Here’s an updated and more detailed workflow that covers various development scenarios, including model and schema changes, Flask app code updates, dependency installations, and database content changes. It also outlines the process for testing the app on Heroku’s front end.

### **Comprehensive Development Workflow:**

This guide explains how to handle different development tasks in a Flask app with PostgreSQL. We will cover these situations: 1. **Developer updates models and schema** 2. **Developer updates only Flask app code** 3. **Developer installs new dependencies** 4. **Developer modifies local database contents (adds/deletes users, etc.)** 5. **Testing front-end code on Heroku**

Each scenario involves development locally on the development branch and deployment to Heroku via the stable-versions branch.

### **1. Developer Updates Models and Schema**

When you modify the database schema (e.g., add new fields, models, or relationships):

#### **Step 1: Modify Models Locally**

* Update your models in the Flask app to reflect the new schema.

#### **Step 2: Create and Apply Migration Locally**

* Generate and apply Alembic migration: bash alembic revision --autogenerate -m "Update schema" alembic upgrade head
* This updates your local PostgreSQL database schema.

#### **Step 3: Commit and Push to development Branch**

* Commit your changes (including migration files) and push to the development branch: bash git add . git commit -m "Update models and apply migrations" git push origin development

#### **Step 4: Merge development into stable-versions**

* Merge development into stable-versions: bash git checkout stable-versions git merge development git push origin stable-versions

#### **Step 5: Apply Migrations on Heroku**

* Apply the migrations on Heroku: bash heroku run alembic upgrade head --app ws-companion

#### **Step 6: Sync Local Database to Heroku (Optional)**

* If you want to ensure that your local data matches Heroku, you can run the database sync with truncate: bash pgsync

### **2. Developer Updates Only Flask App Code**

When you modify the Flask app code (but not the database schema):

#### **Step 1: Make Code Changes Locally**

* Make the necessary updates to your Flask app (e.g., route changes, adding new features).

#### **Step 2: Commit and Push to development Branch**

* Commit your changes to the development branch: bash git add . git commit -m "Update Flask app code" git push origin development

#### **Step 3: Merge development into stable-versions**

* Merge development into stable-versions: bash git checkout stable-versions git merge development git push origin stable-versions

#### **Step 4: Heroku Deployment**

* Heroku will automatically deploy the changes from the stable-versions branch.

### **3. Developer Installs New Dependencies**

When new dependencies (e.g., Flask extensions, Python libraries) are added to the project:

#### **Step 1: Install New Dependencies Locally**

* Install the new dependencies using pip: bash pip install <new\_dependency>

#### **Step 2: Update requirements.txt**

* Update the requirements.txt file with the new dependencies: bash pip freeze > requirements.txt

#### **Step 3: Commit and Push to development Branch**

* Commit the updated requirements.txt to the development branch: bash git add requirements.txt git commit -m "Add new dependencies" git push origin development

#### **Step 4: Merge development into stable-versions**

* Merge development into stable-versions: bash git checkout stable-versions git merge development git push origin stable-versions

#### **Step 5: Install Dependencies on Heroku**

* Heroku automatically installs new dependencies from the updated requirements.txt during the next deployment.

### **4. Developer Modifies Local Database Contents (Adds/Deletes Users, etc.)**

When you add or delete users or modify database contents (without changing the schema):

#### **Step 1: Modify Database Locally**

* Modify the database by adding/deleting users or updating contents via the app, admin interface, or psql.

#### **Step 2: Sync Local Database to Heroku with Truncate (Optional)**

* Sync the local database to Heroku, ensuring the changes are reflected: bash pgsync

This will truncate the Heroku database and replace it with the local data, ensuring both environments match.

### **5. Testing Front-End Code on Heroku**

When the developer wants to test only front-end code changes on Heroku:

#### **Step 1: Make Front-End Changes Locally**

* Update the front-end code (HTML, CSS, JS) and test locally.

#### **Step 2: Commit and Push to development Branch**

* Commit the front-end code changes to the development branch: bash git add . git commit -m "Update front-end code" git push origin development

#### **Step 3: Merge development into stable-versions**

* Merge development into stable-versions: bash git checkout stable-versions git merge development git push origin stable-versions

#### **Step 4: Test on Heroku**

* Heroku will deploy the changes automatically. You can now test the front-end code on Heroku.

### **Final Workflow Summary:**

1. **Model/Schema Changes**: Make changes locally, apply migrations, and push changes to Heroku.
2. **Flask App Code Updates**: Modify Flask code locally, commit, merge, and deploy to Heroku.
3. **New Dependencies**: Install locally, update requirements.txt, and Heroku will handle the installation on deployment.
4. **Database Content Changes**: Modify local database contents, then sync to Heroku with truncate.
5. **Front-End Testing**: Modify and test front-end code locally, then push to Heroku for deployment and testing.

This workflow ensures a smooth development process while keeping local and Heroku environments in sync. Let me know if you need further customization or clarification!