Testing frameworks & strategies

Testing Patterns:

Each test should follow the following pattern:

1. Set up the test data / test inputs
2. Call your method under test
3. Assert that the expected results are returned

Tests need to be quick, covered, and repeatable. You should be running your tests often, testing all (non-trivial) parts of the code, and getting the same results each time you run.

Test one condition per test. This helps you to keep your tests short and easy to troubleshoot when one is failing.

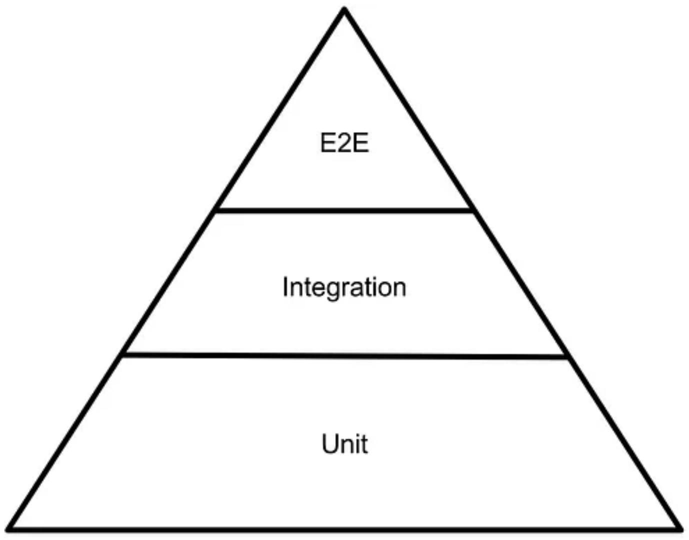
Use meaningful test names, each name should describe what the test does.

Write tests during development, not after the fact.

Testing Levels:

The idea of a testing pyramid was developed by Mike Cohn to get you to think about different layers of testing, where each layer increases complexity and interaction but decreases the total number of tests. It is based on the following principles:

1. Write tests with different granularity
2. The more high-level you get the fewer tests you should have



As with production code you should strive for simplicity and avoid duplication. In the context of implementing your test pyramid you should keep two rules of thumb in mind:

1. If a higher-level test spots an error and there's no lower-level test failing, you need to write a lower-level test
2. Push your tests as far down the test pyramid as you can

Unit tests are written to verify the functionality of individual units of code, such as functions or methods. Unit tests are usually automated and run quickly, and they should test all possible cases and edge cases. These are frequently written by the developer of that unit of code (white box), and are typically written using libraries like 'unittest' or 'pytest'.

Best Practices:
1. Write tests for edge cases and invalid inputs.
2. Use mocks for external dependencies to ensure isolation

Testing Tools:
Python: **unittest,** pytest
JavaScript: Jest, **Mocha**
Database: *Not always done at unit level*

Mocking Tools:
Python: **unittest.mock,** pytest-mock
JavaScript: **Mocha**

Issues: Jest needs Node.js to be 18.14.0 or later, but https://coding.csel.io/ environment is at 18.12.1.

Rationale: uniittest is already installed in our coding environment, and we will have some amount of experience from performing Lab-5. It works with Django and Flask as well as basic Python. Jest is the most common unit test framework for Javascript known for its ease of use, minimal configuration setup, and strong integration with React; however Jest does not work with the versions in our coding environment. Mocha is a test framework running both in Node.js and in the browser. This framework makes asynchronous testing simple by running tests serially, and has been tested to work in our coding environment.

Example Unit Tests to Be Written:
1. Frontend
    1. Display a user profile with mocked data
2. Backend
    1. Check if a new password matches the password requirements
3. Database
    1. Entries can be inserted and deleted from a table

Note: At the unit testing level, generally the writing of test is a subtask of the existing development story and is not a separate story in itself. No stories written for this level of testing.

Integration tests are written to verify the interaction between different units of code. Integration tests are usually automated, but they may take longer to run than unit tests. Integration tests should cover all possible combinations of code units (any which interact) and ensure that they work correctly together. These may be written by the writer of the underlying code (white box) but is often completed by an independent developer (black box). They can be written using the same libraries as unit testing or by using libraries tailored to specific functions.

Best Practices:
1. Mock external *services* for predictable, repeatable results.

Testing Tools:
unittest or pytest: Testing larger flows where code units work together.
Postman: Automate API tests.
WireMock: Mock HTTP services.
TestContainers: Run real dependencies in isolated containers.
REST Assured: Automate API tests.

Rationale: REST Assured is fairly popular with plenty of tutorials available, is free and open source, and supports any HTTP method but has explicit support
for POST, GET, PUT, DELETE, OPTIONS, PATCH and HEAD and includes specifying and validating e.g. parameters, headers, cookies and body easily.

Example Integration Tests to Be Written:
1. Calls to your services' REST API
2. Reading from and writing to databases
3. Calling other application's APIs
4. Reading from and writing to queues
5. Writing to the filesystem

End to End tests are written to verify the entire system, from the user interface to the back-end code. End-to-end tests are usually manual and take the longest time to run, but they are essential for ensuring that the system works correctly from the user's perspective. These are written by an independent developer (black box). Tools like Selenium or WebDriver Protocol are commonly used for this level of testing if automated, for example in a continuous delivery workflow.

Best Practices:
1. Limit E2E tests to essential workflows.
2. Use synthetic test data and isolate tests from production systems.

Testing Tools:
Selenium: Automate browser interactions.
Cypress: Developer-friendly E2E testing for web apps.

Rationale: While E2E testing eliminates repetitive manual testing that consumes time and effort, our project duration and long-term forecast (<3 months, no development release schedule) does not support building this level of testing.

Selenium is fairly popular with plenty of tutorials available, is free and open source, and the test scripts can be written in several languages (including Java and Python).

Example E2E Tests to Be Written:
1. User can log into web application and get their unread messages

Tech Stack and Setup Instructions:

Assumes the coding.csel.io Environment:

Base Code:
1. Python 3.10.8
2. (Java) openjdk 11.0.17 2022-10-18
3. (Node.Js) v18.12.1

Test Code:
1. unittest, Mocha
2. REST Assured
3. *Selenium (Optional)*

Python and Java are already installed in the https://coding.csel.io/ environment.
unittest is already installed in the https://coding.csel.io/ environment.
Mocha is not installed by default. See this tutorial to getting started with Mocha:
    https://mochajs.org/#getting-started
REST Assured is not installed by default. See this tutorial to getting started with REST Assured:
    <TBD, Story to Follow Up>
Selenium is not installed by default. See this tutorial to getting started with Selenium:
    <TBD, Story to Follow Up>

Stories to Make:

Run Mocha setup tutorial
    As a developer,
    I want to, setup Mocha
    So that, all developers can write unit level tests for code written in Javascript in the coding.csel.io environment
Run REST Assured setup tutorial
    As a developer,
    I want to, setup REST Assured
    So that, all developers can write integration level tests for code APIs in the coding.csel.io environment
Run Selenium setup tutorial (optional)
    As a developer,
    I want to, setup Selenium
    So that, all developers can write E2E level tests for code in the coding.csel.io environment
Create Test Code Standard Template
    As a developer,
    I want to, create Test-Code templates
    So that, developers have an standard upon which to base tests
        Subtasks:
            Unit Level: unnittest
            Unit Level: Mocha
            Integration Level: REST Assured
            E2E Level: Selenium (optional)

Note: Additional Stories should be created to test specific code functionality at the integration or End-to-End levels; dependent on User Stories developed for FrontEnd / BackEnd / Database

References:

Test Pyramid:

https://martinfowler.com/articles/practical-test-pyramid.html

https://dev.to/craftedwithintent/understanding-the-testing-pyramid-and-testing-trophy-tools-strategies-and-challenges-k1j

Unittest**:**

https://realpython.com/python-testing/

Jest:

https://medium.com/@eva.matova6/unit-testing-in-react-a-practical-guide-for-frontend-developers-b3362c7151d7

https://www.browserstack.com/guide/jest-framework-tutorial

Mocha:

https://mochajs.org/

REST Assured:

https://github.com/rest-assured/rest-assured/wiki/GettingStarted

https://medium.com/@theautobot/rest-api-and-rest-assured-tutorial-understanding-and-implementing-restful-web-services-33fd5659a09a

Selenium:

https://www.browserstack.com/guide/python-selenium-to-run-web-automation-test