# First step

## 1- Creating the project folder:

pass\_manager/

my\_pass\_project/

# contain the actual project files

documentation/

#inside of the my\_pass\_projecct we make this dir contains a word file to document what we do

## 2- Creating the virtual environment:

In the (my\_pass\_ project /):

[python -m venv venv]

# Create the venv

[source venv/Scripts/activate]

# activate the venv

## 3- Create the .gitignore and echo these:

In the (my\_pass\_ project /):

[echo "\_\_pycache\_\_" >> .gitignore]

# ignore the cache that python make for the scripts

[echo "\*.pyc" >> .gitignore]

# python save the cache in a .pyc files so we ignore them all

[echo "venv/" >> .gitignore]

# ignore the virtual environment dir

[echo ".env" >> .gitignore]

# ignore the environment variables which could contain sensetive data

## 4- Initialize Git Repo:

In the (my\_pass\_ project /):

[git init]

# initialize the local repo

[git add .]

# add all of the files to the staging area

[git commit -m “initial project with venv and .gitigonre]

# commit to the repo with a comment attached

###### What we currently have?

* a project directory (pass\_manager)
* in that directory we have my\_pass\_project/ and inside it is the documentation/
* in the (my\_pass\_project) a virtual environment (venv)
* a .gitignore file which includes (\_\_pychache\_\_, \*.pyc, venv/, .env)

# Initialize the database

### 1- create the database dir and databse files

in the (my\_pass\_project) dir:

[mkdir data]

[cd data]

[touch passwords.db]

# this is follow the separate of concern principle

### 2- ignore the data dire

in the (my\_pass\_project) dir:

[echo “data/” >> .gitignore]

# so that the data won’t get pushed for everyone to see it

### 3- database schema:

We will have 3 entities (tables):

[users, sites, passwords]

| **Column** | **Type** | **Description** | **Notes** |
| --- | --- | --- | --- |
| id | INTEGER | Primary key, auto-incremented unique id | Uniquely identifies each password record |
| user\_id | INTEGER | Foreign key linking to the user table | Identifies who owns this password |
| site\_id | INTEGER | Foreign key linking to the sites table | Identifies the website or service |
| username | TEXT | The username for the site | Useful when a site allows multiple usernames |
| password | TEXT | The password stored (encrypted later) | The actual password value |
| created\_at | TEXT | Timestamp of when password was created | Format: ISO 8601 datetime string |
| is\_backup | INTEGER | Flag to mark backup or active password | 0 = active, 1 = backup |

### 4- the is\_backup strategy:

We wil save the current password and the last used password:

**Step 1: Check for existing backup for this user and site**

* + - * Query if there is a password with is\_backup = 1 for the same user and site.
      * If yes, delete that old backup, because we only want **one backup** at a time.

**Step 2: Change the current active password's flag**

* + - * Find the current active password (is\_backup = 0) for the user and site.
      * Change its is\_backup to 1 → this makes it the **backup**.

**Step 3: Insert the new password as active**

* + - * Insert a **new row** with the new password.
      * Set is\_backup = 0 (active).

### 5- create the database folder

in the (my\_pass\_project) dir:

[mkdir database]

# to store the .py files related to the database

[touch \_\_init\_\_.py]

# so python can deal with the database folder as a package

[touch db\_config.py]

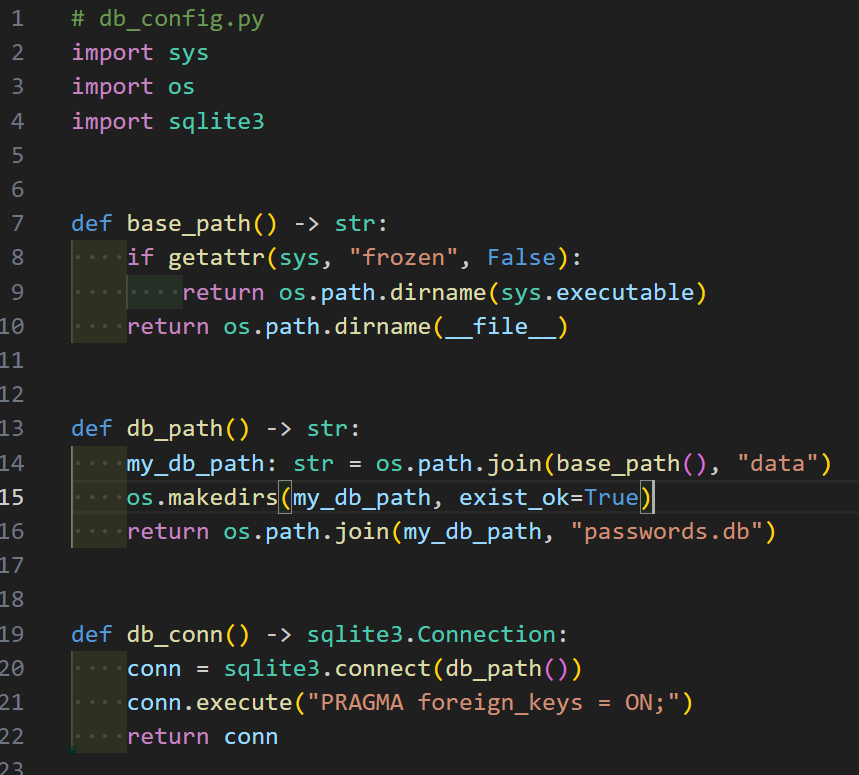
# responsible for methods that usually is going to be used more than one time like the function resposible for database connection [touch db\_schema.py]

# holds the schema code

###### What we currently have?

* a project directory (pass\_manager)
* in that directory we have my\_pass\_project/ and inside it is the documentation/
* in the (my\_pass\_project) a virtual environment (venv)
* a .gitignore file which includes (\_\_pychache\_\_, \*.pyc, venv/, .env)
* in the (my\_pass\_project) dir: we have the (data/) dir:
* inside of it there the (passwords.db) file:
* added the (data/) dir to the .gitignore file
* draw database schema
* created the database folder with files: (\_\_init\_\_.py, config.py, schema.py)

## db\_config.py



### 1- Imports

* **Explanation:**
  + - sys: Access system-specific variables and functions, such as sys.executable and sys.frozen.
    - os: File and path operations, like joining paths, creating directories, and reading file paths.
    - sqlite3: Python’s built-in SQLite database module. Used to connect to .db files and execute SQL commands.

### 2- base\_path()

* **Purpose:**
* Returns the root directory of the application, depending on how the program is being run:
* If it's running as a compiled .exe file (via PyInstaller), it returns the directory where the .exe file is located.
* If it's running as a Python script, it returns the directory where this db\_config.py file is located.

### 3- getattr(sys, "frozen", False)

* + - This checks whether the app is "frozen" — a PyInstaller term meaning "converted into an .exe file."
    - PyInstaller sets the attribute sys.frozen = True when your app is bundled.
    - This lets your script detect whether it’s being run from a .py script or a bundled .exe.

### 4- sys.executable

* + - If running as .exe, this gives the full path of the executable.
    - e.g., C:/Users/User/Desktop/dist/my\_app.exe

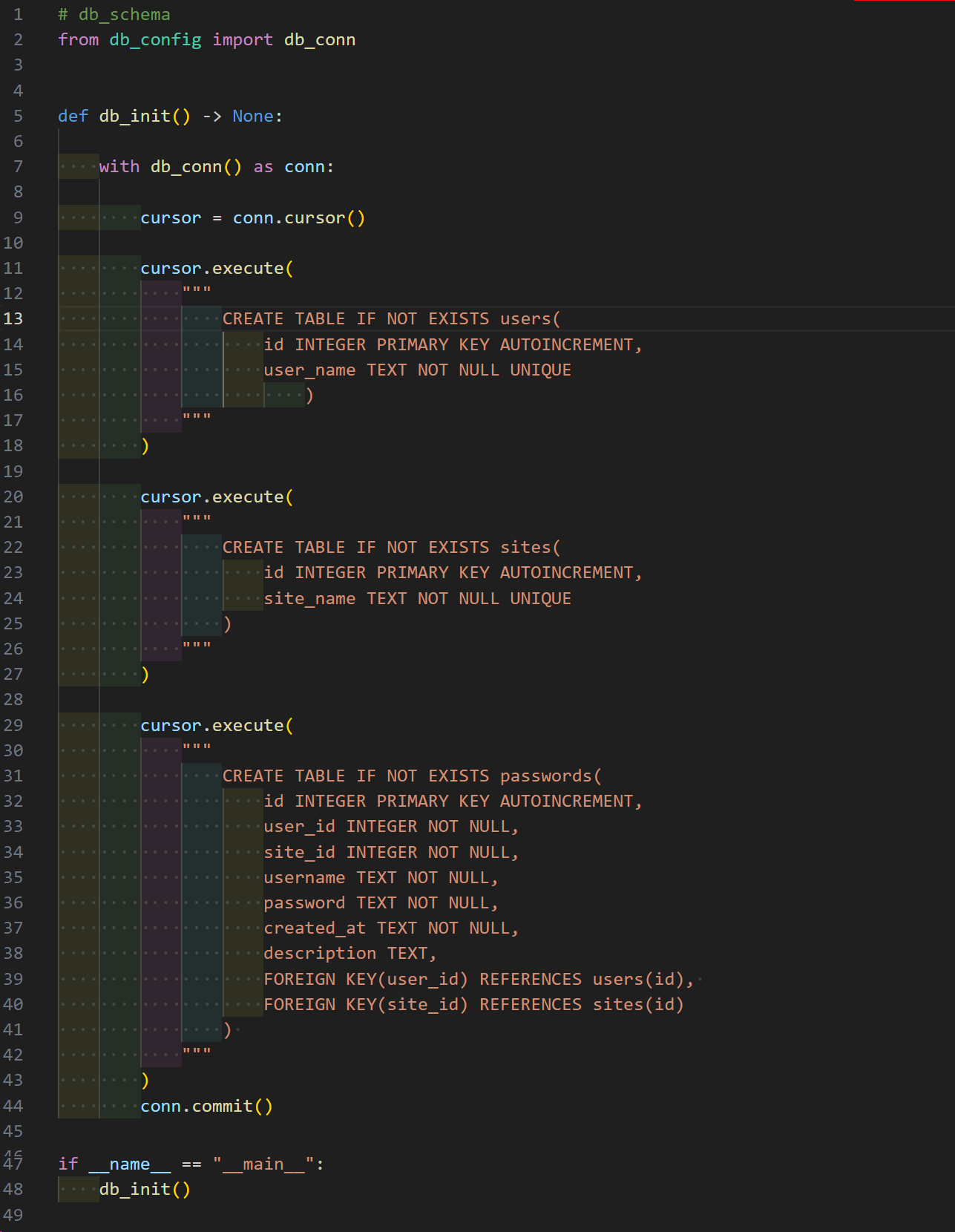
### 5- os.path.dirname(sys.executable)

* + - This strips the file name and returns the folder where the .exe is located.
    - e.g., C:/Users/User/Desktop/dist

### 6- \_\_file\_\_

* + - This is a built-in variable in Python scripts.
    - It represents the current file’s absolute or relative path.
    - os.path.dirname(\_\_file\_\_) gives the folder of db\_config.py.

## db\_schema



### 1- imports

* + - Purpose: You are importing the db\_conn() function from the db\_config.py file (which you wrote earlier).
    - db\_conn() returns an active SQLite database connection with PRAGMA foreign\_keys = ON.
    - This lets you reuse your database connection logic cleanly and avoids duplication.

### 2- def db\_init() -> None:

* + - This defines a function named db\_init() with no parameters.
    - The return type -> None is a type hint that says: this function does not return anything.
    - Purpose: to initialize the database structure (create tables).

### 3- with db\_conn() as conn:

* + - Opens a connection to your SQLite database using a with statement.
    - with ensures the connection is automatically closed even if an error occurs (this is good practice).
    - The conn object is now your live link to the database.

### 4- cursor = conn.cursor()

* + - Creates a cursor object.
    - A cursor is like a “remote control” used to send commands to the database (like creating tables, inserting data, querying, etc.).

### 5- First Table: users

* Creates a table called users if it doesn't already exist.
* Columns:
  + - id: INTEGER PRIMARY KEY AUTOINCREMENT

→ This is the unique ID of each user. It auto-increments, so each new user gets the next number (1, 2, 3...).

* + - user\_name: TEXT NOT NULL UNIQUE

→ This stores the username. It cannot be NULL, and it must be unique (no two users can have the same username).

### 6- Second Table: sites

* Creates a table called sites if it doesn't exist.
* Columns:
  + - id: Same logic as in users, to identify each site uniquely.
    - site\_name: Must be unique (no duplicate site names), and cannot be null.

### 7- Third Table: passwords

This is the **main table** where you will store passwords for different users on different sites.

| **Column** | **Type** | **Meaning** |
| --- | --- | --- |
| id | INTEGER PRIMARY KEY AUTOINCREMENT | Unique password record ID |
| user\_id | INTEGER NOT NULL | Refers to a user from the users table |
| site\_id | INTEGER NOT NULL | Refers to a site from the sites table |
| username | TEXT NOT NULL | The **login username** used on that site |
| password | TEXT NOT NULL | The actual password (will be encrypted later ideally) |
| created\_at | TEXT NOT NULL | A timestamp string for when this entry was created |
| description | TEXT | Optional notes (e.g., "used for work", "2FA enabled") |

**FOREIGN KEY(user\_id) REFERENCES users(id),**

**FOREIGN KEY(site\_id) REFERENCES sites(id)**

* These lines link the passwords table to the users and sites tables using foreign keys.
* That means user\_id must match a valid id in the users table, and same for site\_id in sites.
* This helps maintain data integrity: you can't insert a password entry for a non-existing user or site.

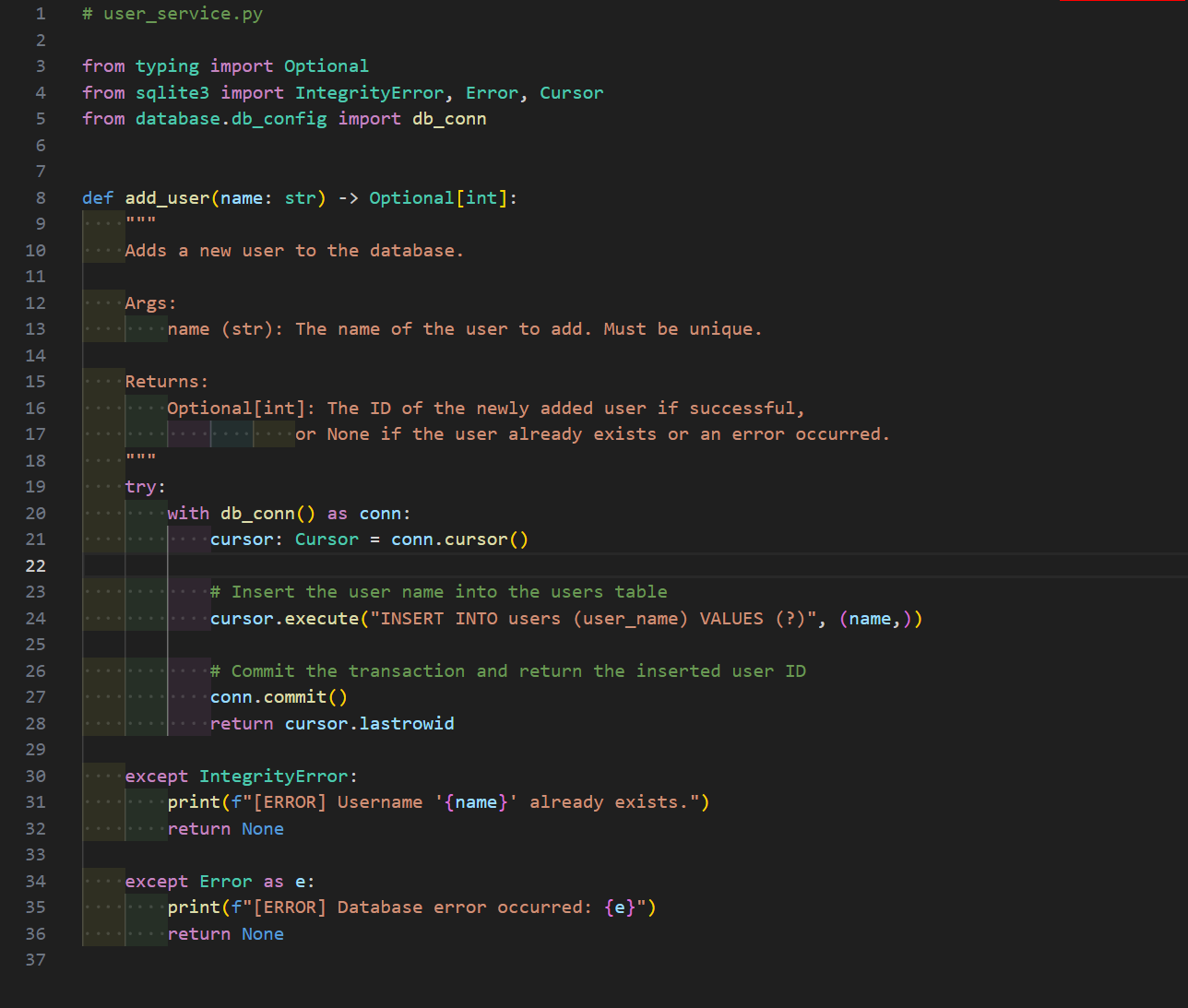
###### What we currently have?

* a project directory (pass\_manager)
* in that directory we have my\_pass\_project/ and inside it is the documentation/
* in the (my\_pass\_project) a virtual environment (venv)
* a .gitignore file which includes (\_\_pychache\_\_, \*.pyc, venv/, .env)
* in the (my\_pass\_project) dir: we have the (data/) dir:
* inside of it there the (passwords.db) file:
* added the (data/) dir to the .gitignore file
* draw database schema
* created the database folder with files: (\_\_init\_\_.py, config.py, schema.py)
* edited the (db\_config.py, db\_schema.py)

# Services

- in the (my\_pass\_project/) we mkdir (service/) that will hold our database inserting logic:

## user\_service.py



## site\_service.py

