# Grad-Cafe App - Dependency Guide (Windows)

## OVERVIEW

This document lists the dependencies required to run the Grad-Cafe Flask app (module_5/src/app.py), generate the pydeps graph, and execute the end-to-end data pipeline on Windows.

## PYTHON & RUNTIME

* Python 3.11+ (your project uses 3.13). Create and activate a virtual environment.
  PowerShell:
    python -m venv .venv
    .\.venv\Scripts\Activate.ps1
* pip (bundled with Python) to install project libraries.

## PYTHON LIBRARIES (import-based)

* Flask (and its transitive deps: Jinja2, Werkzeug, Click, ItsDangerous) - Used for routing, request handling, and rendering the Analysis page.
* psycopg (3.2.x) and optionally psycopg_pool
  - Used by query_data.py and load_data.py to talk to PostgreSQL. *
configparser, typing, functools, json, os, sys, subprocess (standard library)
- Config, type hints, and orchestration of external steps.
* pydeps (dev tool) - to visualize import dependencies for app.py
  - Install with: pip install pydeps
* pylint (dev tool) – SAST scanning
  - Install with: pip install pylint
* pytest, pytest-cov (testing tool) – Unit and integration testing
  - Install with: pip install pytest

## SYSTEM DEPENDENCIES

* Graphviz (provides the 'dot' executable required by pydeps to render graphs)
- Install one of:
    winget install Graphviz.Graphviz -e --accept-source-agreements --accept-packageagreements
    choco install graphviz -y
    (or) MSI from the Graphviz website
- Ensure PATH contains: C:\Program Files\Graphviz\bin
- Verify: dot -V

## DATABASE / SERVICES

* PostgreSQL server accessible from your machine
- Required schema: public.applicants (per Module-3 spec)
- Provide connection settings via config.ini ([db] or [database] depending on your design).
* (Optional) LLM hosting (Module-2 reference) if you trigger the LLM standardization step during Pull Data.

## OPERATIONAL (runtime) DEPENDENCIES (not import-based)

* The ETL pipeline stages are launched via subprocess from app.py:
- module_2\scrape.py  - scrapes new rows
- module_2\clean.py   - cleans/normalizes scraped fields
- module_3\load_data.py - loads rows into PostgreSQL
* Because they are spawned (not imported), pydeps will NOT include these files in the module graph. They are real runtime dependencies your environment must support (correct paths, permissions, Python available, and DB reachable).

**Below is the breakdown of what's in my app.py dependency graph and how each part is used:**

1. app.py (root node) — The Flask entry point that wires routes and the Part-B buttons; it orchestrates reading analysis rows and kicking off pipeline actions.

2. Flask (package node) — Provides Flask, render_template, request, redirect, and url_for; I use it for routing, form handling, and rendering the Analysis page.

3. query_data (local module) — I import get_rows() from here to fetch the analysis tuples (question-answer) so the web layer stays thin and the SQL logic is isolated.

4. subprocess (stdlib) — Used to run the scraper/cleaner/loader as external processes, capturing stdout/stderr so the web thread isn't blocked by long-running work.

5. os / sys (stdlib) — Used for path resolution, environment/venv interpreter selection, and robust invocation of the right Python executable for the pipeline.

6. Arrow direction in the graph: an arrow from A TO B means "A imports B.", edges point from the importer to the imported module (dependency direction).
Missing arrows mean there's no import relationship.

7. How that maps to request flow: When a user hits a Flask route in app.py, code is already imported. The route handler then calls functions (e.g., get_rows() from query_data) or launches subprocesses for scrape.py, clean.py, load_data.py. The arrows show that app.py can see query_data because it imports it, but there's no arrow to the pipeline scripts because they're spawned, not imported.

8. Transitive dependencies: If app.py -> query_data and query_data -> psycopg, then app.py transitively depends on psycopg. In the diagram, we can see a path reflecting this chain; pydeps may or may not draw a direct shortcut edge.

What the graph does not show: Execution order, HTTP call sequencing, or runtime branching. It did not depict ETL steps because scrape/clean/load_data are external processes; they're part of the runtime pipeline flow, not the import graph.

***Finally, please note that scrape.py, clean.py, and load_data.py do not appear in the diagram because they're executed via subprocess (operational dependencies) rather than imported (no import edge, so pydeps omits them).