Module 5: Software Assurance & Security Report (With AI)

1. Installation and To Reproduce Environment

The application is packaged to ensure consistency across environments. Using a **setup.py** file we can make a formal installation and reproduce a reliable module 5.

The Importance of Packaging in Software Engineering Implementing a setup.py file takes a group of individual scripts and assembles them into a formal Python package, which is a fundamental practice in software assurance and to reproduce environment exactly. Packaging allows the project to be installed in "editable mode" (pip install -e .), which ensures that internal module imports work consistently across different environments, regardless of the user's current working directory. By explicitly defining dependencies and metadata, we eliminate the common problem of "it works on my machine", allowing automated tools like uv or CI/CD pipelines to recreate the exact environment needed for secure execution.

Using uv (Recommended)

uv provides excellent software assurance through the "force synchronization", This ensures that the environment is reproduced flawlessly and matches the specification exactly and there are no missed configurations.

1. uv venv
2. source .venv/bin/activate (or .\.venv\Scripts\activate on Windows)
3. uv pip sync requirements.txt
4. pip install -e .

Using pip

1. python -m venv .venv
2. source .venv/bin/activate (or .\.venv\Scripts\activate on Windows)
3. pip install -r requirements.txt
4. pip install -e .

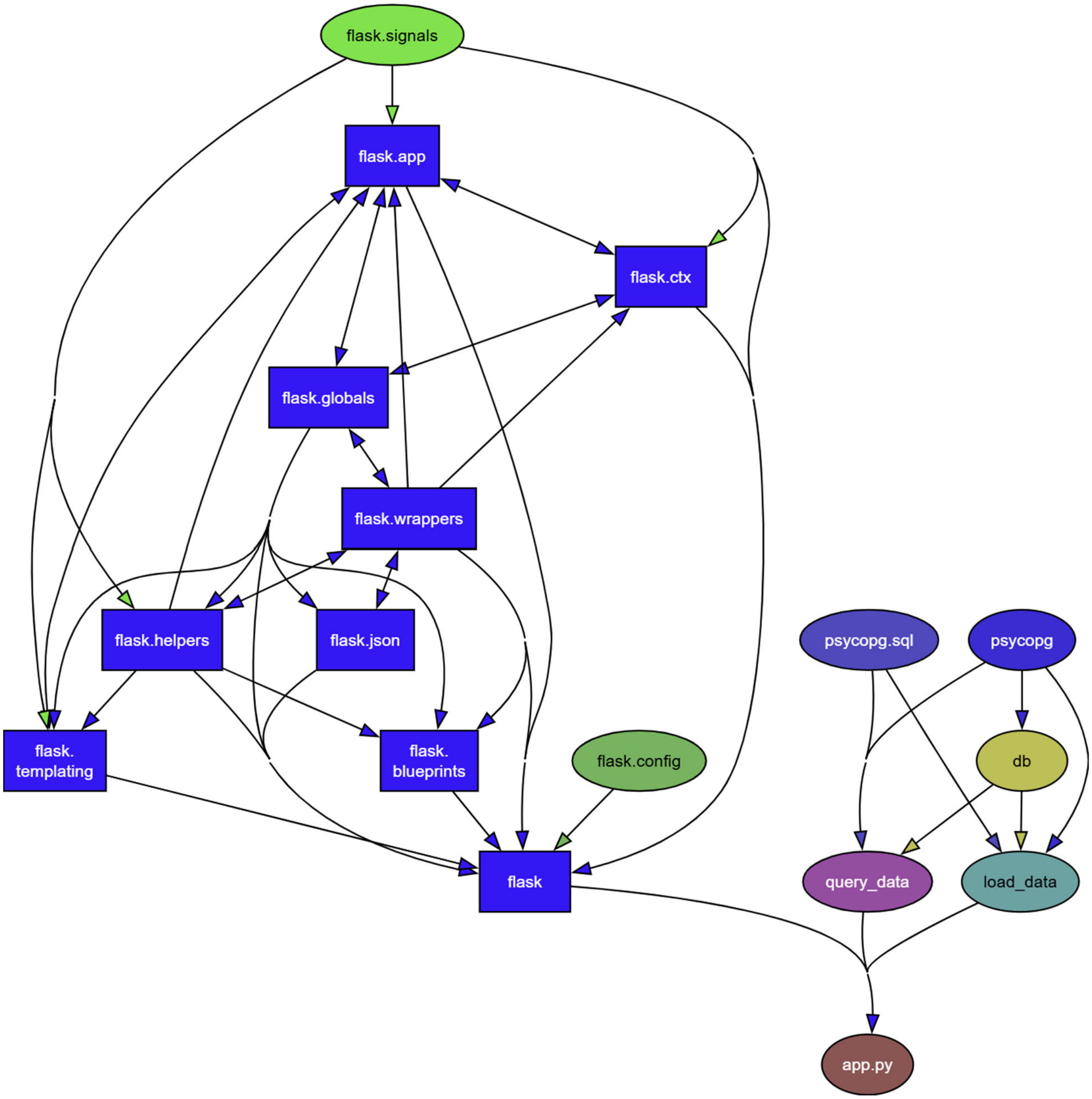
---

2. Dependency Graph Summary

The dependency graph depicts a modular Three Tier application architecture with the app.py module at its core, since it is the presentation and routing hub for my Flask app.

As an entry point into the overall application flow, app.py coordinates all high-level flows of control by calling upon query_data.py for analytical outputs and load_data.py for populating the database with data. The dependency graph illustrates that query_data.py serves as a critical logic intermediary between the app.py module and the low-level database access provided through the db.py module. Additionally, since dependencies on the external libraries required for db.py's operations, such as psycopg and python-dotenv, are isolated from one another; the database connection logic and the sensitive secret management necessary to connect securely to the database are isolated from the remainder of the application. The visualization also illustrates a strict one way flow of dependencies, thus preventing circular dependencies from forming and ensuring that the scraper (scrape.py) and data loaders remain decoupled from the web front-end application. This structure shows a clear separation of concerns and will help to reduce code complexity and improve the overall security posture of an application by isolating sensitive database operations from the rest of the app. Lastly, generate_answers_pdf.py demonstrates another layer of utility dependence on the reportlab library for creating printable documents from the processed data.

SCREENSHOT1: Dependency Graph (dependency.svg)

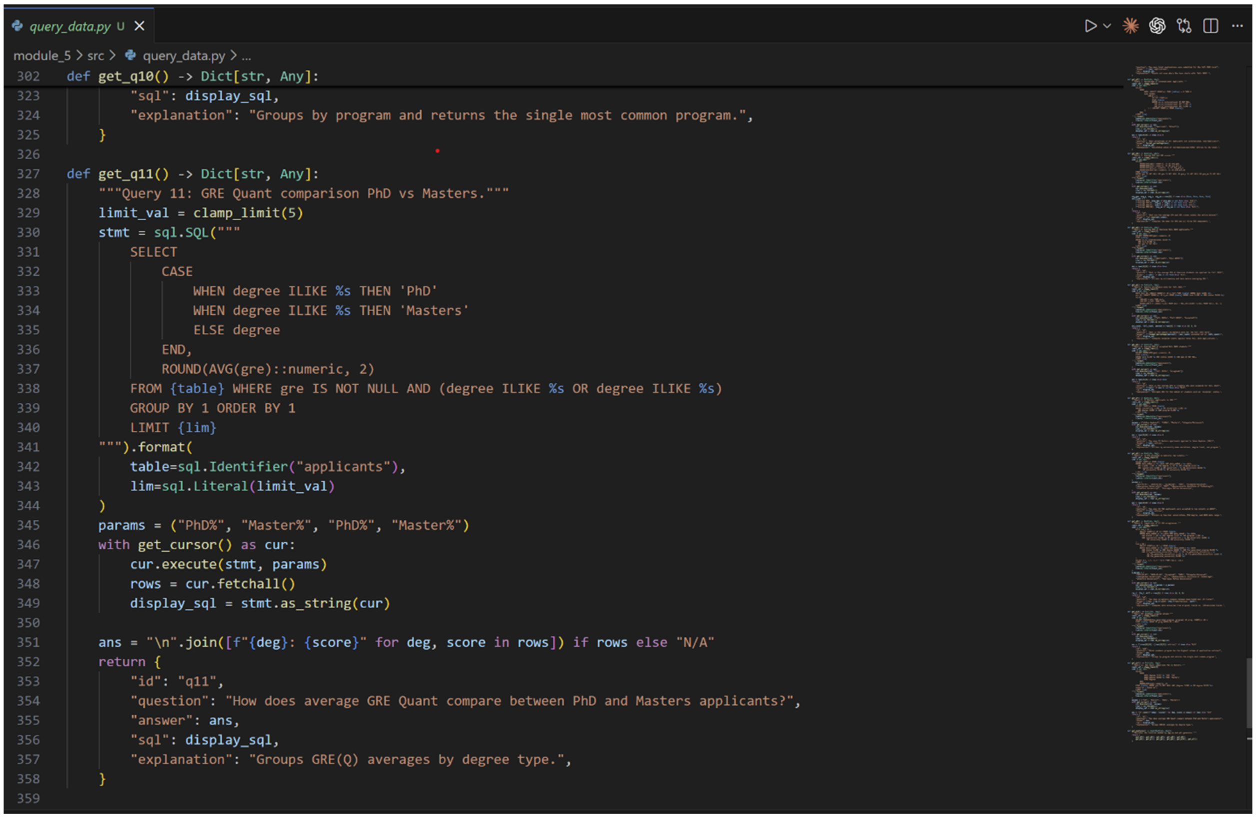

flask.signals
flask.app
flask.ctx
flask.globals
flask.wrappers
flask.helpers
flask.json
flask.
templating
flask.
blueprints
flask.config
flask
psycopg.sql
psycopg
db
query_data
load_data
app.py

3. SQL Injection Defenses

To protect the database from injection attacks, the application logic was rewritten to move away from string formatting and toward safe query composition.

- **Safe Composition:** I use psycopg.sql to build queries. Table names and identifiers are wrapped in sql.Identifier, and static values are wrapped in sql.Literal.
- **Separation of Concerns:** SQL statements are defined as objects separately from their execution.
- **Parameterization:** All user provided data or variables are passed as secondary arguments to the execute() method, using %s placeholders. The database driver handles the escaping, making it impossible for a malicious string to be interpreted as a command.

SCREENSHOT2: Code snippet showing psycopg.sql implementation



```python
def get_q10() -> Dict[str, Any]:
        "sql": display_sql,
        "explanation": "Groups by program and returns the single most common program.",
    }

def get_q11() -> Dict[str, Any]:
    """Query 11: GRE Quant comparison PhD vs Masters."""
    limit_val = clamp_limit(5)
    stmt = sql.SQL("""
        SELECT
            CASE
                WHEN degree ILIKE %s THEN 'PhD'
                WHEN degree ILIKE %s THEN 'Masters'
                ELSE degree
            END,
            ROUND(AVG(gre)::numeric, 2)
        FROM {table} WHERE gre IS NOT NULL AND (degree ILIKE %s OR degree ILIKE %s)
        GROUP BY 1 ORDER BY 1
        LIMIT {lim}
    """).format(
        table=sql.Identifier("applicants"),
        lim=sql.Literal(limit_val)
    )
    params = ("PhD%", "Master%", "PhD%", "Master%")
    with get_cursor() as cur:
        cur.execute(stmt, params)
        rows = cur.fetchall()
        display_sql = stmt.as_string(cur)

    ans = "\n".join([f"{deg}: {score}" for deg, score in rows]) if rows else "N/A"
    return {
        "id": "q11",
        "question": "How does average GRE Quant compare between PhD and Masters applicants?",
        "answer": ans,
        "sql": display_sql,
        "explanation": "Groups GRE(Q) averages by degree type.",
    }
```

4. Database Hardening & Least Privilege

The database has been secured by applying the "Principle of Least Privilege" to the module3_user account. This ensures a "Default Deny" environment where the application can only perform the actions necessary for its function.

The Hardening Process (Step-by-Step)

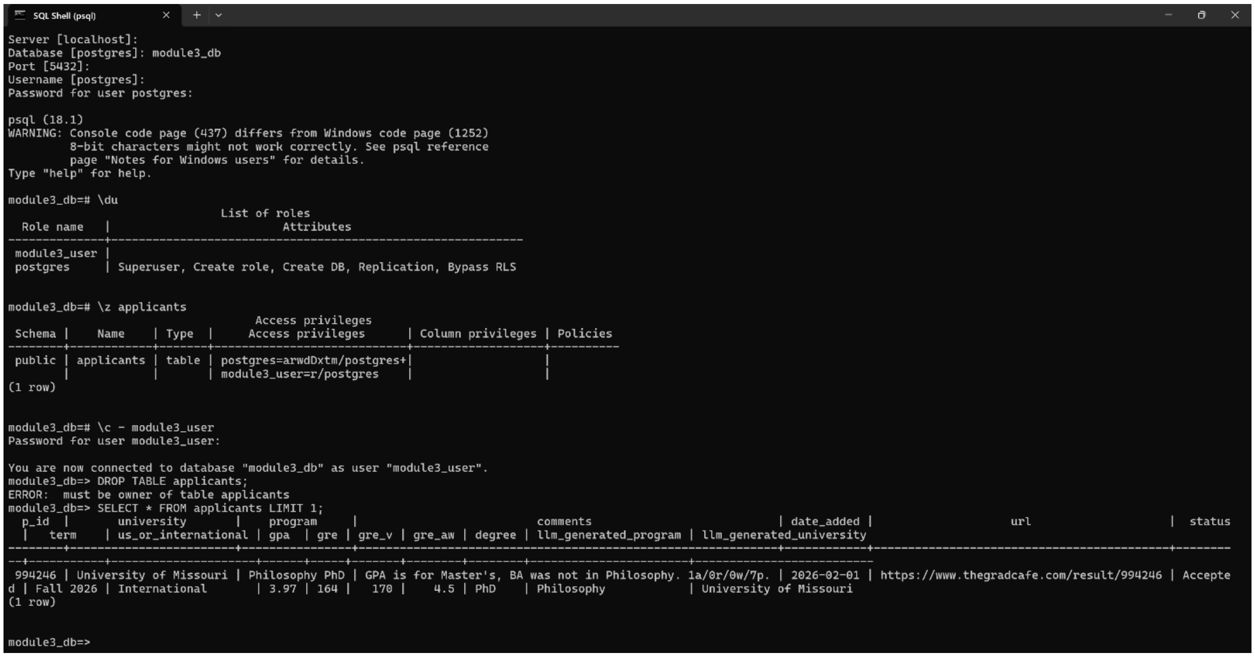
1. **Stripping Administrative Attributes**: Used ALTER ROLE to set account powers to NOCREATEDB and NOCREATEROLE, ensuring compromised credentials cannot be used to bypass security or delete the database.
2. **Implementing "Zero Trust" and Fixing Ownership**: Realized the user could still DROP the table as the default owner; transferred ownership to the postgres admin to close this loophole and revoked all permissions from the PUBLIC role.
3. **Final Lockdown**: Restricted the user to a "Default Deny" environment where they have only CONNECT privileges on the database and SELECT privileges on specific tables.

Hardening Commands Executed:

```
-- Prevent instance leaks

REVOKE ALL ON SCHEMA public FROM PUBLIC;

REVOKE ALL ON DATABASE module3_db FROM PUBLIC;

-- Lock down the schema

ALTER SCHEMA public OWNER TO postgres;

GRANT USAGE ON SCHEMA public TO module3_user;

-- Secure the actual table data

ALTER TABLE applicants OWNER TO postgres;

REVOKE ALL ON TABLE applicants FROM module3_user;

GRANT SELECT ON TABLE applicants TO module3_user;
```

- **Restricted Attributes:** The user account was stripped of administrative powers (NOCREATEDB, NOCREATEROLE).
- **Default Deny:** I revoked all permissions from the PUBLIC role on the database and schema to prevent "ghost permissions" from leaking through.
- **Ownership Transfer:** I transferred ownership of the applicants table to the postgres superuser. This is a critical security boundary: in PostgreSQL, only the owner or a superuser can DROP or TRUNCATE a table.
- **Selective Access:** The module3_user was granted only CONNECT to the database and SELECT on the table.

SCREENSHOT3: Terminal output showing "ERROR: must be owner of table applicants" when trying to drop a table as module3_user



```
Server [localhost]:
Database [postgres]: module3_db
Port [5432]:
Username [postgres]:
Password for user postgres:

psql (18.1)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

module3_db=# \du
                             List of roles
  Role name   |                         Attributes
--------------+------------------------------------------------------------
 module3_user |
 postgres     | Superuser, Create role, Create DB, Replication, Bypass RLS


module3_db=# \z applicants
                                  Access privileges
 Schema |    Name    | Type  |     Access privileges      | Column privileges | Policies
--------+------------+-------+----------------------------+-------------------+----------
 public | applicants | table | postgres=arwdDxtm/postgres+|                   |
        |            |       | module3_user=r/postgres    |                   |
(1 row)


module3_db=# \c - module3_user
Password for user module3_user:

You are now connected to database "module3_db" as user "module3_user".
module3_db=> DROP TABLE applicants;
ERROR:  must be owner of table applicants
module3_db=> SELECT * FROM applicants LIMIT 1;
  p_id  |       university       |    program     |                            comments                            | date_added |                       url                       |  status
  |   term    | us_or_international | gpa  | gre | gre_v | gre_aw | degree | llm_generated_program | llm_generated_university
--------+------------------------+----------------+----------------------------------------------------------------+------------+-------------------------------------------------+--------
--+-----------+---------------------+------+-----+-------+--------+--------+-----------------------+--------------------------
 994246 | University of Missouri | Philosophy PhD | GPA is for Master's, BA was not in Philosophy. 1a/0r/0w/7p. | 2026-02-01 | https://www.thegradcafe.com/result/994246 | Accepte
d | Fall 2026 | International       | 3.97 | 164 |   170 |    4.5 | PhD    | Philosophy            | University of Missouri
(1 row)


module3_db=>
```

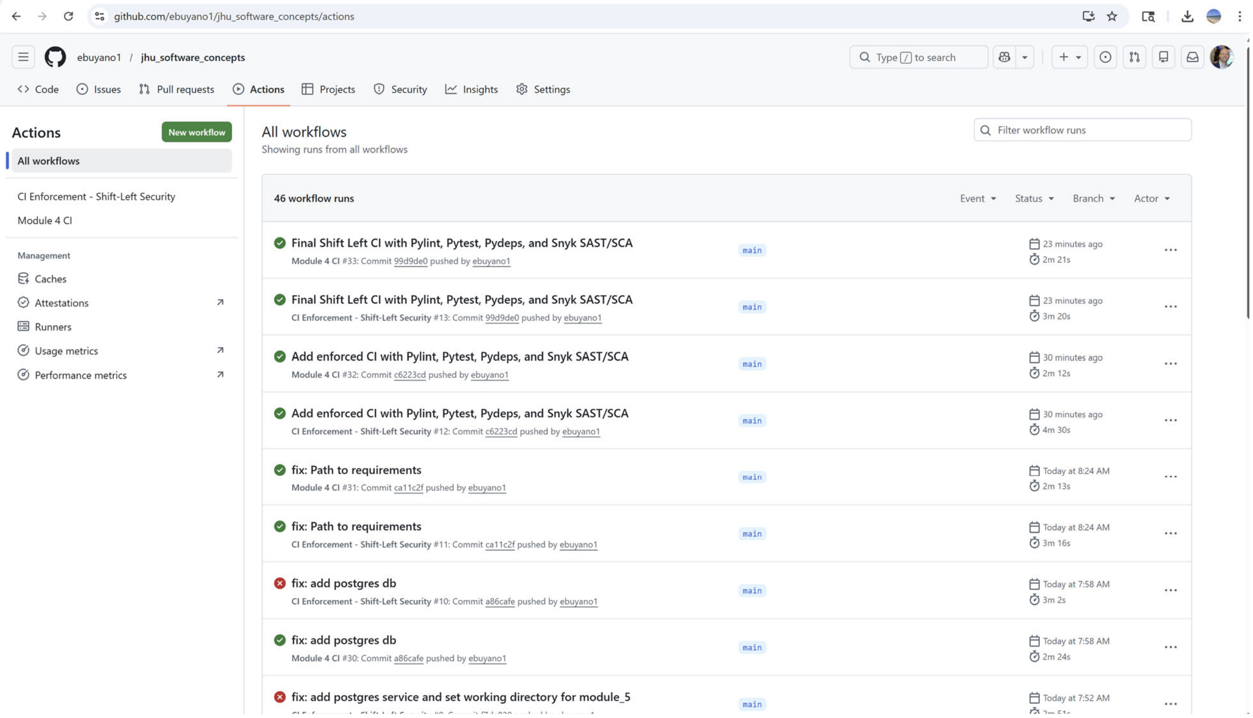
5. Requirements Met for SQL

Requirement	Implementation Detail
LIMIT Enforced	Every query in query_data.py uses a clamp_limit() function to ensure results never exceed 100 rows.
Separated Execution	Queries are defined as sql.SQL objects before being passed to cur.execute().
Safe Parameterization	Placeholder %s syntax is used for all variable data.

6. CI Enforcement with GitHub Actions

I implemented a "Shift-Left" security strategy using GitHub Actions. The build fails if quality or security gates are not met.

SCREENSHOT4: GitHub Actions passing (Green Checkmarks)



All workflows

Showing runs from all workflows

46 workflow runs

✓ Final Shift Left CI with Pylint, Pytest, Pydeps, and Snyk SAST/SCA — main — 23 minutes ago — 2m 21s
Module 4 CI #33: Commit 99d9de0 pushed by ebuyano1

✓ Final Shift Left CI with Pylint, Pytest, Pydeps, and Snyk SAST/SCA — main — 23 minutes ago — 3m 20s
CI Enforcement - Shift-Left Security #13: Commit 99d9de0 pushed by ebuyano1

✓ Add enforced CI with Pylint, Pytest, Pydeps, and Snyk SAST/SCA — main — 30 minutes ago — 2m 12s
Module 4 CI #32: Commit c6223cd pushed by ebuyano1

✓ Add enforced CI with Pylint, Pytest, Pydeps, and Snyk SAST/SCA — main — 30 minutes ago — 4m 30s
CI Enforcement - Shift-Left Security #12: Commit c6223cd pushed by ebuyano1

✓ fix: Path to requirements — main — Today at 8:24 AM — 2m 13s
Module 4 CI #31: Commit ca11c2f pushed by ebuyano1

✓ fix: Path to requirements — main — Today at 8:24 AM — 3m 16s
CI Enforcement - Shift-Left Security #11: Commit ca11c2f pushed by ebuyano1

✗ fix: add postgres db — main — Today at 7:58 AM — 3m 2s
CI Enforcement - Shift-Left Security #10: Commit a86cafe pushed by ebuyano1

✓ fix: add postgres db — main — Today at 7:58 AM — 2m 24s
Module 4 CI #30: Commit a86cafe pushed by ebuyano1

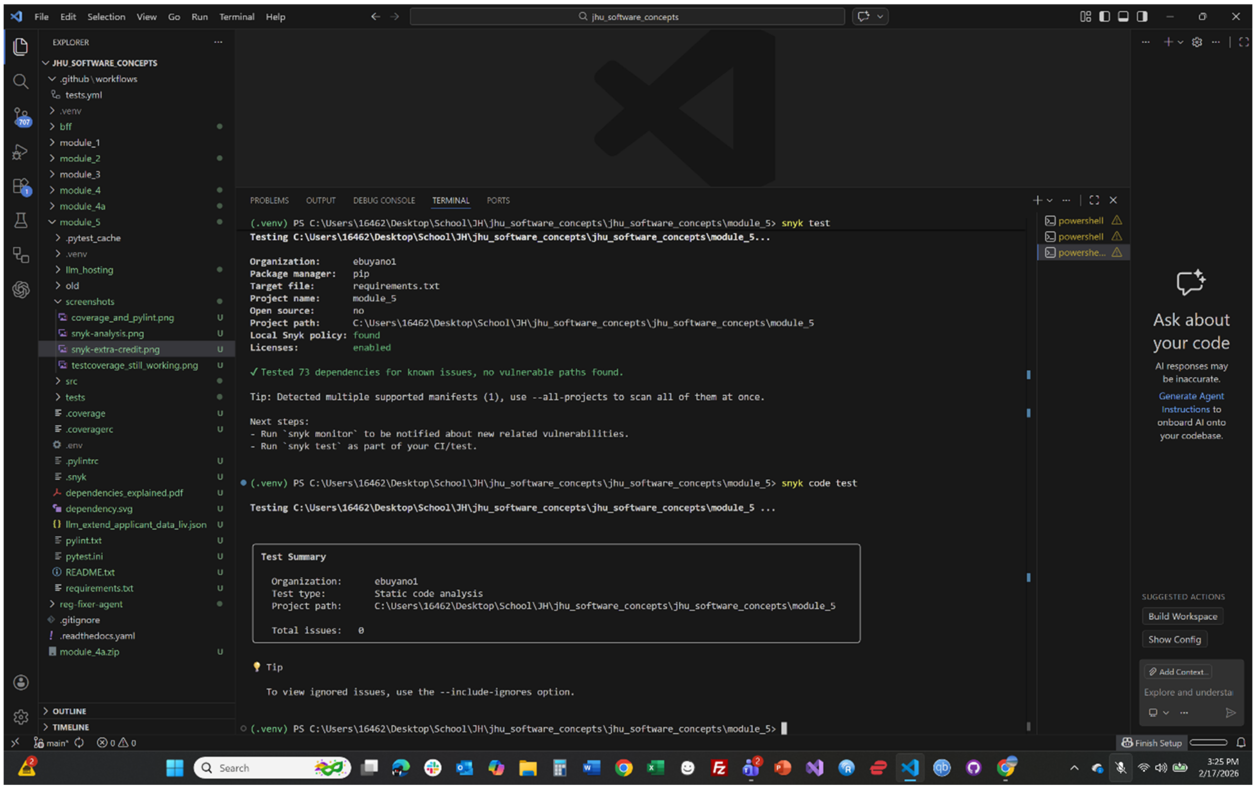
✗ fix: add postgres service and set working directory for module_5 — main — Today at 7:52 AM

7. Extra Credit: Snyk Security Evidence

A Snyk scan was performed on the project dependencies. While some high-severity issues were found in sub-dependencies (like pillow), they were remediated by pinning pillow==12.1.1 in the requirements.txt. For issues with no direct patch (e.g., diskcache), a .snyk ignore policy was implemented with a documented rationale, ensuring the CI build remains secure and passing.

For the diskcache vulnerability (where no patch is available), I implemented a Snyk ignore policy with a 30-day expiry, ensuring the vulnerability is tracked but does not block the CI pipeline while awaiting a maintainer update.

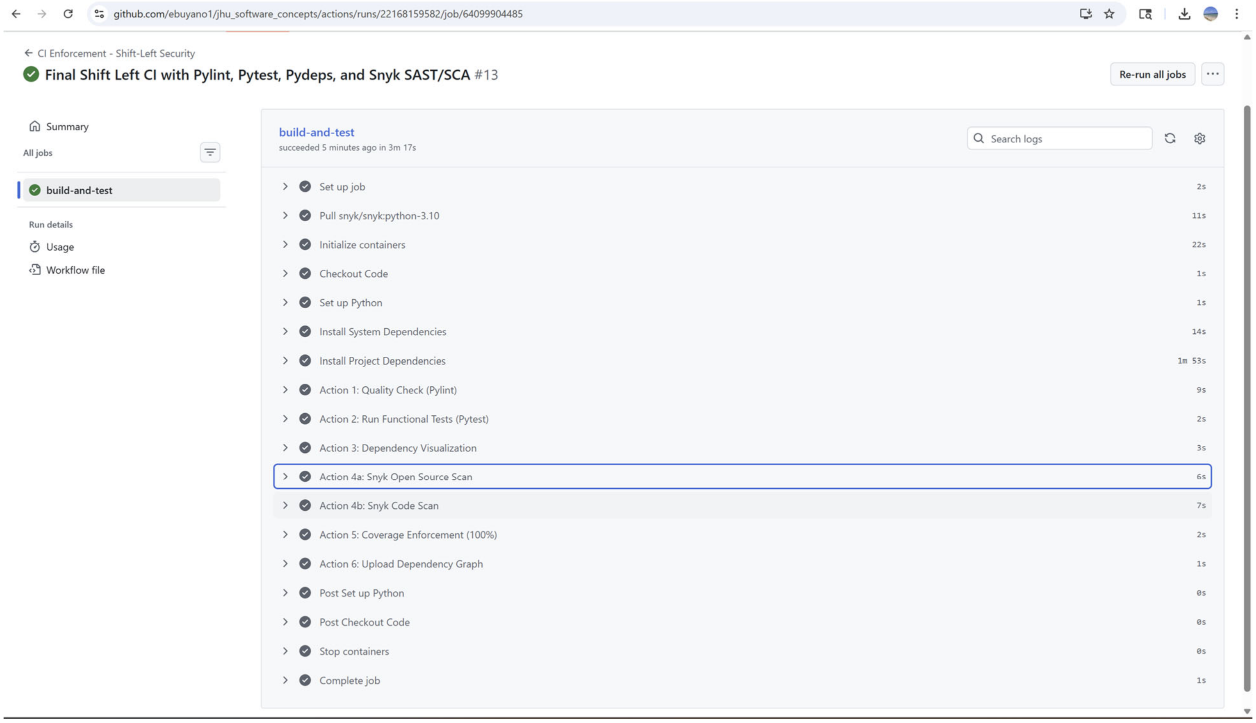
SCREENSHOT5: Snyk test output showing "No vulnerable paths found"



```
(.venv) PS C:\Users\16462\Desktop\School\JH\jhu_software_concepts\jhu_software_concepts\module_5> snyk test
Testing C:\Users\16462\Desktop\School\JH\jhu_software_concepts\jhu_software_concepts\module_5...

Organization:      ebuyano1
Package manager:   pip
Target file:       requirements.txt
Project name:      module_5
Open source:       no
Project path:      C:\Users\16462\Desktop\School\JH\jhu_software_concepts\jhu_software_concepts\module_5
Local Snyk policy: found
Licenses:          enabled

✓ Tested 73 dependencies for known issues, no vulnerable paths found.

Tip: Detected multiple supported manifests (1), use --all-projects to scan all of them at once.

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


(.venv) PS C:\Users\16462\Desktop\School\JH\jhu_software_concepts\jhu_software_concepts\module_5> snyk code test

Testing C:\Users\16462\Desktop\School\JH\jhu_software_concepts\jhu_software_concepts\module_5 ...


 Test Summary

   Organization:      ebuyano1
   Test type:         Static code analysis
   Project path:      C:\Users\16462\Desktop\School\JH\jhu_software_concepts\jhu_software_concepts\module_5

   Total issues:   0


💡 Tip

   To view ignored issues, use the --include-ignores option.


(.venv) PS C:\Users\16462\Desktop\School\JH\jhu_software_concepts\jhu_software_concepts\module_5>
```

8. Shift Left Security CI (ALL PASS)

I enforced **100% Code Coverage** using pytest-cov. Since our CI pipeline will literally fail if coverage drops below 100%, that's a massive software assurance achievement. The CI passes Pylint, Pytest, Snyk Test, Snyk Code Test, Generates dependencies svg . Github Actions shows all passed, Pylint, Pytest, Snyk Test, Snyk Code Test (extra credit), Dependency graph generated and can be downloaded for verification.


github.com/ebuyano1/jhu_software_concepts/actions/runs/22168159582/job/64099904485
CI Enforcement - Shift-Left Security
Re-run all jobs
Summary
All jobs
Run details
Usage
Workflow file
Search logs

Final Shift Left CI with Pylint, Pytest, Pydeps, and Snyk SAST/SCA #13

build-and-test

build-and-test
succeeded 5 minutes ago in 3m 17s

Set up job	2s
Pull snyk/snyk:python-3.10	11s
Initialize containers	22s
Checkout Code	1s
Set up Python	1s
Install System Dependencies	14s
Install Project Dependencies	1m 53s
Action 1: Quality Check (Pylint)	9s
Action 2: Run Functional Tests (Pytest)	2s
Action 3: Dependency Visualization	3s
Action 4a: Snyk Open Source Scan	6s
Action 4b: Snyk Code Scan	7s
Action 5: Coverage Enforcement (100%)	2s
Action 6: Upload Dependency Graph	1s
Post Set up Python	0s
Post Checkout Code	0s
Stop containers	0s
Complete job	1s