Backend Documentation

2025G2

October 2025

1 Introduction

The backend uses **SQLAlchemy** as the Python SQL toolkit and Object Relational Mapper (ORM). This allows us to work with the database using Python classes and objects instead of hardcoded SQL queries. By doing so, data retrieved from the database can be handled as Python objects rather than tuples or dictionaries, providing a cleaner and more intuitive syntax.

2 Database Driver

We use **psycopg3** as our PostgreSQL driver. A driver is a software component that enables communication between the application and the database system. In this context, **psycopg3** acts as the bridge between Python (through SQLAlchemy) and the PostgreSQL server, translating SQLAlchemy's commands into actual SQL understood by the database.

The newest version of psycopg (**psycopg3**) supports both *synchronous* and *asynchronous* modes. In asynchronous mode, queries can be executed concurrently without blocking, allowing for more efficient handling of multiple database operations—useful for concurrent operations and event-driven applications like ours.

3 Environment Variables

We use a `.env` file to store sensitive information such as database usernames, passwords, and connection details. This keeps secret credentials out of the source code, which is important both for security and for flexibility when moving between different environments (for example, development and production).

The file is loaded automatically at runtime using the `python-dotenv` package. Once loaded, the variables can be accessed in Python through `os.getenv()`, which allows us to build the database connection string dynamically without exposing the actual credentials.

An example of a typical `.env` file:

```
DB_USER=atle
DB_PASSWORD=secretpassword
DB_HOST=localhost
DB_NAME=mydatabase
```

4 Database Connection Setup

This section documents the process of establishing a connection to the remote PostgreSQL database server used in this project.

4.1 Prerequisites

4.1.1 VPN Connection

Access to the PostgreSQL database server requires an active VPN connection to the university network.

4.1.2 PostgreSQL Installation

While the database server is remote, a local PostgreSQL client installation is required to test connections and interact with the database from the command line. On macOS, PostgreSQL can be installed via Homebrew:

```
brew install postgresql@14
brew services start postgresql@14
```

Note: The local PostgreSQL server is not required for the application to function, but is useful for development and testing purposes.

4.2 Connection Parameters

The database server uses non-standard connection parameters:

- **Host:** `138.100.82.184` (IP address)
- **Port:** `2345` (non-standard port)
- **Username:** `lectura`
- **Database:** Multiple databases accessible (see Section 4.4)

The hostname `apiict00.etsii.upm.es` resolves to `138.100.82.170`, which does not accept connections. However, phpPgAdmin shows the server running on `138.100.82.184:2345`. Therefore, we use the direct IP address.

The port `2345` is used instead of the default PostgreSQL port `5432`. This is likely a security measure to reduce automated attacks on the standard port, or to allow multiple PostgreSQL instances on the same server.

4.3 Accessing PostgreSQL from Terminal

4.3.1 Basic Connection Command

To connect to the database from the command line:

```
psql -h 138.100.82.184 -U lectura -d <database_name> -p 2345
```

Where `<database_name>` is one of the accessible databases (e.g., `postgres`, `1245`, or `2207`).

4.3.2 Useful PostgreSQL Prompt Commands

Once connected to the database (indicated by the `database_name=>` prompt), the following commands are useful:

Command	**Description**
`\l`	List all databases on the server
`\dt`	List all tables (relations) in the current database
`\d <table_name>`	Describe the structure of a specific table
`\du`	List all users/roles
`\conninfo`	Display current connection information
`\c <database>`	Connect to a different database
`\q`	Quit the PostgreSQL prompt
`q`	Quit the current prompt

Table 1: Useful PostgreSQL prompt commands

4.4 Accessible Databases

Using the command `\l`, we determined that the `lectura` user has CONNECT privileges (`c`) on the following databases:

- `1245` – 51 GB database (Access: `lectura=c/ncorrea`)
- `2207` – 32 GB database (Access: `lectura=c/ncorrea`)
- `postgres` – Default administrative database

The notation `lectura=c/ncorrea` indicates that the user `lectura` has been granted CONNECT permission (`c`) by the user `ncorrea`.

4.5 Testing the Connection

The FastAPI application includes a status endpoint at `/api/status` that tests the database connection by executing `SELECT version()` and returns the PostgreSQL version information if successful. This provides a quick way to verify that the application can communicate with the database server.

4.6 API Connection Status

The following output confirms the successful connection to the database:

```
{
  "status": "API is running",
  "database": "Connected",
  "version": "('PostgreSQL 14.17 (Debian 14.17-1.pgdg120+1) on x86_64-pc-linux-gnu,
  compiled by gcc (Debian 12.2.0-14) 12.2.0, 64-bit',)"
}
```