Flask Application – Full Dev-to-Prod Pipeline Exercise

This exercise guides you through setting up a Flask application for **local development**, **E2E testing**, **containerized production deployment**, and a **GitHub CI/CD pipeline** that builds and pushes production Docker images to Dockerhub.

---

1. Local Setup (Development Environment)

- Install Dependencies

```
pip install -r requirements.txt
```

Install PostgreSQL locally (via Postgres.app, Homebrew, or direct installer) and create a local database for dev.

Install **pgAdmin** to manage databases.

```
- Name: `taskmanager`
- Owner: your local PostgreSQL user
```

- Configure Local `.env` to reflect your username and password
- Migrate Database

```
python migrate.py
```

- Run the App

```
python app.py
```

The app should now run locally with a local PostgreSQL instance.

---

2. Writing Tests

Your test suite must include:

- 3 Unit Tests

Focus on pure Python logic, model methods, and helpers.

Examples:

1. `Task.is_overdue()` logic
2. `User.set_password()` & `check_password()`
3. `_build_postgres_uri()` environment parsing

Create file: `tests/test_unit.py`

- 3 Integration Tests

Use Flask's test client and an in-memory or temporary PostgreSQL database.

Examples:

1. Register + login flow through the API
2. Creating a task via POST
3. Editing or toggling a task

Create file: `tests/test_integration.py`

- 3 End-to-End Tests (Selenium)

Examples:

1. Visit login page → login → verify redirect
2. Create task through UI → verify it appears
3. Toggle or delete task through UI

Create file: `tests/test_e2e.py`

You can run e2e tests with:

```
pytest tests/test_e2e.py
```

---

3. Production Setup

- Create `.env.production`

This environment should not be committed to Git.

```
FLASK_ENV=production
SECRET_KEY=your-production-secret-key

# and the production DB credentials
```

- NeonDB Database Setup

Create an account at: https://neon.tech/

Create a new Postgres project and copy your **connection config**, which goes into the production `.env`.

---

4. Dockerizing the App

Create a file `Dockerfile` for production.

Note: When deployed, the container will receive `SECRET_KEY` and `DATABASE_URL` through environment variables injected by your CI/CD pipeline or runtime environment.

---

5. Dockerhub Setup

5.1 Create an Account

Create an account on: https://hub.docker.com/

5.2 Create an Access Token

Under *Account Settings* → *Security* → *New Access Token*

Store the token securely; you will add it to GitHub Secrets as:

```
DOCKERHUB_USERNAME
DOCKERHUB_TOKEN
```

---

6. GitHub CI/CD Pipeline

Create workflow file `.github/workflows/ci.yml`:

6.1 On Pull Request (dev, main) and Push to dev

- Lint the code
- Run unit/integration/e2e tests
- Build Docker image (just to validate that it builds)

6.2 On Push to Main

- Build Docker image
- Push to Dockerhub using secrets

6.3 Add Secrets to GitHub Repo

Add:

```
ENV_VARIABLES
DOCKERHUB_USERNAME
DOCKERHUB_TOKEN
```

---

7. Project Deliverables

7.1 A PDF or Word including:

- Link to **Dockerhub repo**
- Screenshot of Docker image pushed
- Link to **GitHub repo**
- Screenshot of successful CI/CD build pipeline

7.2 Video Recording

Record a short video showcasing:

- Code push
- CI/CD pipeline running on GitHub
- Docker image being pushed to Dockerhub

You may use **Loom** or any screen-recording tool.
