Pourfect-AI Milestone 4 Documentation

Testing Documentation

The main testing framework used to run unit tests for **datapipeline** was Pytest. Using three test files we validated the results of preprocess_rag, preprocess_data and finetuning_data. The testing strategy for these test files included mocking and assertions. Many of our external dependencies (i.e. GCP, databases) were mocked to isolate the functionality being tested. Assertions were used to check if the functions were called with the correct arguments and to verify if the function outputs matched their expected.

The main testing framework used to run unit tests for **models** was Pytest. Using two test files we validated the results of train_model and model_rag. The testing strategy for these test files included mocking and assertions. Many of our external dependencies (i.e. GCP, Vertex AI, chromaDB) were mocked to isolate the functionality being tested. Assertions were used to check if the functions were called with the correct arguments and to verify if the function outputs matched their expected. For example, we tested the chatting functionality with the mocked model to ensure that it would output a response to the user.

The main testing framework used to run unit tests for **api-service** was Pytest. Using two test files we validated the results of utils_llm_rag and utils_chat. The testing strategy for these test files included mocking and assertions. Many of our external dependencies (i.e. GCP, Vertex AI, chromaDB) were mocked to isolate the functionality being tested. Assertions were used to check if the functions were called with the correct arguments and to verify if the function outputs matched their expected. For example, we tested generating embeddings for input text to ensure the embeddings are of the correct length and we tested generating a chat response from a sample input text to ensure the model outputted a feasible response.

The main testing framework used to run unit tests for **frontend-simple** was Pytest. Using one test file we validated the html setup of our front end. The testing strategy for this test file included ensuring that the html of the website matched what we were expecting. For example, we tested that all of the navigation links were rendering as expected, that the hero section and buttons were correct with the correct text included, and that the title page was correct with the correct title: “Pourfect AI.”

The main testing framework used to run the **integration** tests was Pytest. The integration tests check that our api server can be reached locally and that the chatbot can give a correct mojito recipe with key words such as “rum”, “mint”, and “lime”. It also ensures that the response structure is valid, including that it is a list with the appropriate sections.

To run the tests locally, you can follow the steps outlined below. These steps assume you're working in a Python environment and using unittest along with pipenv for dependency management.

1. You need the following prereqs: python, pipenv, Docker
2. Clone the repo: https://github.com/nikimekstrom/AC215_GitGirls.git

Testing datapipeline:

1. Install project dependencies:
    a. cd /path/to/your/project/src/datapipeline
    b. pipenv install --dev
2. Install docker
3. Build and run the docker container
    a. docker build -t my-datapipeline-image .
    b. docker run --name datapipeline-container -d my-datapipeline-image
4. Run the tests locally
    a. pipenv run pytest ../../tests/test_datapipeline* --cov=./ --cov-report html
5. Verify the html report is above 50%
    a. ls -R src/datapipeline/htmlcov
6. Clean up the docker containers
    a. docker stop datapipeline-container
    b. docker rm datapipeline-container

Datapipeline Coverage Report:

```
▶ Run pipenv run pytest ../../tests/test_datapipeline* --cov=./ --cov-report=term --cov-report=html
============================= test session starts ==============================
platform linux -- Python 3.12.7, pytest-8.3.3, pluggy-1.5.0
rootdir: /home/runner/work/AC215_GitGirls/AC215_GitGirls
plugins: anyio-4.6.2.post1, cov-6.0.0
collected 13 items

../../tests/test_datapipeline_finetuning.py ....                        [ 30%]
../../tests/test_datapipeline_preprocess_data.py ...                    [ 53%]
../../tests/test_datapipeline_preprocess_rag.py ......                  [100%]

---------- coverage: platform linux, python 3.12.7-final-0 -----------
Name                 Stmts   Miss  Cover
----------------------------------------
__init__.py              0      0   100%
finetuning_data.py      97     63    35%
preprocess_data.py      33      9    73%
preprocess_rag.py       79     12    85%
----------------------------------------
TOTAL                  209     84    60%
Coverage HTML written to dir htmlcov


============================== 13 passed in 8.16s ==============================
```

Testing models:

1. Install project dependencies:
    a. cd /path/to/your/project/src/models
    b. pipenv install --dev

   Build and run the docker container
    c. docker build -t my-models-image .
    d. docker run --name models-container -d my-models-image
2. Run the tests locally
    a. pipenv run pytest ../../tests/test_models* --cov=./ --cov-report html

Models Coverage Report:

```
▶ Run pipenv run pytest ../../tests/test_models* --cov=./ --cov-report=term --cov-report=html
============================= test session starts ==============================
platform linux -- Python 3.12.7, pytest-8.3.3, pluggy-1.5.0
rootdir: /home/runner/work/AC215_GitGirls/AC215_GitGirls
plugins: anyio-4.6.2.post1, cov-6.0.0
collected 5 items

../../tests/test_models_model_rag.py ...                                 [ 60%]
../../tests/test_models_train_model.py ..                                [100%]

---------- coverage: platform linux, python 3.12.7-final-0 -----------
Name             Stmts   Miss  Cover
----------------------------------
model_rag.py        54     19    65%
train_model.py      51     13    75%
----------------------------------
TOTAL              105     32    70%
Coverage HTML written to dir htmlcov


============================== 5 passed in 6.90s ===============================
```

Testing api-service:

1. Install project dependencies:
    a. cd /path/to/your/project/src/api-service
    b. pipenv install --dev

    Build and run the docker container
    c. docker build -t my-api-service-image .
    d. docker run --name api-service-container -d my-api-service-image
2. Run the tests locally
    a. cd /path/to/your/project/src/api-service/api/utils
    b. pipenv run pytest ../../../../tests/test_utils* --cov=./ --cov-report=term --cov-report=html

API Service Coverage Report:

```
▶ Run pipenv run pytest ../../../../tests/test_utils* --cov=./ --cov-report=term --cov-report=html
============================= test session starts ==============================
platform linux -- Python 3.12.7, pytest-8.3.3, pluggy-1.5.0
rootdir: /home/runner/work/AC215_GitGirls/AC215_GitGirls
plugins: anyio-4.6.2.post1, cov-6.0.0
collected 7 items

../../../../tests/test_utils_chat.py ...                                [ 42%]
../../../../tests/test_utils_llm_rag.py ....                            [100%]

---------- coverage: platform linux, python 3.12.7-final-0 -----------
Name               Stmts   Miss  Cover
------------------------------------
chat_utils.py         54     11    80%
llm_rag_utils.py      51      5    90%
------------------------------------
TOTAL                105     16    85%
Coverage HTML written to dir htmlcov


============================== 7 passed in 2.39s ===============================
```

Testing frontend-simple:

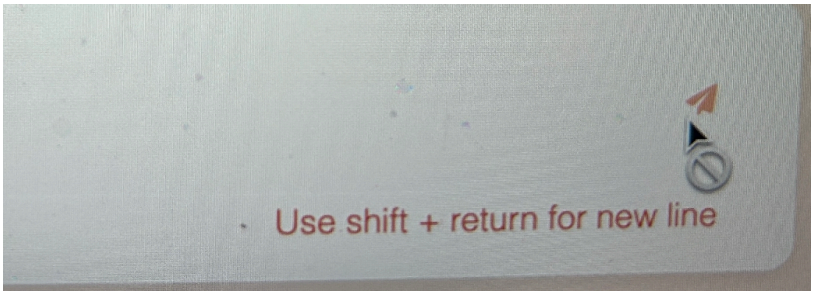
1. Install project dependencies:
    a. cd /path/to/your/project/src/frontend-simple
    b. pipenv install --dev

   Build and run the docker container
    c. docker build -t my-frontend-image .
    d. docker run --name frontend-container -d my-frontend-image
2. Run the tests locally
    a. cd /path/to/your/project/src/api-service/frontend-simple/utils
    b. pipenv run pytest ../../tests/test_frontend*

Frontend-Simple Passing Tests (no coverage report):

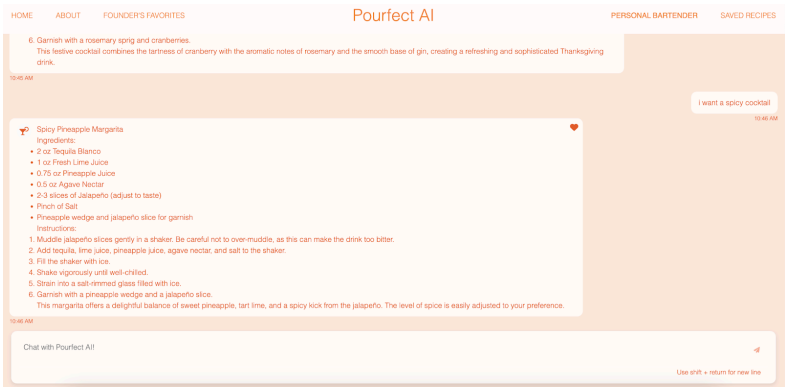
```
▶ Run pipenv run pytest ../../tests/test_frontend*
============================= test session starts ==============================
platform linux -- Python 3.12.7, pytest-8.3.3, pluggy-1.5.0
rootdir: /home/runner/work/AC215_GitGirls/AC215_GitGirls
plugins: cov-6.0.0
collected 5 items

../../tests/test_frontend_simple.py .....                                [100%]

============================== 5 passed in 1.40s ===============================
```

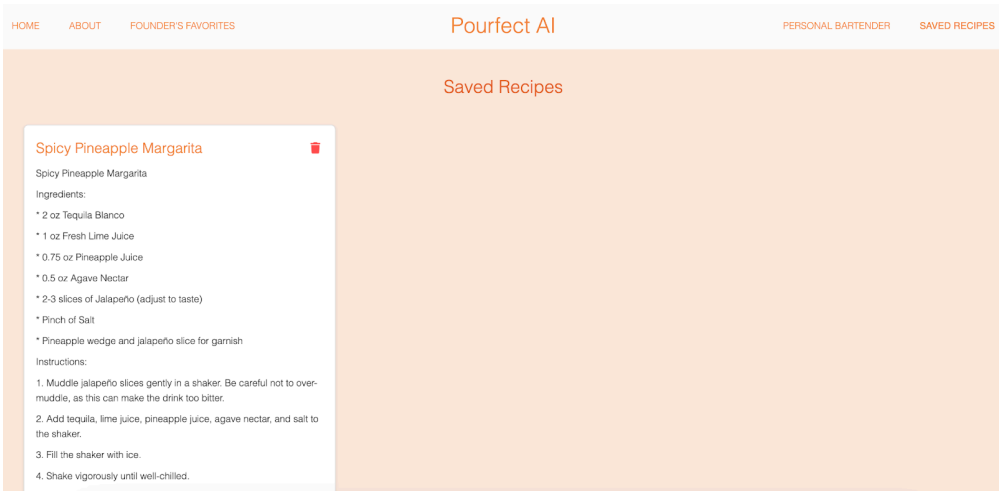
Manual testing frontend:

- Unable to press send button on chat page when there is no input text
    - 

- Input box clears after pressing enter
- Integration test – after typing a message and clicking send in the input box, the frontend renders the new message in the chat display and clears the input box

- 

- Saves recipe from LLM's response in the 'SAVED RECIPES' tab if you click on the heart
- 

- 

Integration tests:

1. Start Datapipeline Docker container:
    a. cd /path/to/your/project/src/datapipeline
    b. sh docker-shell.sh
    c. python preprocess_rag.py
2. Start API Service Docker container (new terminal):
    a. cd /path/to/your/project/src/api-service
    b. sh docker-shell.sh
    c. uvicorn_server
3. Run the tests locally (new terminal)
    a. cd /path/to/your/project/
    b. pipenv run pytest -v tests/test_integration.py