, and executing commands in isolation, ensuring reproducible setups across machines. Developers can seed the database with the `seed_emissions` management command and verify data using the Django shell. SQLite provides simplicity in development, and its integration with Docker ensures persistence of the `db.sqlite3` file while keeping the system lightweight. Although SQLite is used here, the architecture is *database-agnostic*, and other relational databases can be integrated for production with minimal changes.

Testing and development practices are a key focus. The project contains four test modules: `test_domain_entities.py` for domain objects, `test_emissions_api.py` for REST API endpoints, `test_infrastructure_repository.py` for repository functionality, and `test_usecases_get_emissions.py` for application logic. Tests run inside Docker, creating isolated databases for reliability. This supports **TDD (Test-Driven Development)**, ensuring code correctness and maintainability. The project also follows **SOLID principles** and design patterns such as the **repository pattern** for decoupling domain from persistence, the **strategy pattern** for flexible filtering, and **use cases** for orchestrating domain objects. REST API endpoints allow querying emissions by multiple filters like year, country, or emission type. Overall, the *modular architecture*, *clear folder structure*, *Dockerization*, and thorough testing make the backend **robust, maintainable, and ready for future expansion or production deployment**.