**CS673 Software Engineering (**

**Team 5 - FaFi**

**Software Test Document**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| Brendan Truong | QA Leader | *Brendan Truong* | 09/26/2022 |
| Aidan Chang | Design and Implementation | *Aidan Chang* | 09/26/2022 |
| Derric Syme | Configuration leader | *Derric Syme* | 9/26/2022 |
| Patounezambo Ouedraogo | Requirement Leader | *Patounezambo* | 9/26/2022 |
| Zengrui Luo | Security leader | *Zengrui Luo* | 9/26/2022 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
| **1** | **Brendan Truong** | **09/26/22** | **Update to Add Iteration 1 implementation** |
|  |  |  |  |

[Testing Summary](#_sm5odwyvuk3j)

[Manuel Tests Reports](#_pqso2mbjyzx4)

[Automated Testing Reports](#_mtfbusfb0eq3)

[Testing Metrics](#_rijyjeu2ojqa)

[References](#_15tmymhipvdv)

[Glossary](#_8n34lvocupub)

# Testing Summary

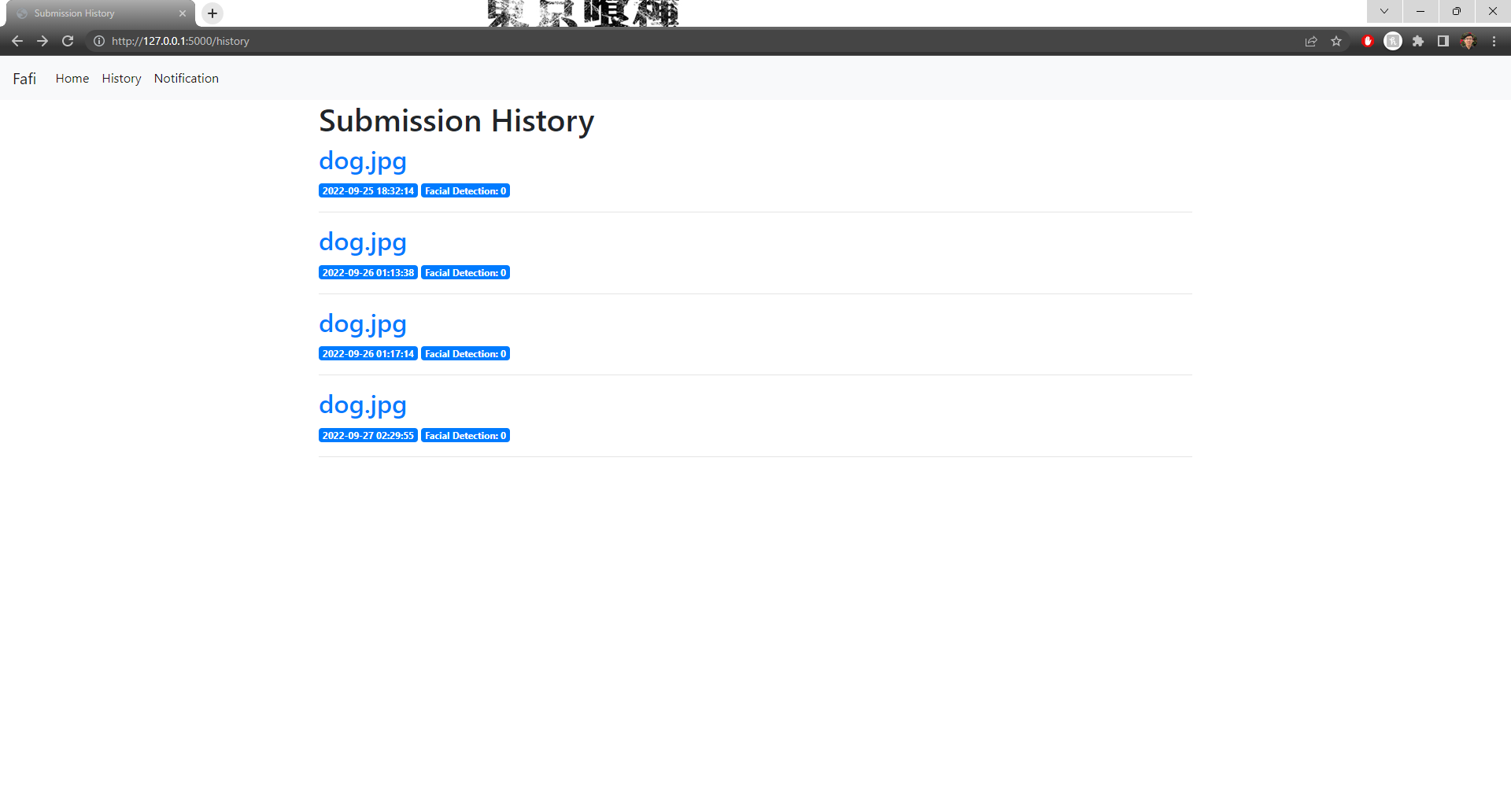
Tests will be written in PyTest. Each team member is expected to create unit tests for functionalities that they develop. The QA leader will provide special focus in integration testing of front end UI and the backend along with performance testing of the end to end application from a user experience perspective. The QA leader is responsible for ensuring the test suite executes without failure.

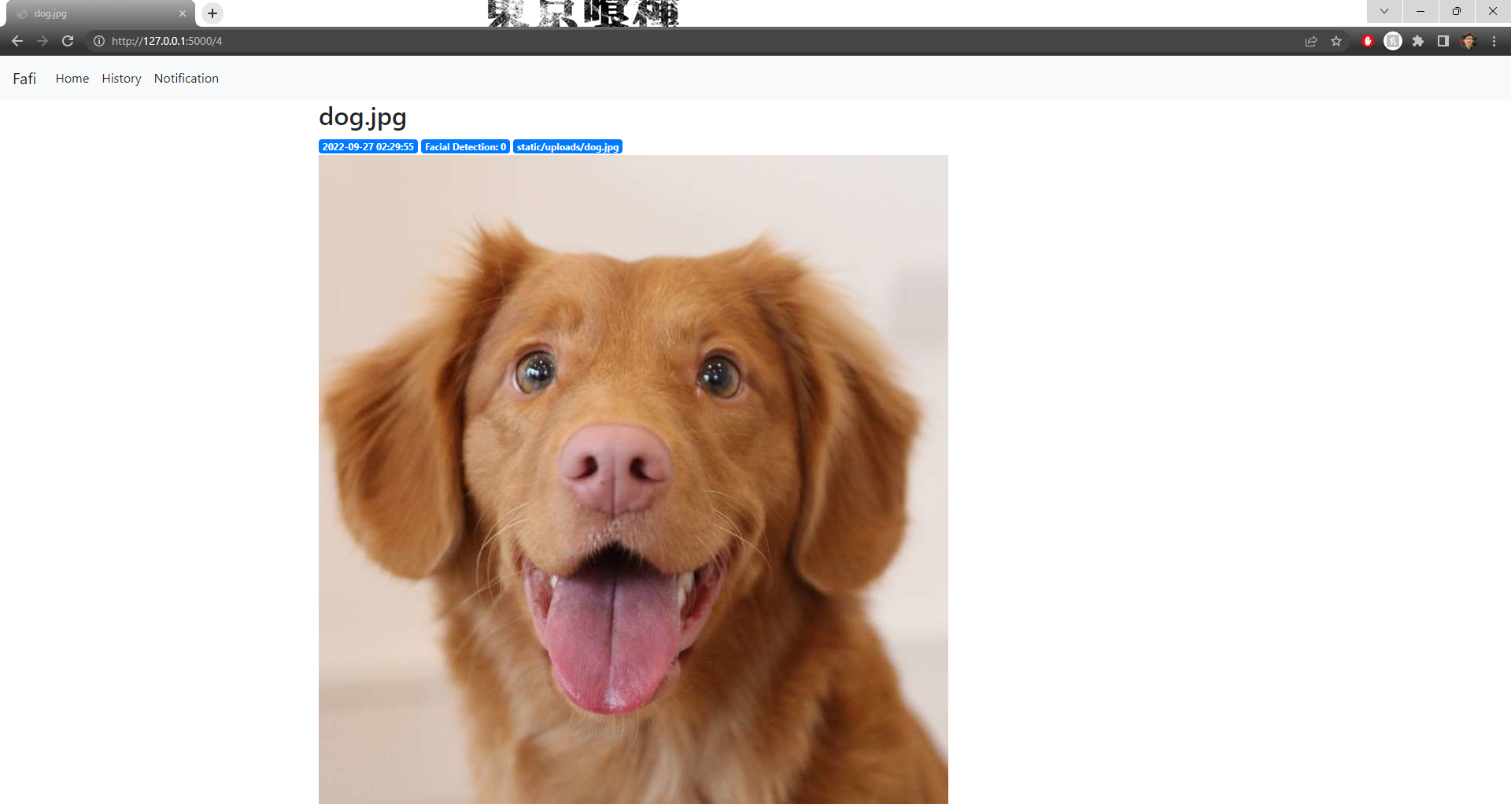
* + Unit Testing
    - Use PyTest to assert expected function outputs to ensure defined functions behave as expected.
  + Regression Testing
    - PyTest tests using GET and POST requests to the flask web application and checking response code to ensure introducing new logic does not break the web framework
    - PyTest tests using expected SQL queries against our DB to ensure there are no breaking changes
  + System and Application Testing
    - Testers will assume the role of a user and interact with the web application
      * Visit the different tabs
      * Pass in arbitrary and various inputs to each input field
      * Ensure the user experience fits the expectations laid out in the requirements
    - Test web application running on different browsers

# Manual Testing Report

* Test Case: Testing of initial Fafi Web User Application Template
* New or old: New
* Test items: Test local running Flask web application
* Test priority: High
* Dependencies: None
* Preconditions:
  + Requirements are installed locally
  + DB is initialized with `init\_db.py` script
* Input data: local image
* Test steps:
  + Set up preconditions
  + Run Flask app
  + Mock login through landing page
  + Upload file on home page
  + View uploaded file on history tab
    - Note file metadata is displayed correctly
  + View notification tab
* Postconditions: Uploaded file has relevant contextual information stored in database, and file is stored locally
* Expected output: Uploaded image and upload time is viewable in the history tab. Navigation between tabs work.
* Actual output: Uploaded image and upload time is viewable in the history tab. Navigation between tabs work.
* Pass or Fail: Pass
* Bug id/link: None
* Additional notes:

# 



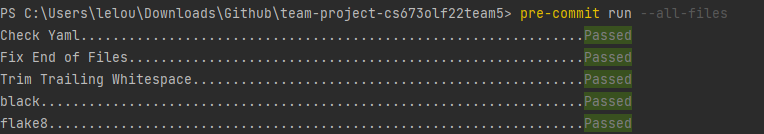


# Automated Testing Report

We currently have two automated testing reports:

1. **Pre-Commit**

Pre-commit makes use of Git hook scripts to identify any issues before a git commit. We use this to automate styling enforcement to PEP8 standards, using Flask8. A check against Flask8 must pass which checks if the code adheres to PEP8 styling guidelines. We also use it to lint for other styling checks such as against yaml, and end of file newlines.

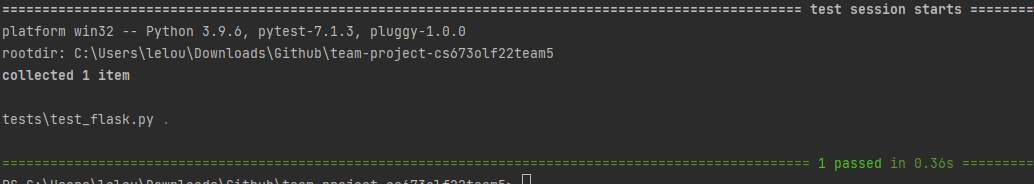


The configuration for this is on our repo: .pre-commit-config.yaml

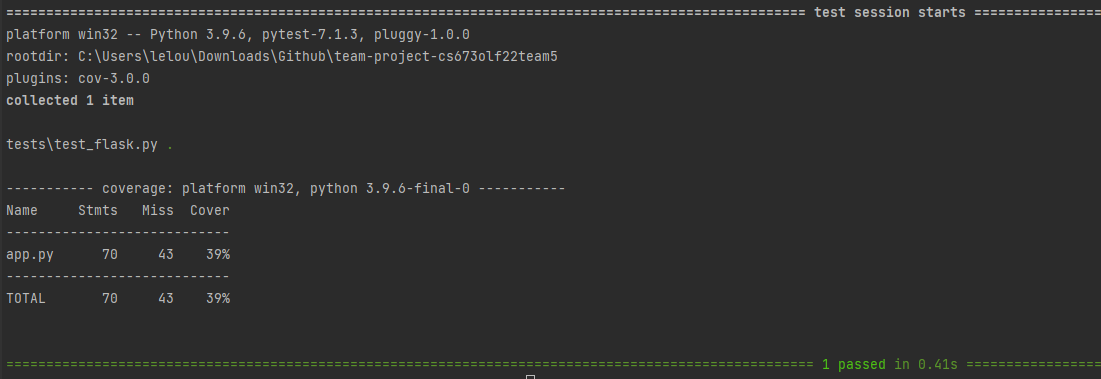
1. **PyTest**

The pytest framework allows us to easily write unit tests against our python functions. We can use it to assert expected behavior, and with this test function behavior and system/regression tests that check the web application responses.

These tests can be run automatically using the pytest command:



We are also utilizing the pytest-cov plugin to check for pytest code coverage, one of our testing metrics:



Our pytest framework is located in our testing folder: /tests

In the coming iterations, we hope to automate this into our CI/CD using Github Actions. We currently have a configuration for that drafted in our repo: ./github/workflow/cicd.yaml

# Testing Metrics

| Metric Name | Description |
| --- | --- |
| # of Test Cases | The amount of test cases. |
| Test Case Pass Rate | The percentage of test cases that pass. |
| Confidence Rate | How confident (%) the algorithm is that the presented image contains a cat. |
| Accuracy | 1 |
| Test Code Coverage | The percentage of our code that our test cases cover. Ideally we would have tests that check all our methods. |
| Cost | The amount of person hours invested by the team. |
| # of User Stories completed | The amount of user stories completed. |

# References

* PyTest: <https://docs.pytest.org/en/7.1.x/>
* PyTest Cov: <https://pytest-cov.readthedocs.io/en/latest/>
* Github Actions: <https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-python>
* Pre-Commit: <https://pre-commit.com/>
* Flake8: <https://flake8.pycqa.org/en/latest/>

# 

# Glossary

QA - Quality Assurance

UI - User Interface

CI/CD - Continuous Integration and Continuous Delivery