## 🧱 Step 1: Create Your Project Folder

In your terminal, run:

mkdir server

cd server

## 🐍 Step 2: Set Up a Virtual Environment

Create a virtual environment:

python -m venv venv

### 🔑 Activate the Environment

First, check which terminal you’re using:

echo $0

If it shows one of these:

* bash, zsh, or sh → you’re on **Linux/Mac** or **Git Bash**
* cmd.exe or powershell.exe → you’re on **Windows**

Now activate accordingly:

#### ▶️ On Windows:

source venv/Scripts/activate

#### ▶️ On Mac/Linux:

source venv/bin/activate

## 📦 Step 3: Install Dependencies

Install the main dependencies:

pip install fastapi uvicorn sqlalchemy psycopg2-binary pydantic-settings

### 📘 What Each Package Does

| **Package** | **Description** |
| --- | --- |
| **FastAPI** | The main web framework |
| **Uvicorn** | ASGI server to run FastAPI |
| **SQLAlchemy** | ORM for database models and queries |
| **psycopg2-binary** | PostgreSQL database adapter |
| **pydantic-settings** | Simplifies environment variable and config management |

## 🗂️ Step 4: Project Structure

Here’s the recommended folder layout for your FastAPI backend:

server/

└── app/

├── main.py

├── config/

├── controllers/

├── models/

├── routes/

├── schemas/

├── services/

└── utils/

**Server>config>database\_config.py:**

from sqlalchemy import create\_engine, text

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy.orm import sessionmaker

from sqlalchemy.exc import SQLAlchemyError, OperationalError

from dotenv import load\_dotenv

import os

import re

import time

## 2️⃣ Importing libraries

from sqlalchemy import create\_engine, text

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy.orm import sessionmaker

from sqlalchemy.exc import SQLAlchemyError, OperationalError

from dotenv import load\_dotenv

import os

import re

import time

* **sqlalchemy** → main library for talking to databases in Python
  + create\_engine → creates a connection to your database
  + declarative\_base → base class for defining tables (models)
  + sessionmaker → creates sessions to interact with DB
  + SQLAlchemyError, OperationalError → catch database errors
* **dotenv** → loads environment variables from a .env file
* **os** → read environment variables from the system
* **re** → regex module, used for masking passwords
* **time** → (not really used in this version, can be ignored)

## 3️⃣ Load environment variables

load\_dotenv()

* Reads your .env file so os.getenv("VAR\_NAME") can access the variables.
* Example .env file:
* DATABASE\_URL=postgresql://user:pass@localhost/chemistry\_db
* DOCKER\_ENV=false

## 4️⃣ Base class for models

Base = declarative\_base()

* Base is a **parent class** for all your database tables.
* Every table you define in SQLAlchemy will inherit from Base.
* Example:

class Compound(Base):

\_\_tablename\_\_ = "compounds"

id = Column(Integer, primary\_key=True)

name = Column(String)

## 5️⃣ Get database URL

def get\_database\_url():

    database\_url = os.getenv("DATABASE\_URL" , "")

* This function figures out **what database to connect to**.
* Returns a **string** like "postgresql://user:pass@localhost/dbname".

### Inside get\_database\_url():

database\_url = os.getenv("DATABASE\_URL", "")

* Reads the environment variable DATABASE\_URL.
* If it’s not set, returns an empty string.

def get\_database\_url():

    database\_url = os.getenv("DATABASE\_URL" , "")

    if not database\_url:

        is\_docker = os.environ.get("DOCKER\_ENV") == "true"

        default\_host = "mysql" if is\_docker else "localhost"

        database\_url = os.getenv("SQLALCHEMY\_DATABASE\_URL" ,

            f"mysql+pymysql://root:password@{default\_host}:3306/chemistry\_app\_db"

        )

        return database\_url

* If DATABASE\_URL is **empty**, it tries:
  1. Check if running in Docker (DOCKER\_ENV=true)
  2. Set default host: mysql (Docker) or localhost (local machine)
  3. Use SQLALCHEMY\_DATABASE\_URL if set, otherwise a default MySQL URL
* Then it **returns** that URL immediately.

# Standardize driver format for MySQL

    if 'mysql://' in database\_url and 'mysql+pymysql://' not in database\_url:

        database\_url = database\_url.replace('mysql://', 'mysql+pymysql://')

    # Standardize driver format for PostgreSQL

    if 'postgres://' in database\_url and 'postgresql://' not in database\_url:

        database\_url = database\_url.replace('postgres://', 'postgresql://')

* Fixes database URLs so SQLAlchemy understands them.
* Example:
  + mysql://root:pass@localhost/db → mysql+pymysql://root:pass@localhost/db

    # Fix hostname for local development

    is\_docker = os.environ.get('DOCKER\_ENV') == 'true'

    if 'mysql' in database\_url and not is\_docker and '@mysql:' in database\_url:

        database\_url = database\_url.replace('@mysql:', '@localhost:')

    return database\_url

* If the URL says @mysql: but we’re **not in Docker**, replace it with @localhost:.
* Ensures local development works.

## 6️⃣ Create database engine

def create\_db\_engine(database\_url):

"""

Create and configure a SQLAlchemy engine with proper error handling.

"""

* This function **connects your Python app to the database**.
* Returns an engine object used to run queries.

        masked\_url = mask\_db\_password(database\_url)

        print(f"Connecting to database at: {masked\_url}")

* mask\_db\_password() hides the password in logs.
* Example:
* postgresql://user:\*\*\*\*\*\*@localhost/db

engine\_kwargs = {

"pool\_pre\_ping": True,

"pool\_recycle": 3600,

}

if 'mysql' in database\_url:

engine\_kwargs["connect\_args"] = {"connect\_timeout": 15}

engine = create\_engine(database\_url, \*\*engine\_kwargs)

return engine

* pool\_pre\_ping=True → checks if DB connection is alive
* pool\_recycle=3600 → recycles connections every hour
* MySQL-specific connect\_timeout → 15 seconds
* create\_engine() → creates the actual connection object

### Error handling

except Exception as e:

print(f"Error creating database engine: {str(e)}")

print("Attempting to create engine with basic configuration")

...

* If something fails, try a **simpler connection**
* Last fallback: use a **local SQLite file**
* Ensures the app doesn’t crash immediately

## 7️⃣ Mask database password

def mask\_db\_password(database\_url):

"""

Mask the password in a database URL for safe logging.

"""

return re.sub(r'(://[^:]+:)[^@]+(@)', r'\1\*\*\*\*\*\*\2', database\_url)

* Replaces the password in the URL with \*\*\*\*\*\*
* Safe to print in console

## 8️⃣ Initialize engine and session

SQLALCHEMY\_DATABASE\_URL = get\_database\_url()

engine = create\_db\_engine(SQLALCHEMY\_DATABASE\_URL)

SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

* SQLALCHEMY\_DATABASE\_URL → final URL used
* engine → connection object
* SessionLocal → **factory** for database sessions

## 9️⃣ Provide DB session to FastAPI

def get\_db():

db = SessionLocal()

try:

yield db

finally:

db.close()

* Each API request calls get\_db()
* yield db → gives a session for the request
* finally: db.close() → closes the session automatically

✅ **Summary in plain English**

1. Load environment variables (.env)
2. Figure out which database URL to use
3. Fix the URL so SQLAlchemy understands it
4. Create a database engine (connection)
5. Create a session factory (SessionLocal)
6. Provide a function (get\_db) to get a session safely for each request