Logging and HTML Reports

How to generate HTML reports of pytest

You can generate HTML reports for your Pytest tests using the `pytest-html` plugin. This plugin allows you to create an easy-to-read HTML report after running your tests.

Steps to Generate HTML Reports in Pytest:

1. Install `pytest-html` Plugin

First, you need to install the `pytest-html` plugin using `pip`:

```
pip3 install pytest-html
```

2. Run Pytest with HTML Report Option

After the plugin is installed, you can generate an HTML report by running the following command:

```
pytest --html=report.html
```

This will run your tests and generate an HTML report named `report.html` in the current directory.

3. Customize the HTML Report

You can add a title and some additional metadata (like environment info) to your HTML report using extra options:

```
pytest --html=report.html --self-contained-html --capture=tee-sys
\
    --tb=short --maxfail=5 \
    --metadata=Environment,Staging --metadata=Browser,Chrome
```

- `--html=report.html`: Specifies the output file for the HTML report.
- `--self-contained-html`: Embeds all the assets (CSS, JS) inside the report file.
- `--capture=tee-sys`: Ensures output is displayed to both stdout and captured in the report.
- `--metadata=Key,Value`: Adds extra metadata (e.g., environment, browser) to the report.

4. Viewing the Report

Once the tests are finished, open the `report.html` file in any browser to view the test results, including pass/fail status, stack traces, and logs for failed tests.

Example Command:

`pytest --html=results/test_report.html --self-contained-html`
This command will generate a detailed HTML report with all test results, including test execution times, errors, and more.

Additional Features:

- **Screenshots**: You can capture and include screenshots in the report if a test fails (useful in UI tests).
- **Logs**: You can also capture logs or output to be included in the report.

Introduction to Logging in Python Tests

Logging is an essential part of developing and debugging software, and this holds true for testing as well. Logging in Python provides a way to track events that happen during the execution of a program. When writing tests, logging can be used to capture relevant information about test execution, helping you debug issues and get insights into test behavior.

Why Use Logging in Tests?

- **Debugging**: Logs provide detailed insights into the flow of execution and the state of variables.
- **Error Tracking**: It becomes easier to track down the root cause of errors when failures happen.
- **Test Reports**: Logs are often included in test reports, providing context for test failures and helping developers understand what went wrong.

Python's `logging` Module

Python's `logging` module allows you to track events in your application, both during development and testing. Unlike print statements, logging can capture different levels of events (e.g., `DEBUG`, `INFO`, `WARNING`, `ERROR`, `CRITICAL`), which can be configured to show or hide certain information.

Basic Usage of `logging`

Here's how you can set up basic logging in Python:

```
import logging

# Configure basic logging
logging.basicConfig(level=logging.DEBUG, format='%(asctime)s - %
(levelname)s - %(message)s')

logging.debug("This is a debug message")
logging.info("This is an info message")
```

```
logging.warning("This is a warning message")
logging.error("This is an error message")
logging.critical("This is a critical message")
```

Using Logging in Tests

When writing tests, logging can provide real-time feedback about the flow and state of your tests. You can add logging to your test cases to capture detailed information about test execution.

Example: Logging in Unit Tests with `unittest`

```
import logging
import unittest

# Configure logging for the test module
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %
(levelname)s - %(message)s')

class TestLoggingExample(unittest.TestCase):

    def test_addition(self):
        logging.info("Starting test_addition")
        result = 1 + 1
        logging.debug(f"Result of addition: {result}")
        self.assertEqual(result, 2)

    def test_division(self):
        logging.info("Starting test_division")
        try:
            result = 10 / 0
        except ZeroDivisionError as e:
            logging.error(f"Error occurred: {e}")
            self.fail("Division by zero occurred")

if __name__ == '__main__':
    unittest.main()
```

Logging Levels

- **DEBUG**: Detailed information, useful for diagnosing problems.
- **INFO**: Confirmation that things are working as expected.
- **WARNING**: Indication of something unexpected or potential issues.
- **ERROR**: A more serious issue preventing a function from working.
- **CRITICAL**: A severe error that likely leads to program termination.

Logging in Pytest

If you're using Pytest, you can integrate logging into your tests as well. Pytest provides built-in support for logging and can capture logs during test execution.

Example: Logging in Pytest

```python
import logging
import pytest

# Configure logging
logging.basicConfig(level=logging.DEBUG, format='%(asctime)s - %
(levelname)s - %(message)s')

def test_logging_in_pytest():
    logging.info("Starting the test in pytest")
    result = 5 + 3
    logging.debug(f"Result is {result}")
    assert result == 8

def test_logging_failure():
    logging.info("Starting the failing test")
    result = 10 / 0  # This will raise ZeroDivisionError
```

Pytest Log Capture

Pytest automatically captures logs and displays them in case of test failures. To display logs even for passing tests, you can use the `--log-cli-level` option:

```
pytest --log-cli-level=INFO
```

Configuring Logging in `pytest.ini`

You can configure logging globally for your Pytest test suite by modifying the `pytest.ini` file:

```
# pytest.ini
[pytest]
log_cli = true
log_level = INFO
log_format = %(asctime)s %(levelname)s %(message)s
log_date_format = %Y-%m-%d %H:%M:%S
```

This ensures that logging is enabled and configured for all your tests without needing to modify individual test files.

Best Practices for Logging in Tests

- **Use appropriate logging levels**: Use `DEBUG` for development, and `INFO` or `ERROR` for reporting significant events or errors.
- **Capture important test steps**: Log key steps in your tests, like setup, execution, and teardown, to track the flow of the test.
- **Avoid excessive logging**: Be cautious with excessive logging, as it can clutter test output and make it difficult to focus on important information.
- **Include logs in reports**: If you're generating HTML or other forms of test reports, ensure the logs are included to help analyze test failures.

Conclusion

Logging is a powerful tool in Python tests that can help you monitor, debug, and understand test execution. By using Python's built-in `logging` module, you can track key events, capture error messages, and make your test suite more informative. Whether you're using `unittest`, `pytest`, or another test framework, integrating logging will greatly improve your ability to troubleshoot and maintain test code.

How to integrate pytest logs into html reporting

Integrating Pytest logs into HTML reports is a useful way to provide detailed test execution information, especially when you are generating reports for sharing with stakeholders or team members. You can achieve this by using the `pytest-html` plugin along with Pytest's built-in logging support. Here's how to do it.

Steps to Integrate Pytest Logs into HTML Reporting:

1. Install Required Plugins

You need both the `pytest-html` and `pytest-logging` (or simply use the built-in logging configuration) plugins to generate the HTML report and capture logs.

Install the necessary dependencies:

```
pip install pytest-html pytest
```

2. Set Up Logging Configuration

You can configure logging in your tests so that the logs are captured and displayed in the HTML report.

Example setup for basic logging configuration in your `pytest.ini`:

```
# pytest.ini
[pytest]
log_cli = true
log_cli_level = INFO
log_cli_format = %(asctime)s %(levelname)s %(message)s
log_cli_date_format = %Y-%m-%d %H:%M:%S
log_file = pytest_log.log
log_file_level = DEBUG
```

This will ensure that logging happens for all tests at the specified logging level, and it will capture both `INFO` and `DEBUG` level messages.

3. Configure the HTML Report

You can generate the HTML report using the `pytest-html` plugin. Use the `--html` option when running tests to specify the location of the generated report.

To include logs in the HTML report, you also need to enable the `--self-contained-html` option:

```
pytest --html=report.html --self-contained-html --log-cli-level=INFO
```
This command will:

- Generate an HTML report at `report.html`.
- Include logs in the report.
- Ensure that all assets (CSS, JS) are embedded in the HTML report, so it is self-contained and easy to share.

4. **Add Logs to Your Test Cases**

You can add logging statements to your test cases so that the relevant logs are captured during test execution and included in the HTML report.

Here's a simple example:

```
import logging
import pytest

# Configure logging
logging.basicConfig(level=logging.DEBUG, format='%(asctime)s - %(levelname)s - %(message)s')

def test_example_pass():
    logging.info("This is a passing test")
    assert 1 == 1

def test_example_fail():
    logging.info("This is a failing test")
    assert 1 == 2
```
When you run this using the previous command (`pytest --html=report.html --self-contained-html --log-cli-level=INFO`), both passing and failing tests will include the logs in the HTML report.

5. **Customize Report Title and Other Options**

You can also customize the report's title and additional metadata:

```
pytest --html=report.html --self-contained-html --log-cli-level=INFO \
       --metadata=Environment,Staging --metadata=Browser,Chrome
```
This will add the metadata to the report for better context.

6. **View Logs in the HTML Report**

Once the tests are executed, the `report.html` file will be created in the specified location. Open this file in a web browser to view the following:

- **Test results**: Shows which tests passed or failed.
- **Logs**: Detailed logs for each test, available as a collapsible section next to each test.

Example Pytest Command:

```
pytest --html=report.html --self-contained-html --log-cli-level=INFO
```

Example Log Output in HTML Report:

In the HTML report, under each test case, you will see logs like:

```
2024-09-24 12:30:01 - INFO - This is a passing test
2024-09-24 12:30:02 - INFO - This is a failing test
```

Conclusion

By following these steps, you can easily integrate Pytest logs into the generated HTML reports. The `pytest-html` plugin allows you to include the logs in a well-structured, easy-to-read format, making it simple to share detailed test execution information.
