**Asset Management Dashboard — Architecture & Operations**

**1) Overview**

The dashboard is a **Flask web app** that:

* Authenticates users against **SQLite** (User\_control.db, table users).
* Renders a **UBC‑styled dashboard** with three application tiles:
  + Asset Capture Mobile App (port 5001)
  + Asset Reviewer – Mechanical (port 5002)
  + Asset Reviewer – Backflow Devices (port 5003)
* Exposes **whitelisted “Run AI Interpreter” tasks** that launch server‑side Python scripts **asynchronously** and log their output.

**Key files**

* Asset-portal-dashboard.py — Flask app (login, dashboard, task runner, log viewer)
* templates/login.html — login page
* templates/dashboard.html — dashboard page
* static/style.css — UBC styling
* static/logos/\* — logos
* logs/ — task output logs (auto‑created)

**2) Authentication & Session Flow**

**Database**

* **Path**:  
  S:\MaintOpsPlan\AssetMgt\Asset Management Process\Database\8. New Assets\Git\_control\asset\_capture\_app\_dev\data\User\_control.db
* **Table**: users  
  Recommended columns:
  + id (INTEGER PRIMARY KEY)
  + username (TEXT, unique)
  + password\_hash (TEXT)
    - Accepts **real hashes** (pbkdf2:...) **or** **plaintext** (temporary/legacy, as per Option A).
  + is\_active (INTEGER 0/1)
  + (optional) role (TEXT)

Current configuration uses **Option A**: if password\_hash doesn’t look like a hash, it’s treated as plaintext for backward compatibility.

**Login routes**

* GET /login — renders the form.
* POST /login — validates username/password against SQLite:
  + Checks is\_active.
  + If password\_hash starts with pbkdf2:, scrypt:, or argon2:, verifies with werkzeug.security.check\_password\_hash.
  + Otherwise compares as plaintext (temporary).
  + On success: stores session['user'] = {'username': ...} and redirects to /.
  + On failure: flashes “Invalid username or password.”

**Session protection**

* A login\_required decorator protects:
  + GET / (dashboard)
  + POST /run/<task\_key> (task execution)
  + GET /logs, GET /logs/view (log viewer)
* POST /logout clears the session and redirects to /login.

**3) Dashboard UI**

* templates/dashboard.html pulls apps=APPS from Flask and renders cards.
* Each card shows:
  + **Open** — external link to the corresponding app (ports 5001/5002/5003).
  + For Mechanical/Backflow cards, a **“Run AI Interpreter”** button that **POSTs** to /run/<task\_key>.

**Task keys**

* qr\_api\_me → API interface\_ME\_ver00.py
* qr\_api\_bf → API interface\_BF\_ver00.py

**4) Task Execution (Server‑side)**

**Whitelist**

In Asset-portal-dashboard.py:

TASKS = {

"qr\_api\_bf": r"S:\...\QR\_code\_project\_API\API interface\_BF\_ver00.py",

"qr\_api\_me": r"S:\...\QR\_code\_project\_API\API interface\_ME\_ver00.py",

}

Only these keys can be invoked from the UI.

**Launcher (asynchronous, UTF‑8 safe)**

* Uses the **same Python interpreter** as the Flask app (sys.executable).
* Runs with -X utf8 and environment variables PYTHONIOENCODING=utf-8, PYTHONUTF8=1.
* Stdout/stderr are written to ./logs/<script-stem>.<timestamp>.log (UTF‑8).

**What the user sees**

* On POST, the dashboard quickly returns (non‑blocking) with a flash:  
  Started task 'qr\_api\_\*'. Logs: <filename>.log
* You can read logs via:
  + GET /logs (lists recent)
  + GET /logs/view?name=<filename>.log (opens a specific log)

In your AI scripts (API interface\_\*.py), the first lines reconfigure stdout/stderr to UTF‑8 so emojis and special characters print safely on Windows.

**5) File & Folder Layout (suggested)**

Dashboard/

├─ Asset-portal-dashboard.py

├─ logs/ # auto-created task logs

├─ templates/

│ ├─ login.html

│ └─ dashboard.html

└─ static/

├─ style.css

└─ logos/

├─ ubc\_logo.jpg

└─ ubc-facilities\_logo.jpg

Keep the AI interpreter scripts in your existing project folders on S:\... as you already do.

**6) Configuration & Environment**

* **Flask secret key**: set an env var in production:
* set FLASK\_SECRET\_KEY=some-long-random-string
* **Python environment**: make sure the Flask app and your AI scripts share the same interpreter & packages.
* **OpenAI key** (for your AI scripts): loaded via your existing .env file; ensure the Flask process can read the path.
* **Tesseract**: verify pytesseract.pytesseract.tesseract\_cmd path is valid on the server account.

**7) Running Locally (Windows)**

From Dashboard/:

python Asset-portal-dashboard.py

Open: http://127.0.0.1:5080

Ensure the three downstream apps are reachable (5001/5002/5003), or the **Open** links will 404 (that’s OK for testing the dashboard itself).

**8) Deployment (Ubuntu notes)**

* Install Python 3.x, create a venv, install Flask + deps.
* Map your Windows shares (or sync project files) so the server can access:
  + User\_control.db
  + AI scripts and their input/output folders
* If you’ll run behind Nginx:
  + Serve Flask via **gunicorn** or **uWSGI**.
  + Keep the async launcher and logs the same (paths must be reachable by the service account).
* Set environment variables in systemd service file:
  + FLASK\_SECRET\_KEY
  + Any proxy or OpenAI vars your scripts require.

**9) Security Considerations**

* **Whitelisting** prevents arbitrary code execution.
* **Login required** for dashboard, run-task, and logs.
* **Option A (plaintext compatibility)** is temporary. Plan a migration to **hashed passwords**:
  + Generate hashes with werkzeug.security.generate\_password\_hash("password").
  + Replace password\_hash values with those hashed strings.
  + Then switch validate\_password to **require** hashed values only.

**10) Troubleshooting**

* **“Invalid username or password.”**
  + Check users.username spelling and is\_active = 1.
  + If password\_hash holds plaintext (“test”), Option A will accept it. If you moved to real hashes, ensure the hash string begins with pbkdf2: (or scrypt:, argon2:) and was generated correctly.
* **BuildError: Could not build url for endpoint 'index'**
  + Fixed by registering / with endpoint="index" in Asset-portal-dashboard.py.
* **UnicodeEncodeError in logs**
  + Ensure the launcher passes UTF‑8 env vars and -X utf8.
  + Ensure your AI script reconfigures stdout/stderr to UTF‑8 at the top.
* **Buttons stacked vertically**
  + In dashboard.html, use d-flex gap-2 flex-wrap for the button container.
* **CSS/Logo not loading**
  + Verify paths in the template:
    - {{ url\_for('static', filename='style.css') }}
    - {{ url\_for('static', filename='logos/ubc\_logo.jpg') }}

**11) Future Enhancements**

* **Role‑based access** (e.g., only certain users can run tasks).
* **Task status panel**: show the latest log inline and auto‑refresh for a few seconds after launch.
* **Job queue** (Celery/RQ) if you need retries, progress, or concurrency limits.
* **Audit log** table: who ran which task and when.
* **Switch to hashed passwords** and drop plaintext compatibility.