FastAPI URL Shortener Project – Detailed Technical Documentation

This document explains the FastAPI-based URL Shortener project in detail. It is written for a software engineer who is new to backend development and FastAPI. Every component, concept, and function used in the project is explained step by step.

1. What Is This Project?

This project is a backend service that converts long URLs (such as YouTube links) into short, shareable URLs. When a user opens a short URL, the service redirects them to the original URL and keeps track of how many times the link was clicked.

Core features:

- Create a short URL from a long URL
- Redirect users from short URL to original URL
- Track the number of clicks for each short URL
- Expose REST APIs using FastAPI

2. High-Level Architecture

The project follows a standard backend architecture: Client → FastAPI Server → Database (MySQL running in Docker).

FastAPI handles HTTP requests and responses. SQLAlchemy is used as the ORM (Object Relational Mapper) to communicate with the MySQL database. Docker is used to run MySQL in an isolated and reproducible environment.

3. Project Structure Explained

The project is structured in a clean, scalable way commonly used in real backend systems.

```
url-shorten-api/ ■■■ app/ ■ ■■■ api/ → HTTP routes (endpoints) ■ ■■■ core/ → configuration and settings ■ ■■■ db/ → database connection and base classes ■ ■■■ models/ → database models (tables) ■ ■■■ schemas/ → request & response validation (Pydantic) ■ ■■■ main.py → FastAPI application entry point ■■■ create_tables.py → one-time DB table creation script ■■■ compose.yml → Docker configuration for MySQL ■■■ requirements.txt → Python dependencies ■■■ .env → environment variables ■■■ venv/ → Python virtual environment
```

4. FastAPI Basics

FastAPI is a modern Python web framework used to build APIs. It is fast, easy to use, and automatically generates API documentation.

Key FastAPI concepts used in this project:

- FastAPI(): creates the application instance
- APIRouter: groups related routes together
- Dependency Injection: automatically provides database sessions

- Pydantic models: validate request and response data

5. Database and SQLAlchemy ORM

The project uses MySQL as the database. SQLAlchemy acts as the ORM, which means we define Python classes that map directly to database tables.

Why ORM is used:

- Avoid writing raw SQL everywhere
- Database-agnostic code
- Cleaner, safer queries

6. Link Model (Database Table)

The Link model represents one shortened URL stored in the database.

`id → Primary key code → Short unique string (used in URL) original_url→ Long URL (YouTube, etc.) clicks → Number of times the short URL was opened created_at → Record creation timestamp updated_at → Last update timestamp`

Each field is annotated using SQLAlchemy 2.0 typing (Mapped[T]) to make the intent explicit and type-safe.

7. API Endpoints Explained

POST /shorten

This endpoint accepts a long URL and returns a generated short URL. Internally, a random short code is generated and stored in the database.

GET /{code}

This endpoint handles redirection. When a user opens a short URL, the server looks up the code, increments the click counter, and redirects the user to the original URL using an HTTP 307 redirect.

GET /stats/{code}

This endpoint returns statistics about a short URL, including the number of clicks recorded so far.

8. Click Tracking Logic

Click tracking is implemented by incrementing a counter every time the redirect endpoint is accessed. This is stored persistently in the database.

9. Why Docker Is Used

Docker allows running MySQL without installing it directly on your system. It ensures consistent behavior across machines and avoids environment conflicts.

10. How to Run the Project (Summary)

`1. Start MySQL using Docker Compose 2. Activate Python virtual environment 3. Install dependencies 4. Create database tables 5. Start FastAPI using Uvicorn`

Once running, FastAPI automatically provides interactive API documentation at /docs, which is extremely useful for learning and testing.

11. Conclusion

This project demonstrates real-world backend fundamentals: REST APIs, database modeling, Docker usage, ORM concepts, and request handling. It forms a strong foundation for more advanced backend systems.