Step 2: SQL Injection Defenses

What changed and why it's safe?

The query building and identifier handling was centralized with psycopg.sql instead of string concatenation (db_builders.py). The runtime values were parameterized everywhere with %s + execute(..., params). (query_table.py, website.py, laod_data.py, db_builders.py). The SQL identifiers were restricted to a fixed allowlist (sql.Identifier) instead of accepting arbitrary names. (website.py). LIMIT/OFFSET inputs were constrained to a bounded integer range (1..100) before binding. (db_builders.py, query_table.py). In the SQL admin script, role/database names and passwords are safely quoted with %I / %L and quoted psql vars. (create_least_priviledge_app_user.sql)

This is a safer approach against SQL injection because:

- Bound parameters keep data separate from SQL syntax, so user-controlled text cannot alter query structure.
- Identifier injection is blocked because table/column names come from code-defined constants, not raw input.
- Numeric control fields (LIMIT, OFFSET) are validated and clamped before used
- No SQL path in module_5/src appears to build queries with f-strings or + concatenation from request data.

Step 3: Database Hardening:

What permissions were granted and why?

The granted permissions for grad_cafe_app are:

- Connect on database grad_cafe
- Usage on scheme public
- Select, Insert on table public.applicants
- Usage, Select on sequence public.applicants_p_id_seq (needed for serial inserts)

These were granted while also restricting the following:

- No update/delete on public.applicants
- No owner-level rights (alter/drop will be denied)
- No schema create rights (revoke create on schema public from grad_cafe.app)
- Removed inherited schema create from public to prevent indirect object creation
- Role is explicity non-priviledge (ex. Nosuperuser)
- Grant Connect is to current_database()

These were granted to ensure least privilege operation so that the app opens a database session, resolve and access objects in the schema, read the rows for analysis, and the newly pulled rows, and also insert sequence access for serial p_id. The restrictions were added to reduce risk of the app.

Screenshot of SQL Privileges:

```
SELECT format(
    'GRANT CONNECT ON DATABASE %I TO %I',
    current_database(),
    :'app_user'
) \gexec
GRANT USAGE ON SCHEMA public TO :"app_user";

REVOKE ALL ON TABLE public.applicants FROM :"app_user";
GRANT SELECT, INSERT ON TABLE public.applicants TO :"app_user";

REVOKE ALL ON SEQUENCE public.applicants_p_id_seq FROM :"app_user";
GRANT USAGE, SELECT ON SEQUENCE public.applicants_p_id_seq TO :"app_user";

-- Remove inherited CREATE from PUBLIC and explicitly disallow CREATE for app role.
REVOKE CREATE ON SCHEMA public FROM PUBLIC;
REVOKE CREATE ON SCHEMA public FROM :"app_user";
```

Step 4: Python Dependency Graph

In 5–7 sentences, explain the key dependencies shown and what they do.

The key dependency is app_py, which is the main app module in the python dependency graph. It's direct dependencies are flask, huggingface_hub, and llama_cpp/llama_cpp_llama and numpy/numpy_typing.

Flask is the HTTP framework that handles routing, request parsing, and JSON responses for your API endpoints. Within flask, there are other nodes that manage app context, request lifecycle, and rendierng support.

The huggingface_hub is responsible for downloading/caching the model artifact (your GGUF file) from Hugging Face so the app can run locally.

Lllama_cpp and llama_cpp_Llama is the local inference engine that loads that model and generates standardized outputs.

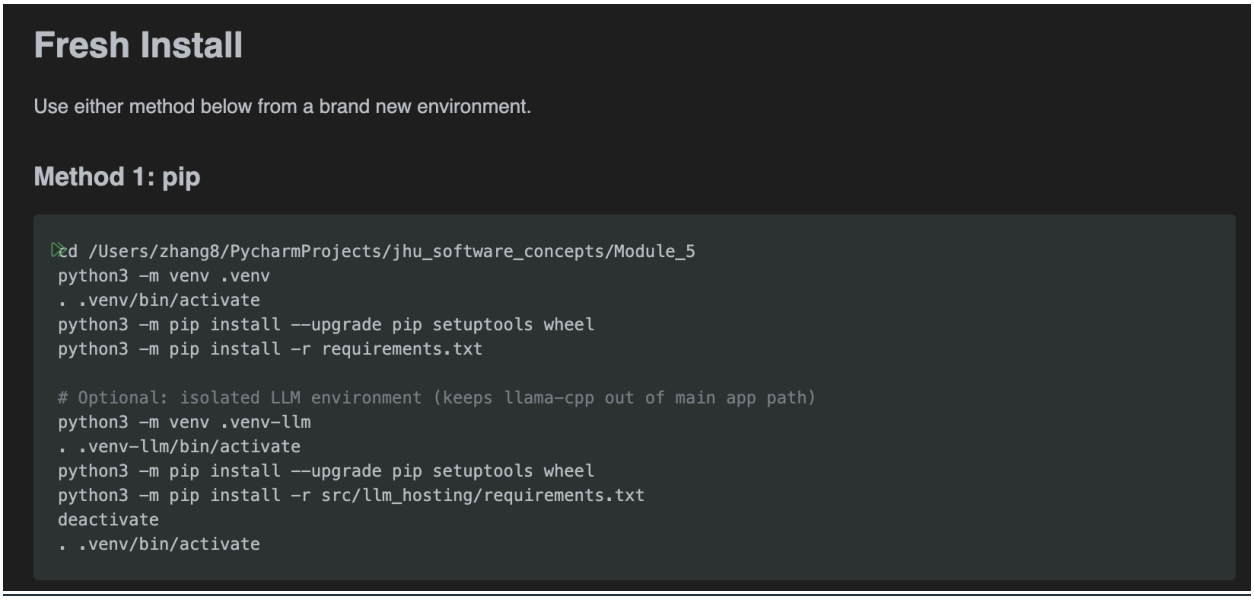
Numpy and numpy_typing is an important indirect supporting dependency used by llama_cpp internals for numeric operations.

Step 5: Reproducible Environment + Packaging

Why packaging matters?

Packaging is important because it makes a project reproducible and runnable by others without guesswork. It makes it clear the exact dependencies/versions to install, creates a standard way to install and run the project, and defines what the code needs. Packaging turns the code from a local folder/computer into something that is shareable and repeatable.

How to install/run via pip and uv (part of README.md)



Fresh Install

Use either method below from a brand new environment.

Method 1: pip

```
cd /Users/zhang8/PycharmProjects/jhu_software_concepts/Module_5
python3 -m venv .venv
. .venv/bin/activate
python3 -m pip install --upgrade pip setuptools wheel
python3 -m pip install -r requirements.txt

# Optional: isolated LLM environment (keeps llama-cpp out of main app path)
python3 -m venv .venv-llm
. .venv-llm/bin/activate
python3 -m pip install --upgrade pip setuptools wheel
python3 -m pip install -r src/llm_hosting/requirements.txt
deactivate
. .venv/bin/activate
```

Method 2: uv

```
cd /Users/zhang8/PycharmProjects/jhu_software_concepts/Module_5
uv venv .venv
. .venv/bin/activate
uv pip install -r requirements.txt

# Optional: isolated LLM environment (keeps llama-cpp out of main app path)
uv venv .venv-llm
. .venv-llm/bin/activate
uv pip install -r src/llm_hosting/requirements.txt
deactivate
. .venv/bin/activate
```

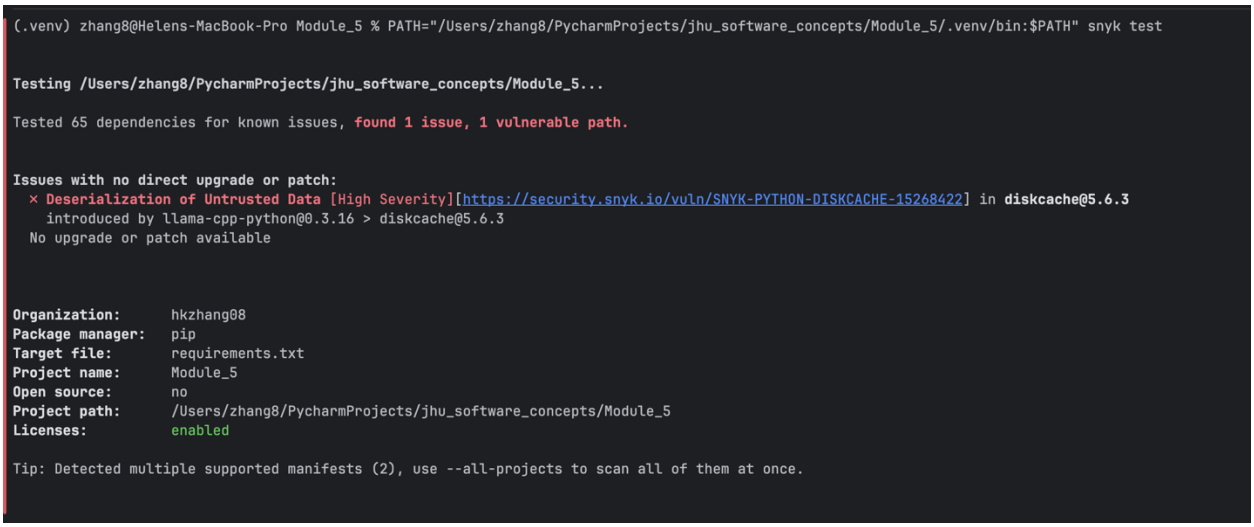
After installing dependencies, configure DB env vars (shown below), then run:

```
flask --app src run
```

Step 6: Snyk Dependency Scan

Required Snyk Test: If vulnerabilities are found, document them and patch/remove packages where applicable.

There was 1 vulnerability found in the snyk test introduced from llama-cpp-python. The patch was to make it optional, so these can be removed from requirements.txt so other environments do not have to install it



Optional Snyk Code Test: Include a screenshot as proof and briefly summarize what it found.

It found that there are untrusted CLI input that can flow into file operations within the app.py program. This means an attacker can control those CLI args and read/write files that your process can access. Although this is intentional – it is flagged on CLI tools. This can be patched to restrict input/output paths to safe base directory.

First Test:

```
(.venv) zhang8@Helens-MacBook-Pro Module_5 % snyk code test

Testing /Users/zhang8/PycharmProjects/jhu_software_concepts/Module_5 ...

Open Issues

 ✗ [MEDIUM] Path Traversal
   Finding ID: 94d37bb6-5553-48df-a462-bd25d2c6671e
   Path: src/llm_hosting/app.py, line 382
   Info: Unsanitized input from a command line argument flows into open, where it is used as a path. This may result in a Path
Traversal vulnerability and allow an attacker to read arbitrary files.

 ✗ [MEDIUM] Path Traversal
   Finding ID: 1ff0d13f-6d70-4ab8-948b-4953645a65b8
   Path: src/llm_hosting/app.py, line 389
   Info: Unsanitized input from a command line argument flows into open, where it is used as a path. This may result in a Path
Traversal vulnerability and allow an attacker to write arbitrary files.

 ✗ [MEDIUM] Path Traversal
   Finding ID: 4b26b392-6e18-4490-98be-2536d2218928
   Path: src/llm_hosting/app.py, line 398
   Info: Unsanitized input from a command line argument flows into json.dump, where it is used as a path. This may result in a
Path Traversal vulnerability and allow an attacker to write arbitrary files.



Test Summary

  Organization:      hkzhang08
  Test type:         Static code analysis
  Project path:      /Users/zhang8/PycharmProjects/jhu_software_concepts/Module_5

  Total issues:   3
  Ignored issues: 0 [ 0 HIGH  0 MEDIUM  0 LOW ]
  Open issues:    3 [ 0 HIGH  3 MEDIUM  0 LOW ]
```

New Test:

```
Testing /Users/zhang8/PycharmProjects/jhu_software_concepts/Module_5 ...


Test Summary

  Organization:      hkzhang08
  Test type:         Static code analysis
  Project path:      /Users/zhang8/PycharmProjects/jhu_software_concepts/Module_5

  Total issues:   0

 Tip

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