Coding Standards for AuthFlow

Table of Contents

1. Introduction ……2
    1.1 Naming Conventions ……..2
    1.2 Style and Spacing ……..……2

---

2. Implementation and Solutions ……3-4
    2.1 Function Standards ……3
    2.2 Use of Third Party Libraries ..……3-4

---

3. Code Review Process ……..4-5
    3.1 Making Pull Requests ……..……...4
    3.2 Reviewing Pull Requests ..……4-5

---

4. Testing ……..……...5-6
    4.1 Technology for Testing……..……..……...5
    4.2 Testing Requirements ..……..……6

Introduction - Cael

1.1 - Naming Conventions

- Camel Case for functions and variables
- No functions or variables with a length greater than twenty four
- Functions and variables with similar names should be avoided
    - Names for functions and variables should be descriptive
    - Ex: If a function is supposed to change a user's password, it could be:
        - changeUserPass(string oldPass, string newPass)
- Function and variable names should not be too abbreviated
    - Ex: If a function is supposed to send data to a server, it should not be named as such
        - sDS(string data)

1.2 - Style and Spacing

- Development done in Visual Studio Code should have two space tab standard indentation
- Each function and variable should have a comment explaining their purpose when they are first declared
    - This way someone reviewing the code can find the function/ variable declaration and immediately see its description
- Spacing between lines of code when necessary
    - Ex: between two function declarations, one line should be blank
- Excessive spacing should be avoided
    - Ex: do not need three blank lines between code

Implementation and Solutions - Brynn

2.1 - Function Standards

- Function names and variable names should follow the CamelCase convention.
- The length of functions and variables should not exceed twenty-four characters.
- Functions and variables with similar names should be avoided to prevent confusion.
- Descriptive names should be used for functions and variables to enhance readability and understanding.
- Avoid excessive abbreviation in function and variable names.
    - For example, use **`changeUserPassword`** instead of **`chngUsrPwd`**.
- Functions should have clear and descriptive parameters.
    - For example, **`changeUserPassword(string oldPassword, string newPassword)`**.
- Functions should have a clear and singular purpose, adhering to the Single Responsibility Principle.

2.2 - Use of Third Party Libraries

- Third-party libraries should be used judiciously, preferring built-in or native solutions when available.
- When using third-party libraries, ensure they adhere to the following standards:
    - Follow CamelCase naming convention for functions and variables.
    - Limit the length of functions and variables to twenty-four characters.

    - Avoid naming conflicts with existing functions or variables.
    - Use descriptive names for functions and variables.
    - Avoid excessive abbreviation in function and variable names.
- Evaluate the reputation, maintenance, and support of third-party libraries before integration into the project.
- Regularly update third-party libraries to benefit from bug fixes, performance improvements, and new features.
- Document the usage of third-party libraries, including installation instructions, version compatibility, and any potential issues or workarounds.

Code Review Process - Connor

3.1 Making Pull Requests

- Clear and descriptive title for the pull request summarizing changes. You should include the reason behind changes.
    - Ex: Title: Refactor authentication module to use JWT for improved security
- Comment should explain the purpose of functions and variables
- Adherence to all other coding standards including but not limited to: naming conventions and styling.
- All necessary unit tests should be updated for changes being made
- Assign pull request to appropriate reviewers

3.2 Reviewing Pull Requests

- Pull requests should be examined focusing on logic, functionality, and adherence to coding standards
- Code should be clear, readable and easy to maintain

- Focus on finding potential bugs and edge cases that have been overlooked
- Pull requests should not be approved until all feedback has been addressed
- Authors should receive criticisms openly and be receptive to feedback, focusing on understanding the concerns raised by reviewers

Testing - Liana

4.1 Technology for Testing

- Software specific to chosen programming language can be used for unit testing
    - CUnit for C
    - JUnit for Java, Mockito for mock testing
    - Jasmine and Mocha for Javascript
- Assertions and assumptions can be made manually in separate files as an alternative to using unit testing software
    - ex) tests.java that contains blocks of isolated sections of code
    - assertion and assumption examples
        - assertEquals(expected, actual)
        - assertTrue(condition)
        - assumeNotNull(Object…objects)
        - assumeTrue(boolean condition)
- Gradle will be used to automate testing and manage any dependencies and plugins
- Higher level integration tests for Java and Javascript can be handled with Cucumber
    - Only use if necessary, otherwise standard unit testing should be enough

4.2 Testing Requirements

- Test contract should be defined that tests the requirements of the program using test cases, unit tests, and integration tests
- Test cases should be a table with the following sections:
    - **Description** of the test case
        - ex) “User should be able to log in”
    - **Assumptions and Preconditions**, the conditions to be met before the case can be tested
        - ex) “Log in feature is implemented, backend function to recognize username and password is implemented”
    - **Test Data**, the variables and values needed for the case
        - ex) “Username, password”
    - **Test Steps**, the steps to be taken for the case to be tested, from the user’s end
        - ex) “Enter username, enter password, click ‘Log In’ button”
    - **Expected Result**, the result expected after test case is executed
        - ex) “User is informed that they have successfully logged in, taken to new logged in screen”
    - **Status**, whether the test passed or failed
- Cases should test:
    - Functional requirements; features of the application
    - Non-functional requirements; performance, service, scaling
    - Security requirements; security of the system