**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** |
| **2** | **Brendan Truong** | **10/12/22** | **Additional manual test cases** |

[Testing Summary](#_sm5odwyvuk3j)

[Manuel Tests Reports](#_pqso2mbjyzx4)

[Automated Testing Reports](#_mtfbusfb0eq3)

[Testing Metrics](#_rijyjeu2ojqa)

[References](#_15tmymhipvdv)

[Glossary](#_8n34lvocupub)

# Testing Summary

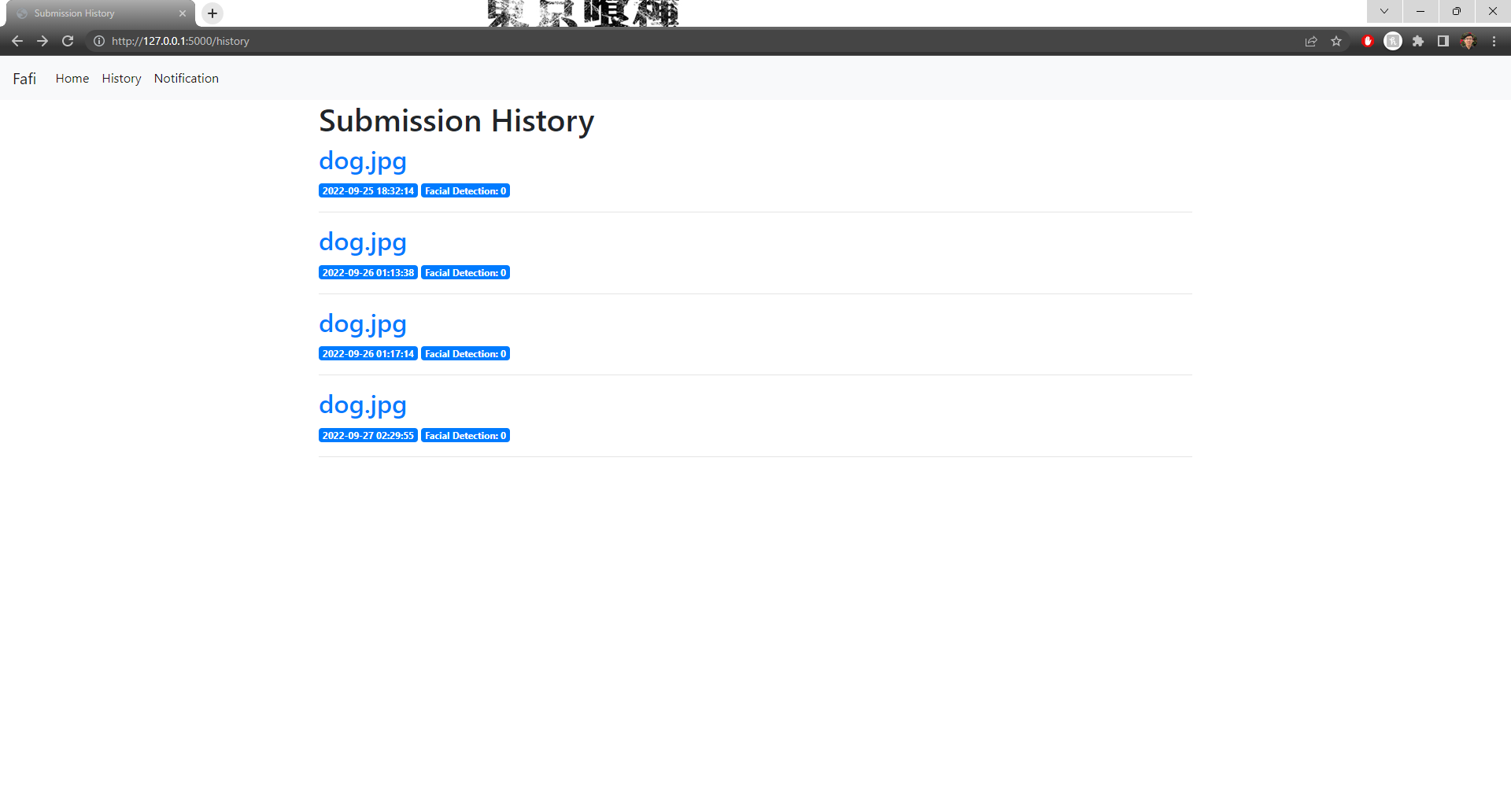
Tests will be written in PyTest. Each team member is expected to create unit tests for functionalities that they develop. The QA leader will provide special focus in integration testing of front end UI and the backend along with performance testing of the end to end application from a user experience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

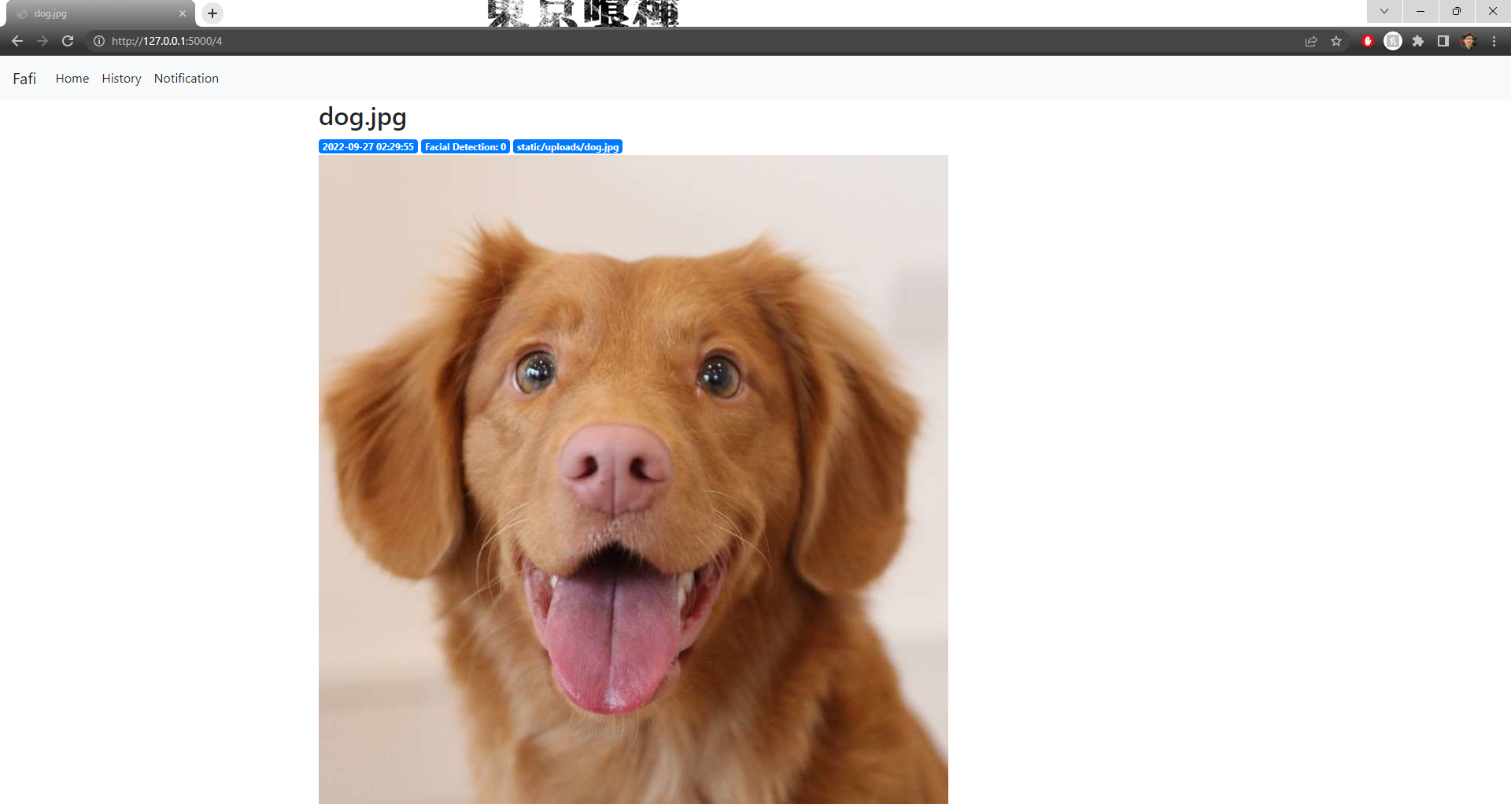
* + Unit Testing
    - Use PyTest to assert expected function outputs to ensure defined functions behave as expected.
  + Regression Testing
    - PyTest tests using GET and POST requests to the flask web application and checking response code to ensure introducing new logic does not break the web framework
    - 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

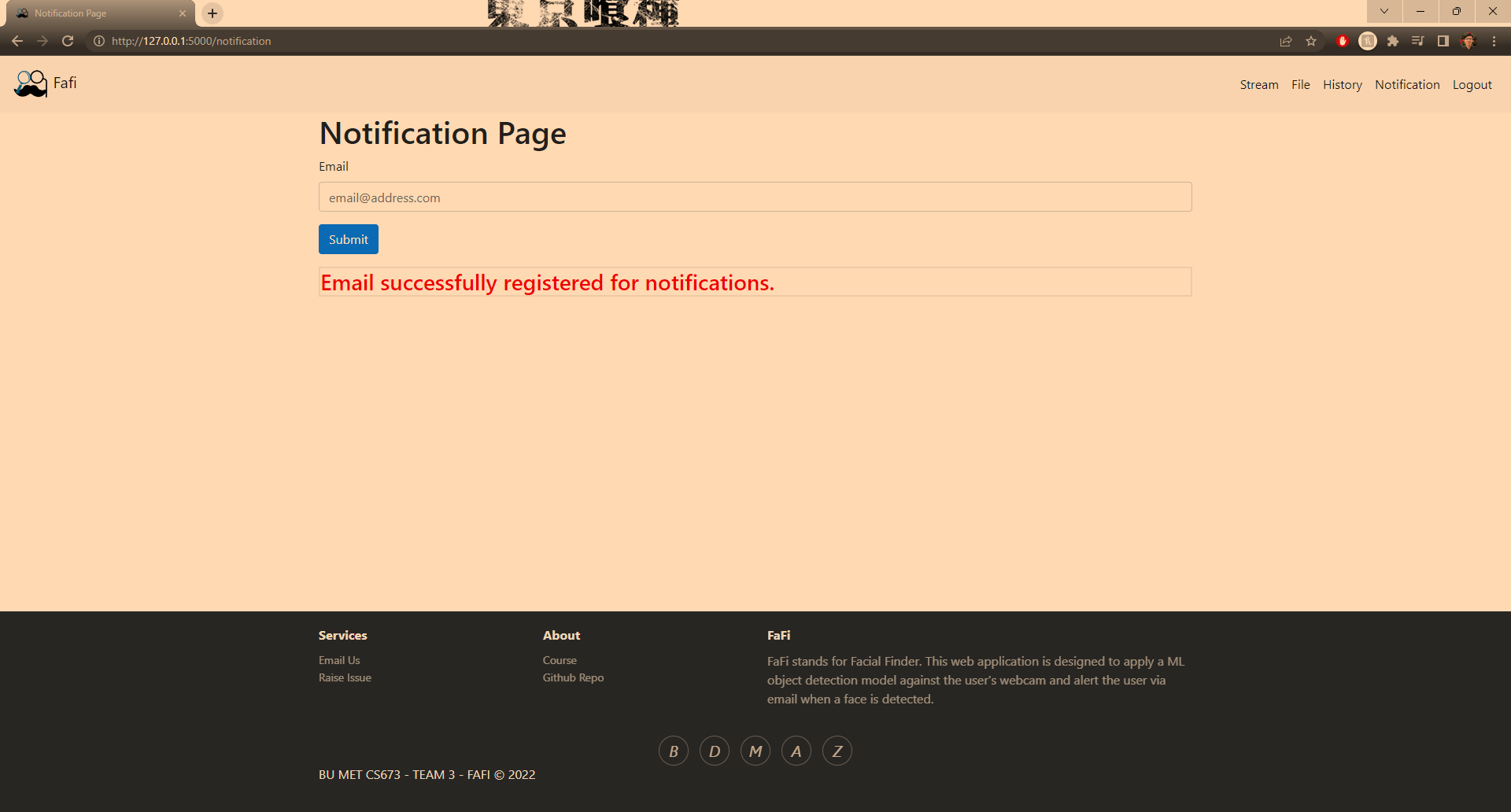
* Test Case: Testing of initial Fafi Web User Application Template
* New or old: New
* Test items: Test local running Flask web application
* Test priority: High
* Dependencies: None
* Preconditions:
  + Requirements are installed locally
  + DB is initialized with `init\_db.py` script
* Input data: 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:

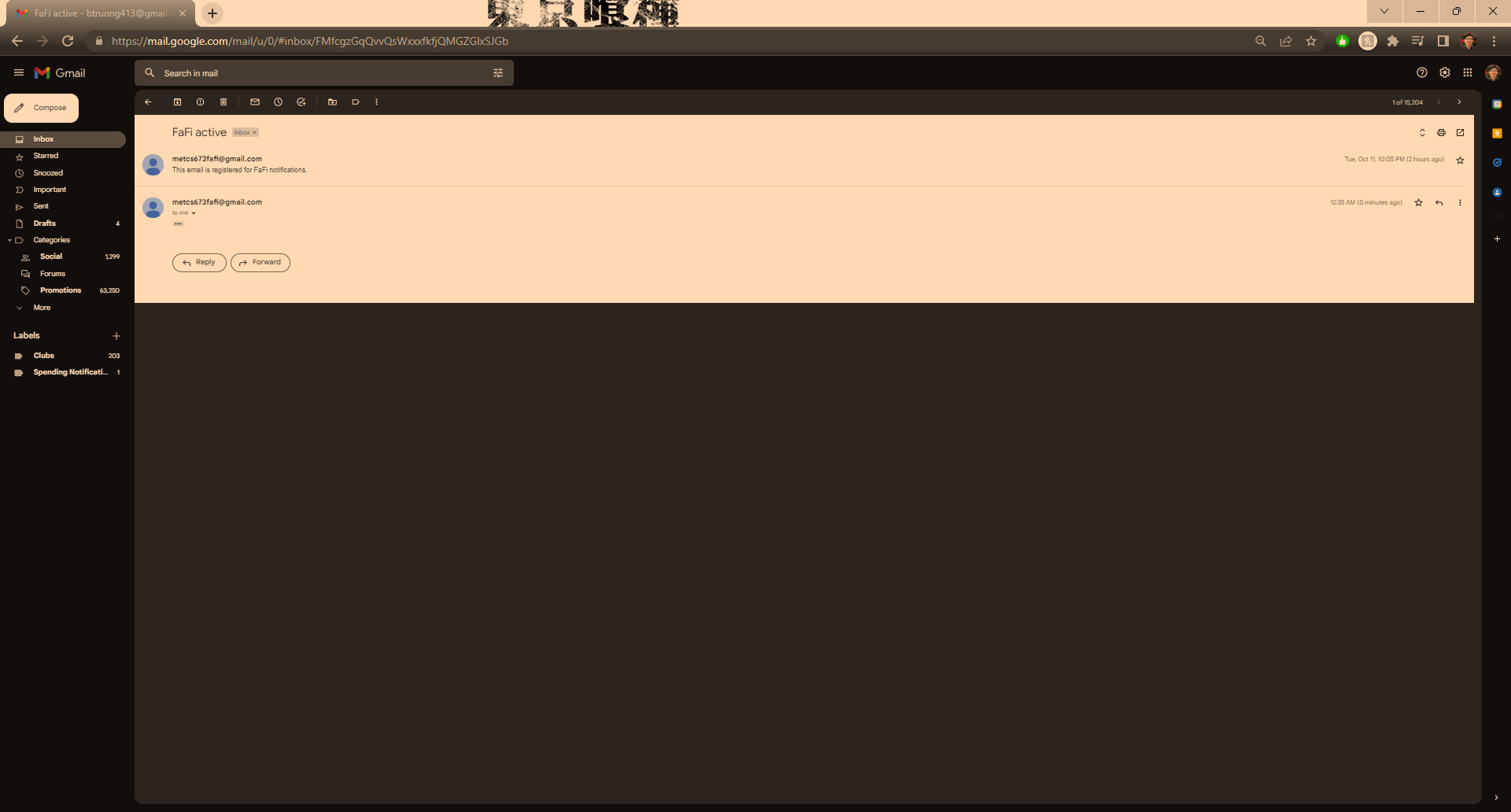
# 



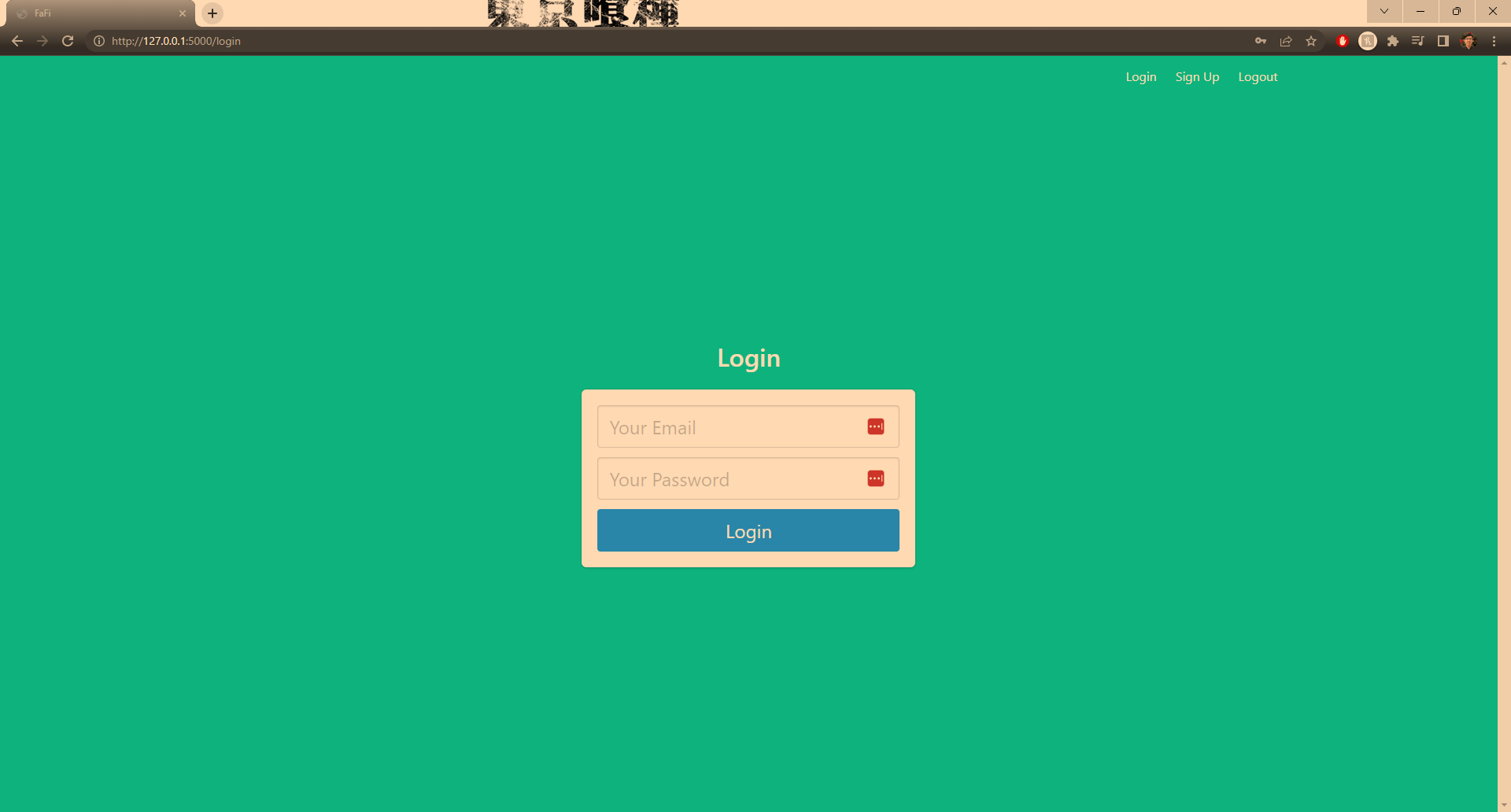


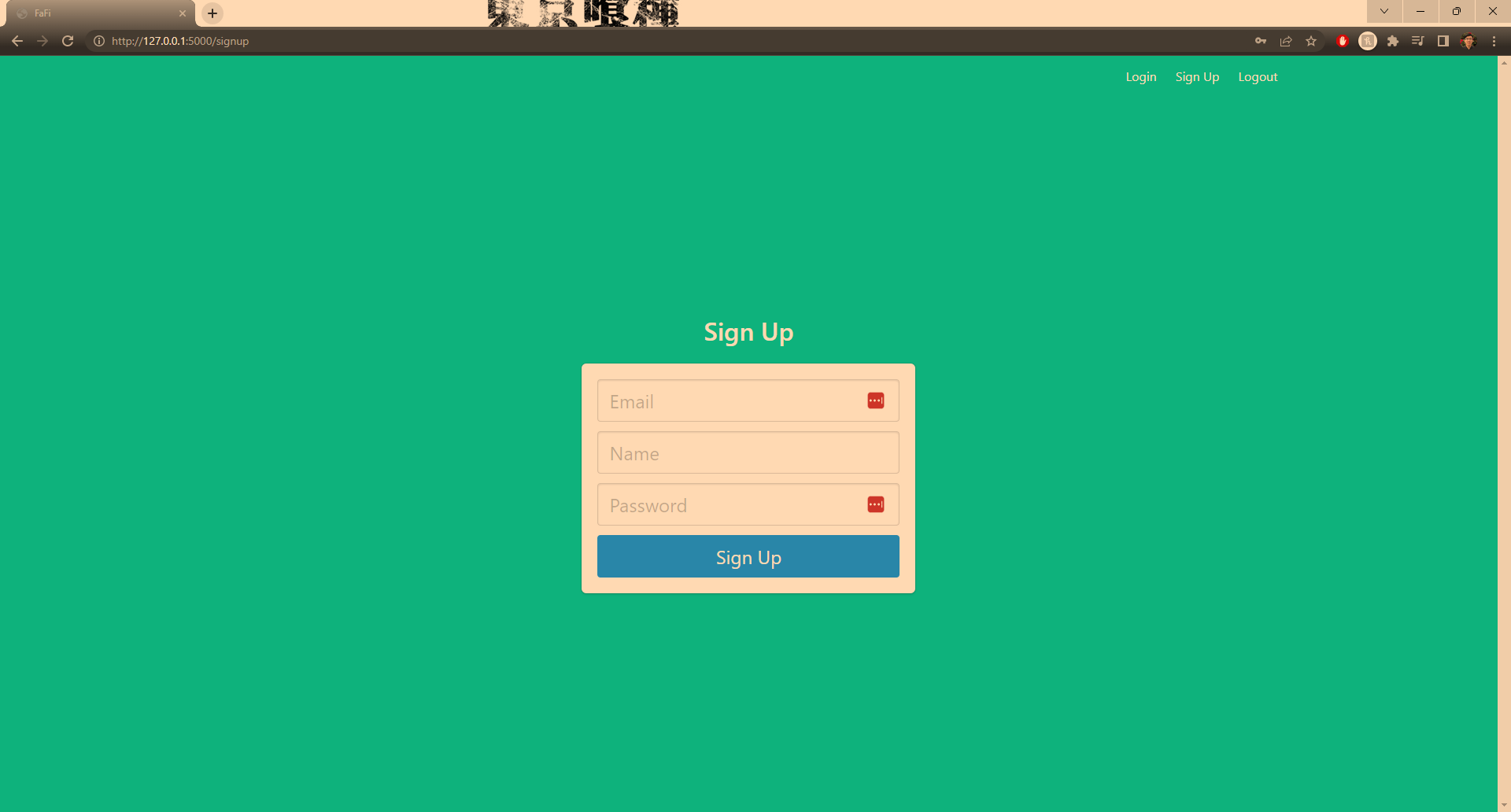
* Test Case: Testing of Notification alerts
* New or old: New
* Test items: Test notification tab implementation
* Test priority: Medium
* Dependencies: None
* Preconditions:
  + Requirements are installed locally
  + DB is initialized with `init\_db.py` script
* Input data: user email address
* Test steps:
  + Set up preconditions
  + Run Flask app
  + Mock login through landing page
  + Navigate to notification page
  + Enter email address
  + View UI response message
  + Check email for signup alert
* Postconditions: Email is sent to email address
* Expected output: An email is received at email address notifying the email is registered
* Actual output: Email is received.
* Pass or Fail: Pass
* Bug id/link: None
* Additional notes:This feature can allow further enhancements to integrate with user login email and provide other notification parameters.

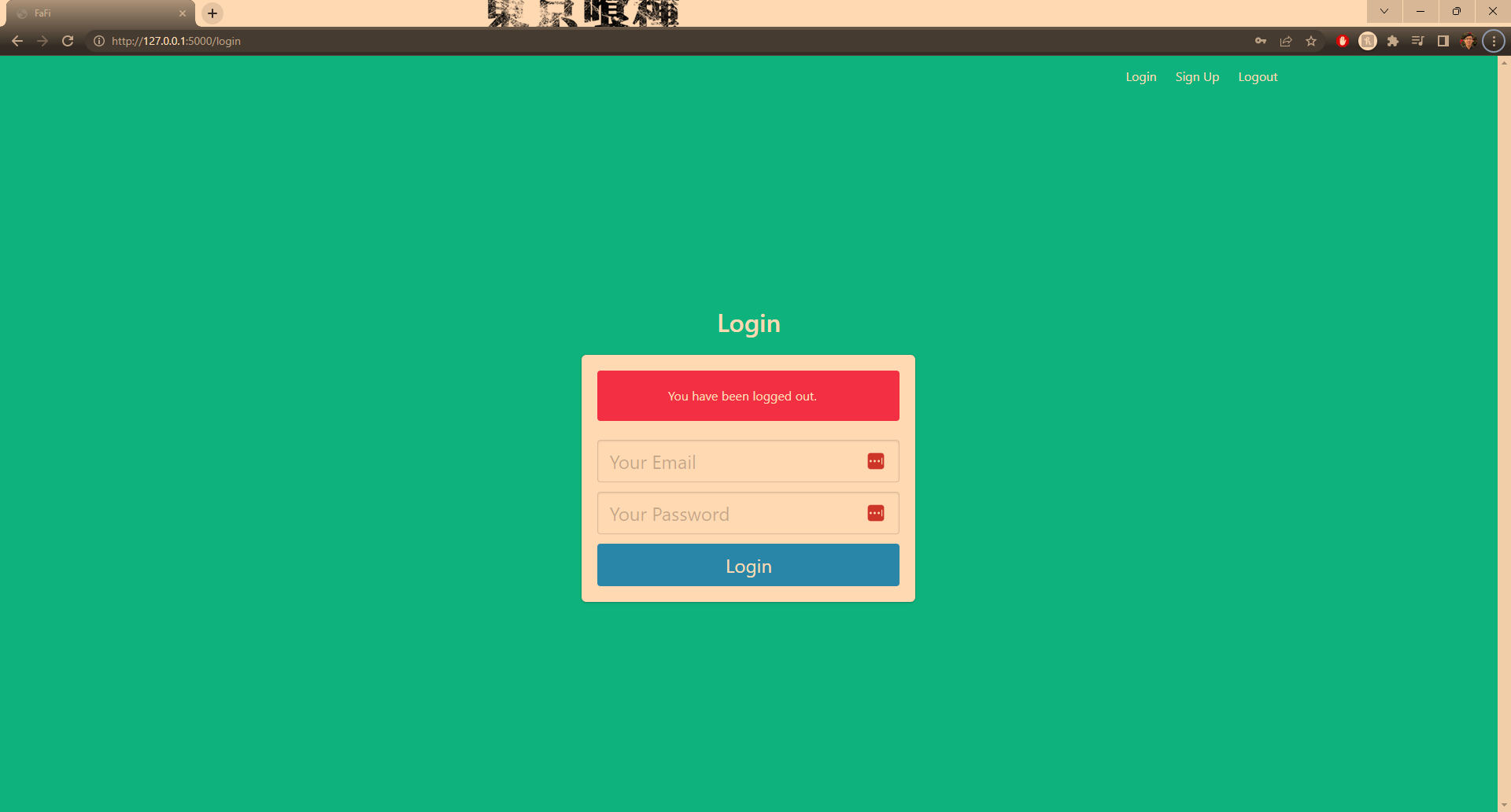


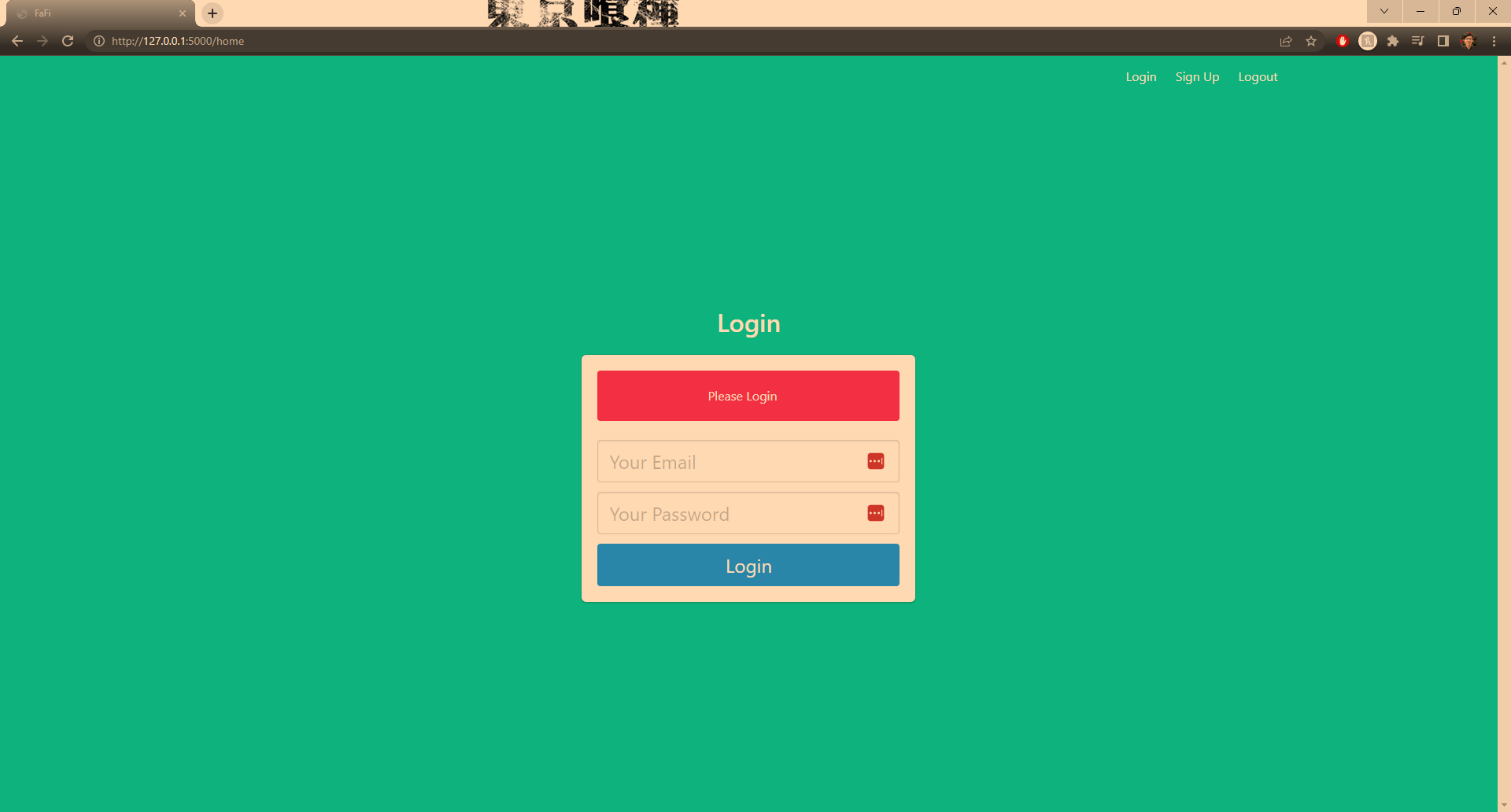


* Test Case: Login
* New or old: New
* Test items: Login page
* Test priority: High
* Dependencies: None
* Preconditions:
  + Requirements are installed locally
  + DB is initialized with `init\_db.py` script
* Input data: user email address, name, and password
* Test steps:
  + Set up preconditions
  + Run Flask app
  + User navigates to login page
  + User attempts to login without signing up
  + User navigates to sign up page
  + User enter correct values for form request (email, name, password)
  + User navigates to log in page and signs in with those credentials
  + User can access FaFI UI
  + User logs out and is redirected to login page
* Postconditions: User cannot access FaFI without logging in.
* Expected output: User account credentials stored within the database. User can log in with signed up credentials. User cannot access FaFi without logging in.
* Actual output: Same as expected output.
* Pass or Fail: Pass
* Bug id/link: None
* Additional notes: Login and sign up functionality works. Perhaps we can integrate email notification on user sign up to the email so they can verify their account is correlated to proper email. Or we can have a profile page to show this metadata.









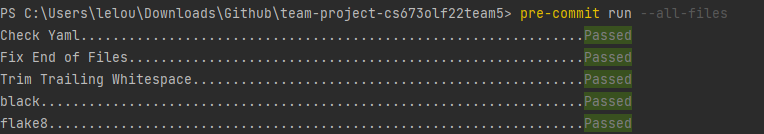
* Test Case: Footer hyperlinks
* New or old: New
* Test items: Footer
* Test priority: Low
* Dependencies: None
* Preconditions:
  + User successfully logs in to FaFI
* Input data: none, user clicks to hyperlinks in footer. If submitting an email, user must input email subject and message. If raising an issue, user must input issue details.
* Test steps:
  + Set up preconditions
  + Run Flask app
  + User navigates to login page
  + User signs up successfully
  + User logins in successfully
  + User clicks on Email Us in Footer
  + User clicks on Raise Issue in Footer
  + User clicks on Course in Footer
  + User clicks on Github Repo in footer
* Postconditions: None
* Expected output: User is redirected appropriately for each hyperlink. Email us redirects to desktop’s email application with FaFi email address. Raise issue redirects to project’s repo’s issues tab. Course redirects to course official website. Github Repo redirects to project’s repo.
* Actual output: Same as expected output.
* Pass or Fail: Pass
* Bug id/link: None
* Additional notes: None
* Test Case: integration testing
* New or old: old
* Test items: Entire application
* Test priority: High
* Dependencies: New features
* Preconditions:
  + Requirements are installed locally
  + DB is initialized with `init\_db.py` script
* Input data: Proper user input to application forms. User image file.
* Test steps:
  + Set up preconditions
  + Run Flask app
  + User navigates to login page
  + User signs up successfully
  + User logins in successfully
  + User interacts with footer
  + User interacts with stream tab
  + User interacts with file tab
  + User interacts with history tab
  + User interacts with notification tab
  + User logs out
* Postconditions: None
* Expected output: User is able to sign up and login properly User is able to navigate through footer hyperlinks. User is able to pass webcam stream in the stream tab and view facial detection. User is able to upload a file. User is able to view history of files. User is able to enter an email address for notification form.
* Actual output: All the above, except web application breaks on file submit.
* Pass or Fail: Fail
* Bug id/link: None
* Additional notes: This error was fixed in the following commit:  
  <https://github.com/BUMETCS673/team-project-cs673olf22team5/commit/3e12cd0be278b318143a30793e4ae48ec1ac7230>  
    
  Otherwise, no issues were faced and now the application is fully functional.
* Test Case: Final Integration Test
* New or old: New
* Test items: Entire Web Application, User walk through
* Test priority: High
* Dependencies: None
* Preconditions:
  + Requirements are installed locally
  + DB is initialized with `init\_db.py` script
* Input data: user email address, name, and password, web cam feed, local image file
* Test steps:
  + Set up preconditions
  + Run Flask app
  + User navigates to login page
  + User attempts to login without signing up
  + User navigates to sign up page
  + User enter correct values for form request (email, name, password)
  + User navigates to log in page and signs in with those credentials
  + User can access FaFI UI
  + User can visit stream page and stream runs
  + User is detected on stream
  + User is notified on the email they sign up on for stream detection
  + User can navigate to file page
  + User can submit a file and be alerted of detection results
  + User can visit history tab
  + User can see all their past submissions
  + User can click on a submission and view the image and time stamp
  + User can visit the settings tab
  + User can pass email and time interval parameter
  + New email is passed and new time interval parameter is passed
  + User navigates to stream
  + User waits and is alerted on their new email after the new time interval
  + User logs out and is redirected to login page
  + User is unable to access previous pages without logging in again.
* Postconditions: User is notified of all their detection submissions and alerts.
* Expected output: User is able to sign up and log in to the app. They can use their stream or pass a file for detection. The results are emailed to the user’s email or those defined in the settings. Email alerts are intervaled by the setting. Users are able to view a history of all their submissions and alert. User cannot access pages without signing in.
* Actual output: All the above functioned
* Pass or Fail: Pass
* Bug id/link: None
* Additional notes: A few slight bugs were noted such as the footer not sticking to the bottom of the page, user stream overlapped portionally by the footer. However this is a cosmetic issue and does not impair the app’s functionality

# 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.

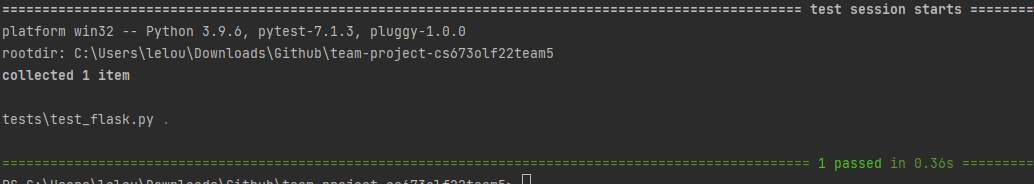


The configuration for this is on our repo: .pre-commit-config.yaml

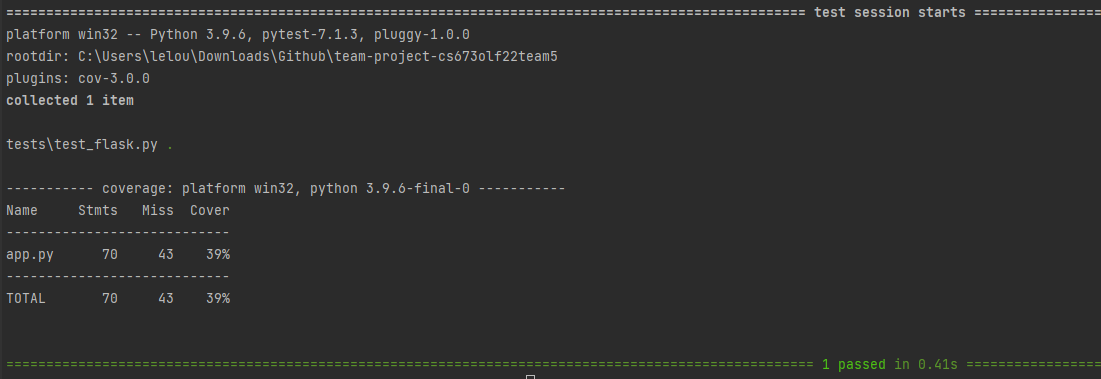
1. **PyTest**

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