ARIF AQEEL AHMAD

Arifaqeel1233@gmail.com

FLASK

ARIF AQEEL AHMAD

***COMPLETE PYTHON DEVELOPMENT JOURNEY***

Contents

[INTRODUCTION TO FLASK 2](#_Toc211520304)

[First Application 2](#_Toc211520305)

[*Application Factory:* 3](#_Toc211520306)

[**Connection to Database** 3](#_Toc211520307)

# INTRODUCTION TO FLASK

Following steps will be followed in the creating the flask project:

* First of all, create a virtual environment
* Activate that environment
* Purpose of doing it, is that we can do work with different versions and every time we will create a new virtual environment for each project

Flask depends on the Werkzeug WSGI toolkit (the standard Python interface between applications and servers.), the Jinja template engine (template language that renders the pages your application serves.), and the Click CLI (framework for writing command line applications), MarkupSafe comes with Jinja, it escapes the unsafe input. ItsDangerous 🡪 sed to protect Flask’s session cookie.

# First Application

* Create a file outside of virtual environment 🡪 app.py
* In this file: from flask(module) import Flask (class)
* Create object of class named app usually
* \_\_name\_\_ is a special built-in variable in Python.
* It tells Flask where your application is located.
* If you run your script directly (e.g., python app.py), then \_\_name\_\_ will be "\_\_main\_\_". This is needed so that Flask knows where to look for resources such as templates and static files.
* To run file/project: flask --app hello run, if the name of file is “app” then command will be:🡪flask app run
* @app.route('/user/<variable name>')
* @app.route(‘/user/<int: user\_id>’)

|  |  |
| --- | --- |
| string | (default) accepts any text without a slash |
| int | accepts positive integers |
| float | accepts positive floating point values |
| path | like string but also accepts slashes |
| uuid | accepts UUID strings |

***NOTE***:

* Instead of creating a Flask instance globally (at the top of code in app.py), you will create it inside a function. This function is known as the application factory.

# *Application Factory:*

* Inside flaskr 🡪 create \_\_init\_\_.py
* \_\_init\_\_.py serves the 2 functions :-
  + - * this is act as application factory(function)
      * it tells Python that the flaskr directory should be treated as a package.
* def create\_app(test\_config=None)🡪🡪🡪test\_config is a optional parameter. It allows you to pass different configurations when testing your app.

app = Flask(\_\_name\_\_ , instance\_relative\_config=True)

* \_\_name\_\_ tells Flask where the app is located
* **instance\_relative\_config=True** → tells Flask that configuration files are **relative to the “instance” folder**, not the main project folder.
* The ***instance folder*** is located outside the flaskr package and can hold local data that shouldn’t be committed to version control, such as configuration secrets and the database file.

# **Connection to Database**

(g)🡪a special object used to store data during a request.

VISIT IT FOR MORE INFO: <https://flask.palletsprojects.com/en/stable/tutorial/database/>

* After creating db connection, now we will add schema file in flaskr(define all tables)
* Add the Python functions that will run these SQL commands to the db.py file:

#### ✅ Explanation:

This line teaches SQLite how to \*\*convert a text value from the database into a Python `datetime` object\*\*.

Let’s break it:

- `sqlite3.register\_converter("timestamp", ...)`

→ Registers a function that converts database column type `"timestamp"` into a Python object.

- `"timestamp"` → The name of the SQL column type you want to handle.

(For example, in your SQL you might have: `created TIMESTAMP`)

- `lambda v: datetime.fromisoformat(v.decode())` →

A small “inline” function that:

1. Takes the value `v` (which is in bytes)

2. Decodes it to a normal string

3. Uses `datetime.fromisoformat()` to turn `"2025-10-17T20:00:00"` into a real Python `datetime` object.

✅ Why it matters:

Without this, timestamps would be just plain text when fetched from the database.

With this, Python automatically turns them into usable `datetime` objects (so you can compare, format, etc.).

---

## 🧠 Full Process Recap

| Step | Function/Code | What Happens |

|------|----------------|--------------|

| 1 | `get\_db()` | Connects to your SQLite database |

| 2 | `open\_resource('schema.sql')` | Opens your SQL schema file |

| 3 | `db.executescript()` | Executes the SQL to create tables |

| 4 | `@click.command('init-db')` | Adds a terminal command called `flask init-db` |

| 5 | `init\_db\_command()` | Runs the initialization when the command is called |

| 6 | `register\_converter("timestamp", ...)` | Automatically converts timestamp columns to Python datetime |

---

## 🧩 How it fits in the Flask project

Later in your \*\*`create\_app()`\*\* (app factory function), Flask will register this command like:

```python

app.cli.add\_command(init\_db\_command)