

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.

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

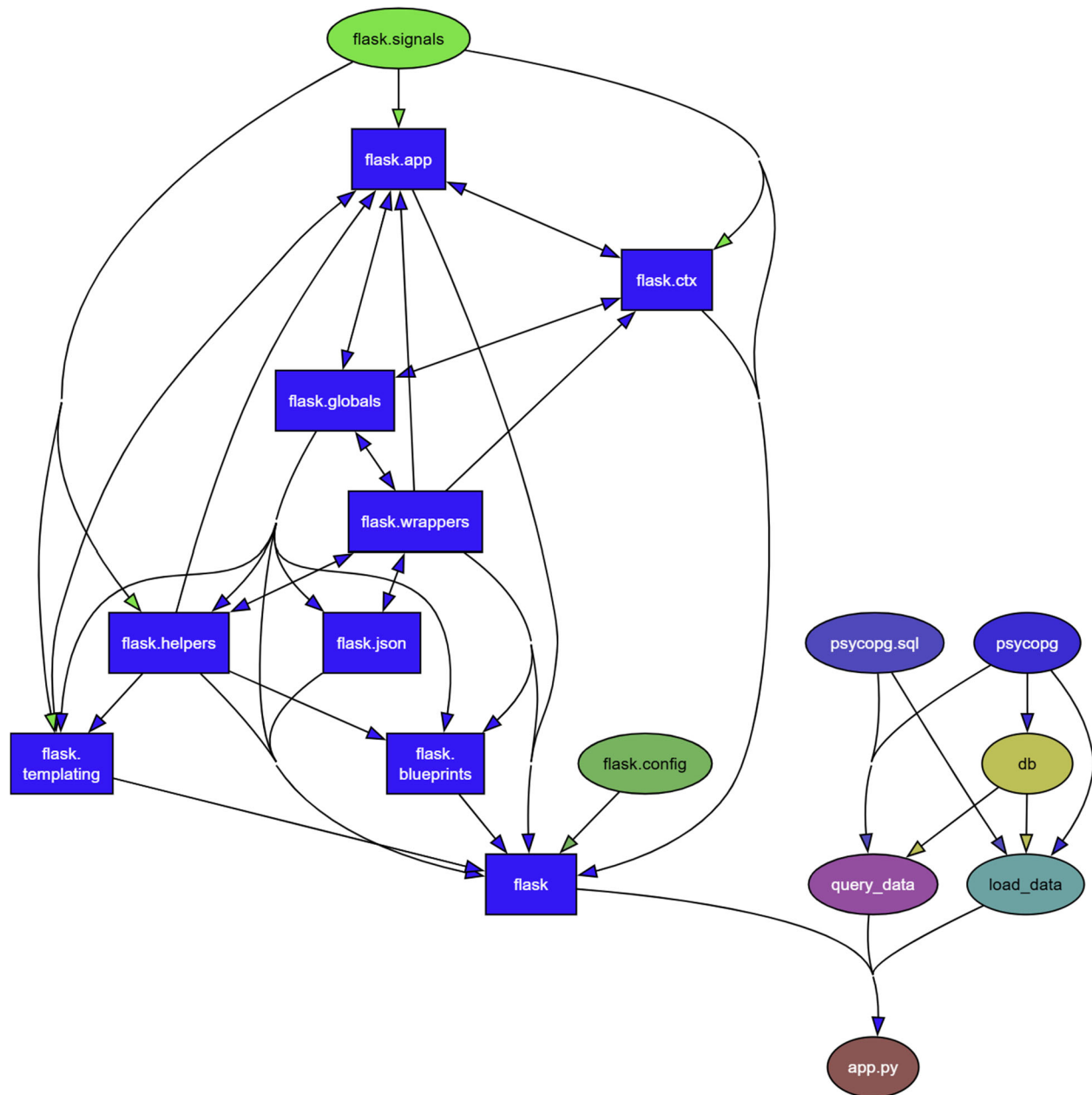
1. `python -m venv .venv`
2. `source .venv/bin/activate` (or `.\.venv\Scripts\activate` on Windows)
3. `pip install -r requirements.txt`
4. `pip install -e .`

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)

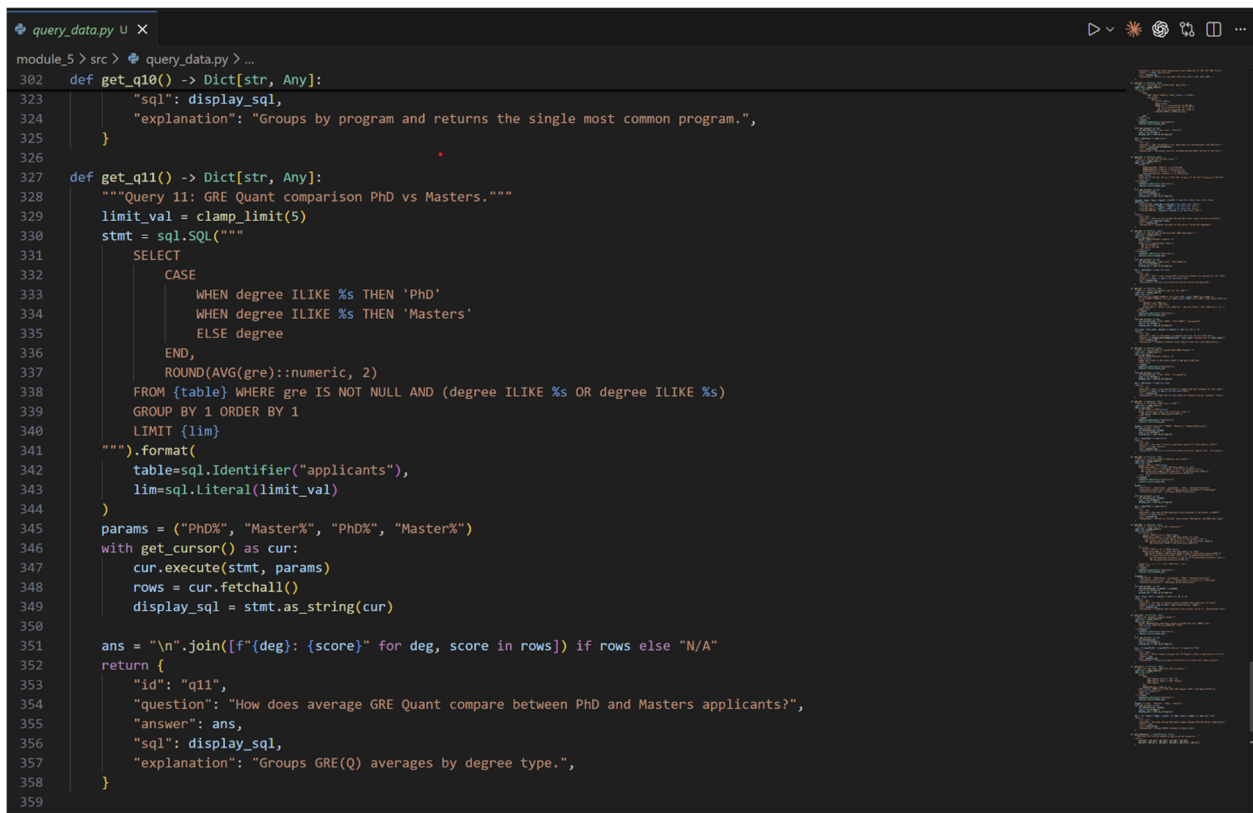


3. SQL Injection Defenses

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



```

302 def get_q10() -> Dict[str, Any]:
323     "sql": display_sql,
324     "explanation": "Groups by program and returns the single most common program.",
325 }
326
327 def get_q11() -> Dict[str, Any]:
328     """Query 11: GRE Quant comparison PhD vs Masters."""
329     limit_val = clamp_limit(5)
330     stmt = sql.SQL("""
331         SELECT
332             CASE
333                 WHEN degree ILIKE %s THEN 'PhD'
334                 WHEN degree ILIKE %s THEN 'Masters'
335                 ELSE degree
336             END,
337             ROUND(AVG(gre)::numeric, 2)
338         FROM {table} WHERE gre IS NOT NULL AND (degree ILIKE %s OR degree ILIKE %s)
339         GROUP BY 1 ORDER BY 1
340         LIMIT {lim}
341     """).format(
342         table=sql.Identifier("applicants"),
343         lim=sql.Literal(limit_val)
344     )
345     params = ("%PhD%", "Master%", "PhD%", "Master%")
346     with get_cursor() as cur:
347         cur.execute(stmt, params)
348         rows = cur.fetchall()
349         display_sql = stmt.as_string(cur)
350
351     ans = "\n".join([f"{deg}: {score}" for deg, score in rows]) if rows else "N/A"
352     return {
353         "id": "q11",
354         "question": "How does average GRE Quant compare between PhD and Masters applicants?",
355         "answer": ans,
356         "sql": display_sql,
357         "explanation": "Groups GRE(Q) averages by degree type.",
358     }
359
  
```

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.

- **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

```
SQL Shell (psql)
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
      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 | gpa | gre_v | gre_aw | degree | comments | date_added | url | status
-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----
994246 | University of Missouri | Philosophy PhD | 3.97 | 164 | 178 | 4.5 | PhD | Philosophy | 1a/8r/0w/7p. | 2026-02-01 | https://www.thegradcafe.com/result/994246 | Accepted
(1 row)

module3_db=>
```

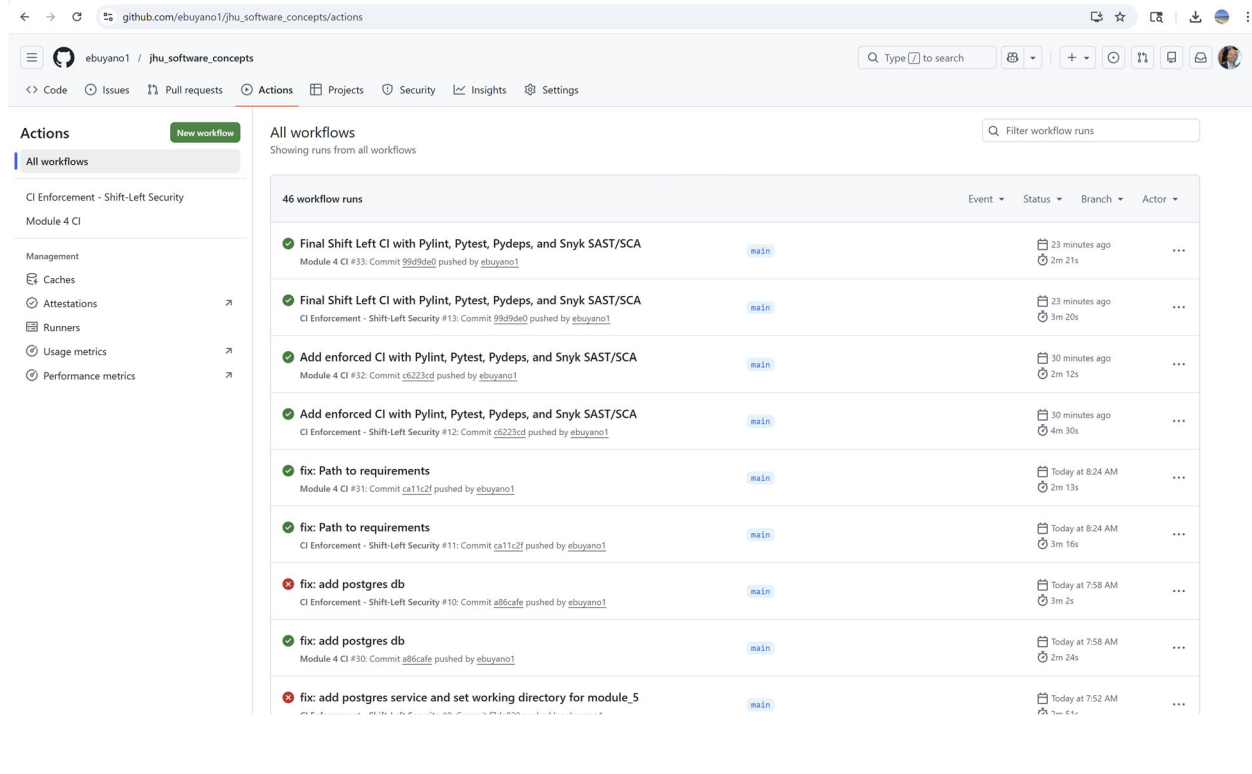
5. Requirements Met for SQL

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)



The screenshot displays the GitHub Actions interface for the repository 'ebuyano1 / jhu_software_concepts'. The 'Actions' tab is selected, showing a list of workflow runs. The left sidebar contains navigation links for 'All workflows', 'CI Enforcement - Shift-Left Security', 'Module 4 CI', 'Management', 'Caches', 'Attestations', 'Runners', 'Usage metrics', and 'Performance metrics'. The main area shows 'All workflows' with a search bar and a table of 46 workflow runs. The table columns are 'Event', 'Status', 'Branch', and 'Actor'. The runs are listed with their names, commit hashes, and timestamps. Most runs are successful, indicated by green checkmarks.

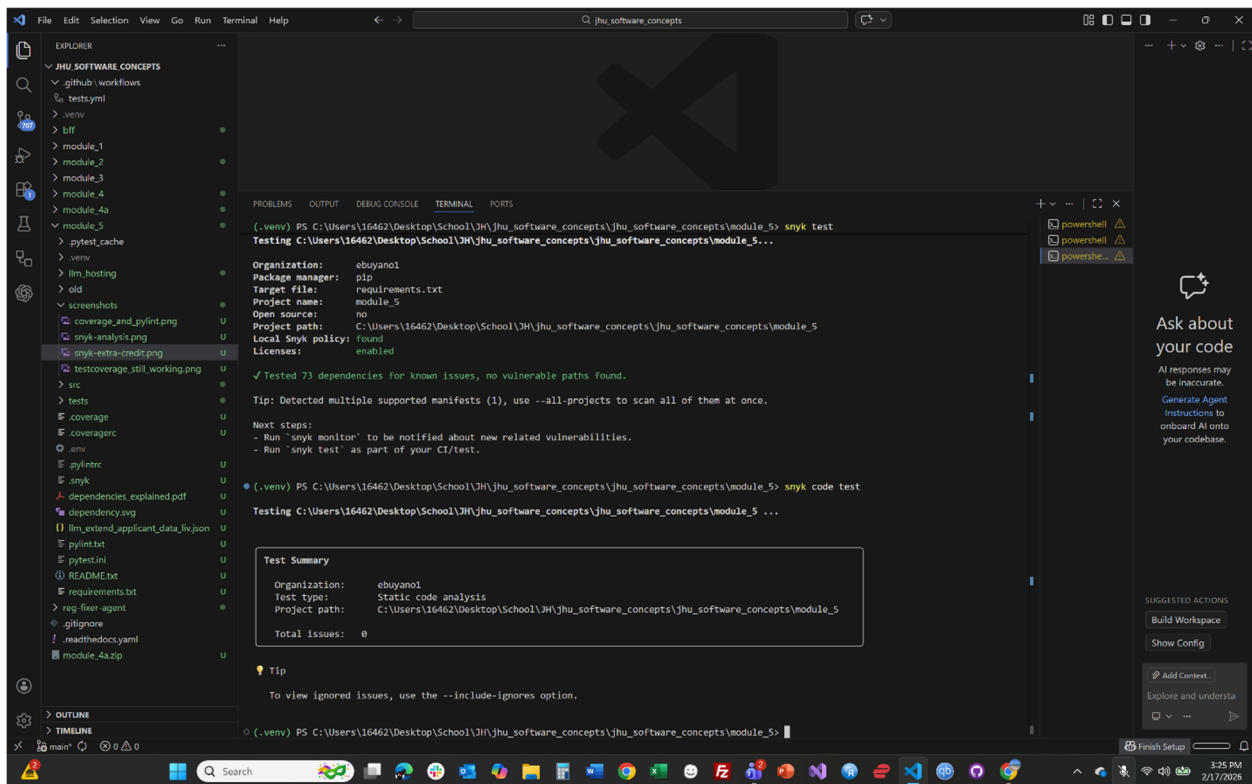
Event	Status	Branch	Actor
Final Shift Left CI with Pylint, Pytest, Pydeps, and Snyk SAST/SCA	Success	main	ebuyano1
Final Shift Left CI with Pylint, Pytest, Pydeps, and Snyk SAST/SCA	Success	main	ebuyano1
Add enforced CI with Pylint, Pytest, Pydeps, and Snyk SAST/SCA	Success	main	ebuyano1
Add enforced CI with Pylint, Pytest, Pydeps, and Snyk SAST/SCA	Success	main	ebuyano1
fix: Path to requirements	Success	main	ebuyano1
fix: Path to requirements	Success	main	ebuyano1
fix: add postgres db	Failure	main	ebuyano1
fix: add postgres db	Success	main	ebuyano1
fix: add postgres service and set working directory for module_5	Failure	main	ebuyano1

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.

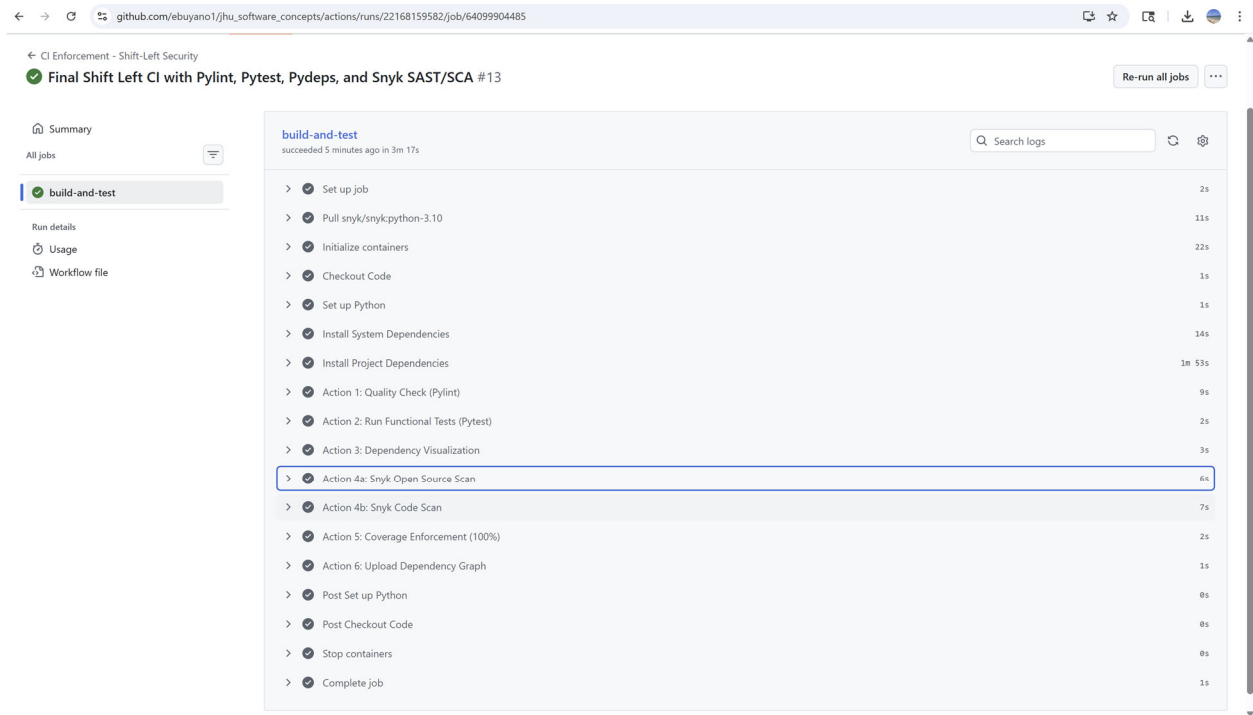
For the diskcache vulnerability (where no patch is available), I implemented a Snyk ignore policy with a 30-day expiry, ensuring the vulnerability is tracked but does not block the CI pipeline while awaiting a maintainer update.

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



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, Snyc Test, Snyc Code Test, Generates dependencies svg . Github Actions shows all passed, Pylint, Pytest, Snyc Test, Snyc Code Test (extra credit), Dependency graph generated and can be downloaded for verification.



The screenshot displays a GitHub Actions workflow run for the repository 'ebuyano1/hu_software_concepts'. The workflow is named 'Final Shift Left CI with Pylint, Pytest, Pydeps, and Snyc SAST/SCA #13' and has a status of 'succeeded 5 minutes ago in 3m 17s'. The workflow consists of several steps, all of which passed successfully. The steps are listed in a table with their names and durations.

Step Name	Duration
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: Snyc Open Source Scan	6s
Action 4b: Snyc 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