Exercise: Linting, Testing & Git Hooks for a Flask App (Local & GitHub)

requirements

1. Set up local linting with `ruff`.
2. Add one unit test and one integration test.
3. Configure Git hooks so that linting and tests run automatically before each `git push`.
4. Configure GitHub to run linting and tests on pull requests targeting the `dev` branch.

---

1. Project Setup

- Create and activate a virtual environment.
- Install dependencies plus the tools you will use:
  - Flask (already in requirements.txt)
  - `ruff` for linting
  - `pytest` (and optionally `pytest-cov`)

```
python -m venv .venv
source .venv/bin/activate  # on Windows: .venv\Scripts\activate
pip install -r requirements.txt
pip install ruff pytest
```

Update `requirements.txt` to include:

- `ruff`
- `pytest`

---

2. Configure Ruff Linting

1. Add a `ruff` configuration file at the project root (`pyproject.toml` or `ruff.toml`).

```
[tool.ruff]
target-version = "py311"
line-length = 88
select = ["E", "F", "I"]
ignore = []
src = ["app"]
```

2. Confirm linting works:

```
ruff check app
```

The command must exit with status 0 when there are no linting issues.

---

3. Add Tests (Unit and Integration)

3.1. Unit Test

- Create a `tests` directory if it does not exist: `tests/`.
- Add a file `tests/test_unit_example.py`.
- Write a unit test for a small, pure function or class method in our Flask app.

3.2. Integration Test

- Add a file `tests/test_integration_example.py`.
- Use Flask's test client to test at least one HTTP endpoint (for example, `GET /health` or `GET /`).

Example skeleton:

```
# tests/test_integration_example.py

from app import create_app  # or app instance, depending on your project
structure

def test_health_endpoint():
    app = create_app()
    client = app.test_client()

    response = client.get("/health")
    assert response.status_code == 200
    assert response.get_json()["status"] == "ok"
```

(Adapt to your actual app factory or app instance.)

3.3. Run Tests Locally

```
pytest
```

All tests must pass before moving on.

---

4. Git Hooks: Run Ruff and Tests Before Push

You must configure Git hooks so that:

- Before `git push`, the following commands are run:
    - `ruff check`

    - `pytest`

- If either command fails, the push must be blocked.

4.1. Pre-push Hook

1. Create `.git/hooks/pre-push`:

```
#!/usr/bin/env bash
set -e

echo "Running ruff..."
ruff check app

echo "Running tests..."
pytest

echo "All checks passed. Proceeding with push."
```

2. Make the hook executable:

```
chmod +x .git/hooks/pre-push
```

3. Verify behavior:

- Introduce a linting or test error.
- Try to `git push` and confirm it fails.
- Fix the issue and confirm `git push` succeeds.

---

5. GitHub CI on Pull Requests to dev

You must configure GitHub to run the same checks (ruff + pytest) on pull requests targeting the `dev` branch.

5.1. GitHub Workflow

1. In your repo, create a directory: `.github/workflows/`.
2. Add a workflow file: `.github/workflows/ci.yml`.

5.2. Behavior Requirements

- When a pull request is opened or updated with `base: dev`:
    - The workflow must run.
    - The workflow must fail if:
        - Ruff finds linting errors.

        - Tests fail.

- The PR must show a failing status check in GitHub if either step fails.

---

6. Deliverables

To complete the exercise, you must:

1. provide a link to your github repo with everything pushed
2. Demonstrate:
    - provide a screenshor of your local githook output
    - provide a screenshor of your completed github workflow