How to Install and Run the Project (pip and uv)

Using pip (Standard Python Method)

1. Create a virtual environment:

```
python3 -m venv .venv
source .venv/bin/activate
```

2. Install dependencies:

```
pip install -r requirements.txt
```

3. Install the project in editable mode:

```
pip install -e .
```

Installing in editable mode ensures the project behaves like a proper Python package and prevents import path issues across development and CI environments.

4. Configure database environment variables.

Either set a full connection string:

```
export DATABASE_URL="postgresql://graduser:grad123@localhost:5432/gradcafe"
```

Or set individual variables:

```
export DB_HOST=localhost
export DB_PORT=5432
export DB_NAME=gradcafe
export DB_USER=graduser
export DB_PASSWORD=grad123
```

5. Run the application:

```
python src/run.py
```

Then open:

http://127.0.0.1:5000

SQL Injection Defenses

In Module 5, I changed the way SQL queries are written so they are safe.

Before, SQL statements could be built by joining text together (for example, adding user input directly into the query). That can be dangerous because someone could insert malicious SQL code.

Now:

- I do **not** combine user input directly into SQL strings.
- I use **placeholders** in the SQL query.
- The actual values are passed separately to `psycopg`.
- The database treats those values as data only — not as SQL commands.

This means even if someone tries to enter something like:

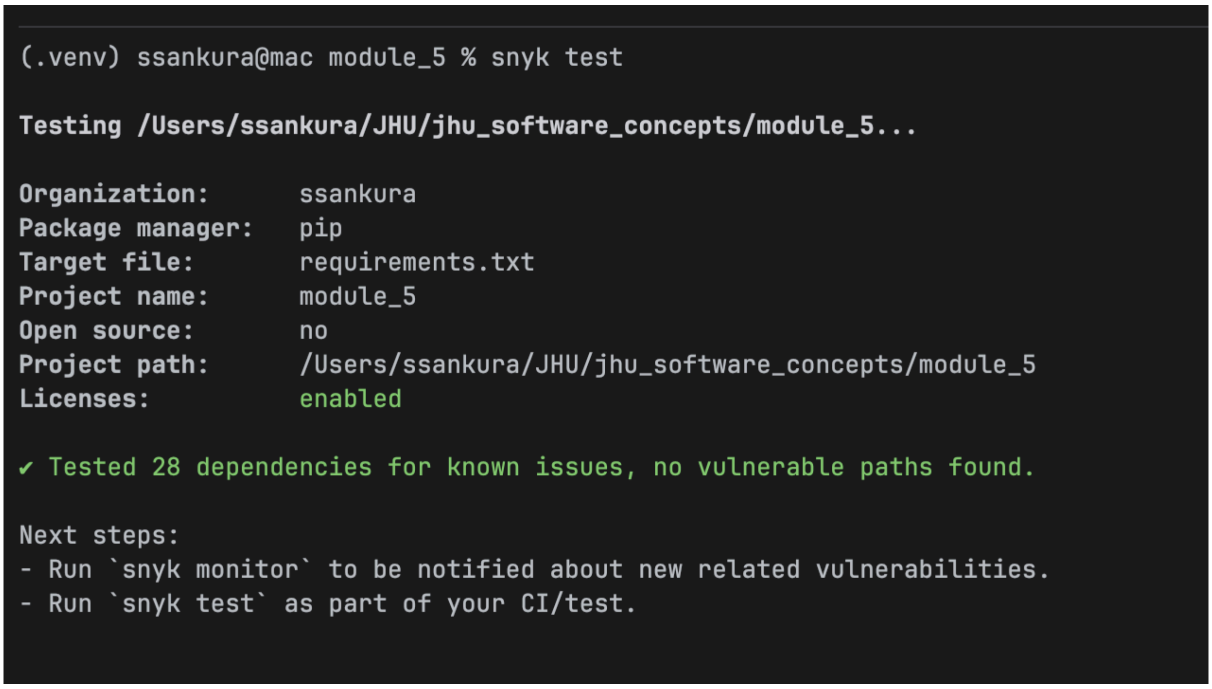
'; DROP TABLE applicants; --

it will be treated as plain text, not executed.

So SQL injection is prevented because:

- SQL structure and user data are kept separate.
- The database driver safely handles the values.

Screenshot of SYNK output



```
(.venv) ssankura@mac module_5 % snyk test

Testing /Users/ssankura/JHU/jhu_software_concepts/module_5...

Organization:      ssankura
Package manager:   pip
Target file:       requirements.txt
Project name:      module_5
Open source:       no
Project path:      /Users/ssankura/JHU/jhu_software_concepts/module_5
Licenses:          enabled

✔ Tested 28 dependencies for known issues, no vulnerable paths found.

Next steps:
- Run `snyk monitor` to be notified about new related vulnerabilities.
- Run `snyk test` as part of your CI/test.
```

Least Privilege Database Configuration

In Module 5, I followed the principle of “least privilege.”

This means the application does **not** use a powerful database user (like a superuser).
Instead, I created a separate database role with limited permissions.

Specifically:

- The application user can **connect** to the database.
- It can **read (SELECT)** from tables.
- It cannot drop tables.
- It cannot create new databases.
- It does not have superuser privileges.

Example permissions granted:

- `GRANT CONNECT ON DATABASE`
- `GRANT USAGE ON SCHEMA public`
- `GRANT SELECT ON ALL TABLES IN SCHEMA public`

This improves security because:

- Even if the application is compromised,
- The attacker cannot delete tables,
- Cannot modify database structure,
- Cannot escalate privileges.

By limiting permissions to only what the application needs, the system is safer and follows security best practices.

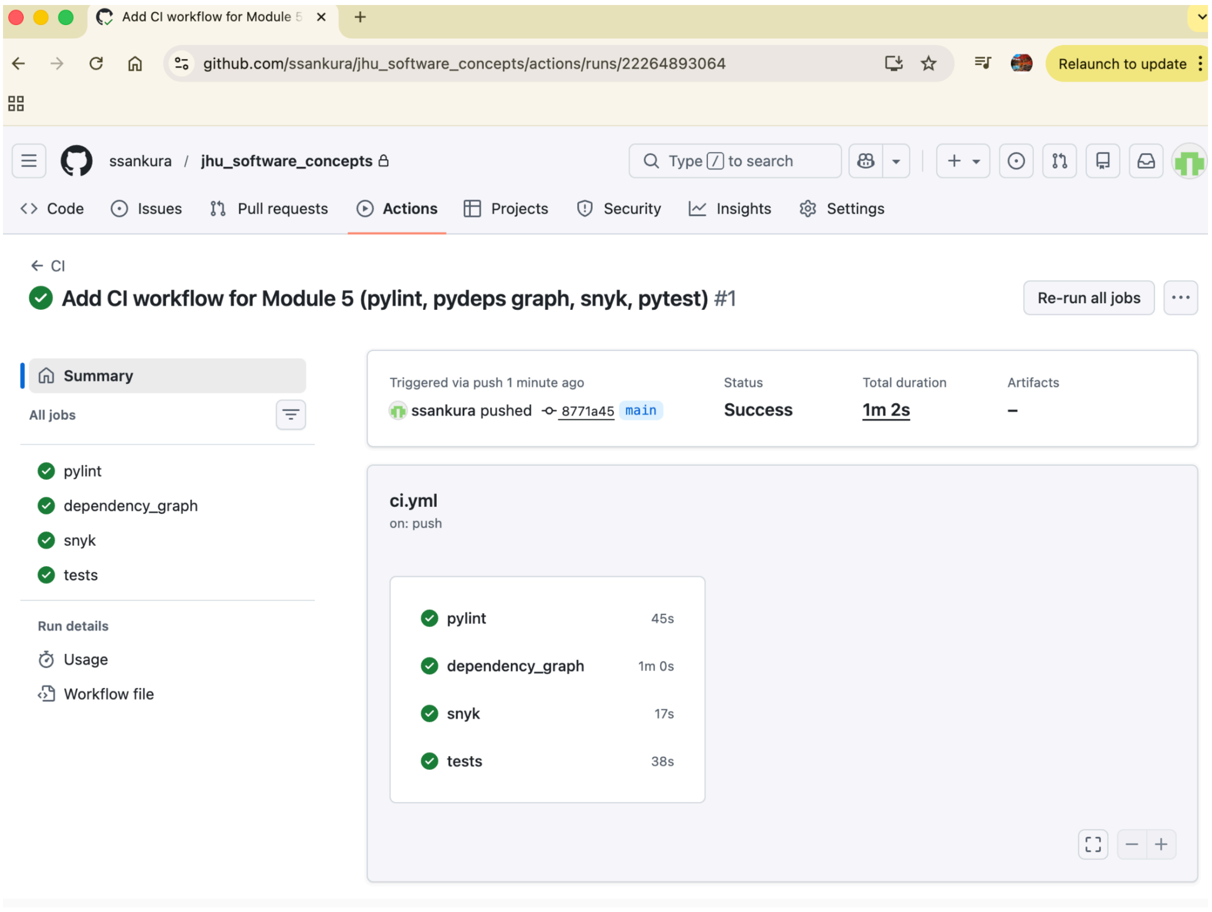
Dependency Graph Summary

The dependency graph shows how the different parts of the Flask application connect to each other. At the center is the main app package, which controls the web pages such as the analysis page and the pull-data page. These pages use the database helper module (`app.db`) to execute SQL queries and retrieve data from PostgreSQL using `psycopg`. The data-loading modules (`load_data.py` and `pull_data.py`) are separate and handle reading JSON data and inserting it into the database. The `query_data.py` module contains reusable query functions that are used by both the web application and the test suite. Standard Python libraries (such as `os`, `json`, and `datetime`) support configuration and data processing, while Flask and psycopg provide the web and database functionality. Overall, the graph demonstrates a clean separation between the web interface, database layer, and data-loading logic, making the project easier to maintain, test, and secure.

SQL LIMIT Enforcement

In Module 5, the SQL `LIMIT` value is carefully controlled before the query is executed. The application does not directly place user input into the SQL statement. Instead, the limit value is checked to make sure it is a valid number. If the value is missing or invalid, the system uses a safe default value of 10. This prevents someone from entering a very large number that could overload the database. The LIMIT value is also passed as a parameter instead of being combined into the SQL string. This keeps user input separate from the SQL structure. By validating and parameterizing the LIMIT clause, the application prevents SQL injection and protects database performance.

Module 5 Workflow Successful Run



Add CI workflow for Module 5 (pylint, pydeps graph, snyk, pytest) #1

Triggered via push 1 minute ago — ssankura pushed 8771a45 main

Status: Success

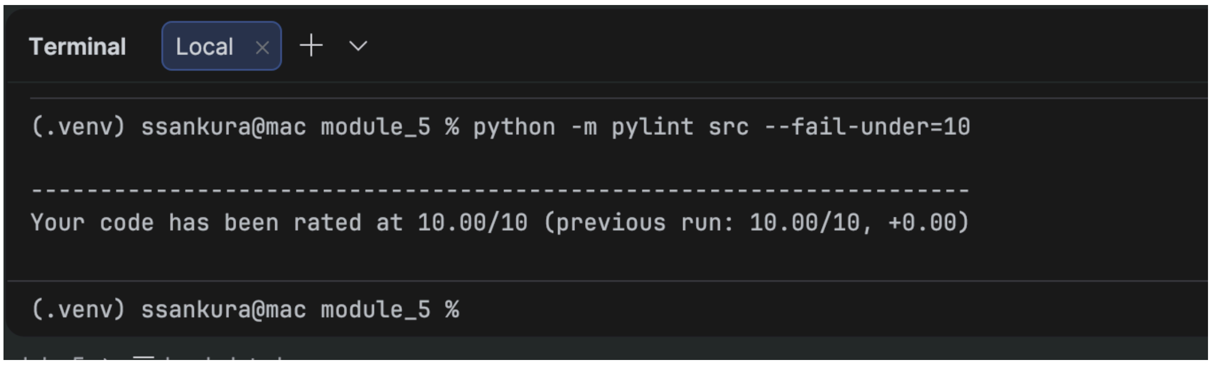
Total duration: 1m 2s

Artifacts: –

ci.yml
on: push

pylint	45s
dependency_graph	1m 0s
snyk	17s
tests	38s

Pylint RUN Score Screenshot:

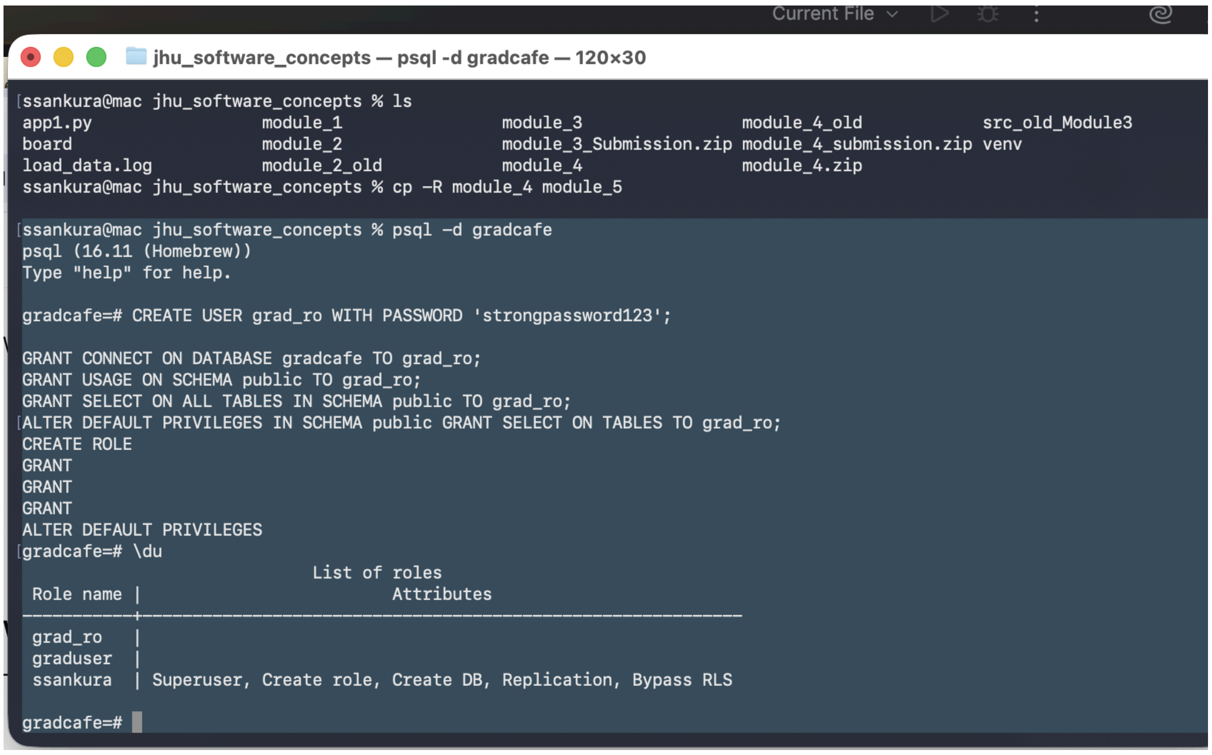


```
(.venv) ssankura@mac module_5 % python -m pylint src --fail-under=10

------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 10.00/10, +0.00)

(.venv) ssankura@mac module_5 %
```

Screenshot of Gradcafe User creation and DB permissions



```
ssankura@mac jhu_software_concepts % ls
app1.py                 module_1                module_3                module_4_old            src_old_Module3
board                   module_2                module_3_Submission.zip module_4_submission.zip venv
load_data.log           module_2_old            module_4                module_4.zip
ssankura@mac jhu_software_concepts % cp -R module_4 module_5

ssankura@mac jhu_software_concepts % psql -d gradcafe
psql (16.11 (Homebrew))
Type "help" for help.

gradcafe=# CREATE USER grad_ro WITH PASSWORD 'strongpassword123';

GRANT CONNECT ON DATABASE gradcafe TO grad_ro;
GRANT USAGE ON SCHEMA public TO grad_ro;
GRANT SELECT ON ALL TABLES IN SCHEMA public TO grad_ro;
ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT SELECT ON TABLES TO grad_ro;
CREATE ROLE
GRANT
GRANT
GRANT
ALTER DEFAULT PRIVILEGES
gradcafe=# \du
                             List of roles
 Role name |                         Attributes
-----------+------------------------------------------------------------
 grad_ro   |
 graduser  |
 ssankura  | Superuser, Create role, Create DB, Replication, Bypass RLS

gradcafe=#
```