AlsoEnergy QACodingChallenge Test Plan

Prepared by: Zachary Smith

Table of Contents

Introduction **2**

Scope 2

In Scope 2

Out of Scope 3

Quality Objective 3

Roles and Responsibilities 3

Test Methodology **3**

Overview 3

Unit testing 4

Unit test design guidelines 4

Conducting unit tests 4

List of unit tests 4

Functional testing 4

Manual functional tests 4

Functional test script design guidelines 4

Executing the functional test script 5

List of functional tests 5

Acceptance testing 5

Bug Triage 5

Test Completeness 5

Resource & Environment Needs **5**

Testing Tools 5

Test Environment 6

Test Deliverables **6**

1 Introduction

This test plan is intended to describe the testing methodology used while developing, code reviewing, and testing an energy alert system as described in the coding challenge.

This plan identifies the items to be tested, the scope of the tests, the features to be tested, the types of testing, the personnel involved in testing, the resources and schedule required to complete testing, and the risks associated with the test plan.

Due to the nature of this coding challenge, and lacking knowledge of source code or UI implementation, a full list of tests cannot be compiled. A list of tests which should be included at a minimum has been included.

1.1 Scope

1.1.1 In Scope

The features as described in the coding challenge need to be tested. These include the alert lifecycle, the notification system, and setting configuration parameters for the notification system.

Feature	**Description**
Generating an alert	An alert should be triggered when a monitored system produces no energy.
Sending an alert email	When an alert is triggered, a user should be notified via email.
Sending a daily alert email	The alert email should repeat on a daily basis until acknowledged.
The “Acknowledge an alert” action	The action within the monitoring system that changes the alert state from “Open” to “Acknowledged”.
UI components relating to alert acknowledgement	The UI components that a user will use to acknowledge an alert.
The “Close an alert” action	The action within the monitoring system that changes the alert state from “Open” or “Acknowledged” to “Closed”.
Sending an “alert closed” email	When an alert status is set to “closed”, the user should be notified via email.
Expecting zero generation between “sunset” and “sunrise”	The system should not send alerts for zero generation between the times specified by “sunset” and “sunrise”.
The “Set the Sunset” action	The action within the monitoring system which sets the “sunset” time.
The “Set the Sunrise” action	The action within the monitoring system which sets the “sunrise” time.
UI components related to setting the “Sunset” and “Sunrise”	The UI components that a user will use to set the sunset and sunrise times.

Reading the non-alert time period	The system should return the “sunset” and “sunrise” times when queried.
UI components used to display the “sunset” and “sunrise”	The UI components that display the times when alerts should not be generated.

1.1.2 Out of Scope

The following are not in the scope of testing:

- The energy generating system
- Hardware Interfaces

1.2 Quality Objective

Objectives of the testing:

- Verify that the alert system conforms to functional specifications
- Verify that the alert system conforms to quality specifications
- Identify and report any bugs or issues discovered during testing

1.3 Roles and Responsibilities

Personnel involved in testing and their roles:

Title	**Role**
QA Engineer	Write test plan, specification, and design tests
Software Developer	Develop the alert system and run unit testing
DevOps Engineer	Integrate automated testing into CI/CD pipeline

2 Test Methodology

2.1 Overview

Due to the specificity of the test requirements (limiting to the notification system and alert lifecycle), only three levels of testing shall be used in this plan:

- Unit testing
- Functional testing
- Acceptance testing

2.2 Unit testing

2.2.1 Unit test design guidelines

- Tests should be written in a test framework appropriate for the language used for development. EG: JUnit if Java is used or pytest if Python is used
- Tests should be written to address the features described in section 1.1.1 “In Scope”.
- Tests should be specific and narrow to address functions within individual components of the application.
- Tests should not stretch beyond the class or component for which they are written
- Tests should make heavy use of mocking
- Tests should not rely on production hardware or servers
- Tests should use in-place databases and servers if possible EG: greenmail, HSQL
- Tests should be written to test all parts of the code, including for expected exceptions
- Tests should be written prior to any new development, and updated to match expected output before making changes to existing code.

2.2.2 Conducting unit tests

- Unit tests should be performed often by the software developer during development to ensure that program function is as expected.
- The DevOps Engineer should take advantage of CI/CD tools such as Jenkins to automate testing against all branches committed to the code repository.

2.2.3 List of unit tests

This list of tests is incomplete due to not having a full list of functions and UI components, but should include at a minimum:

- A test for the getters and setters of the “sunrise” and “sunset” times.
- A test for verifying that an alert is generated when a zero generation signal is detected.
- A test verifying that an alert is not generated between the “sunset” and “sunrise” times.
- A test verifying that an alert status is changed from “Open” to “Acknowledged”.
- A test verifying that an alert status is changed from “Open” or “Acknowledged” to closed.
- A test verifying that an email is sent when an alert is generated.
- A test verifying that a daily email is sent for alerts with a status of “Open”.

2.3 Functional testing

2.3.1 Manual functional tests

- The developer should perform manual functional testing against the application to ensure that their changes perform as expected.
- Other developers should perform manual functional testing during code review to ensure that the changes required by the change request perform as expected.

2.3.2 Functional test script design guidelines

- Tests should use a UI testing framework such as Selenium, Cucumber, or Jubula.
- Tests should be written to be modular and reusable such that smaller UI manipulation can be used to build up larger, more complicated tests.
- Tests should take advantage of a test server which can be signaled with a utility function in order to mock events necessary to test interaction with the notification system.

- Tests for particular features or bugs should be integrated into a suite of functional tests used for regression testing.

2.3.3 Executing the functional test script

- The DevOps Engineer should take advantage of CI/CD tools such as Jenkins to automate functional testing if possible.
- If it is not possible to automate running the functional test script, then it should be run by another software developer, or the QA Engineer as part of code review.
- Functional test results should be reviewed prior to accepting a bug/feature branch into a build branch in the code repository.

2.3.4 List of functional tests

This list of tests is incomplete due to not having a full list of UI components or implementation, but should include at a minimum:

- A test which sets the “sunset” time.
- A test which sets the “sunrise” time.
- A test which verifies the “sunset” and “sunrise” times have been changed successfully.
- A test which acknowledges an alert.

2.4 Acceptance testing

Acceptance testing is manual testing performed by the stakeholder or initial reporter of the change request related to the feature or bug. This testing is done to ensure that the requested feature has been resolved to the end user’s satisfaction. This testing is done on a release candidate of the application, during a final testing phase prior to deployment.

2.5 Bug Triage

Errors and issues are expected as part of testing. These should be analyzed and triaged as they are encountered.

- If an issue discovered during testing is related to the bug or feature currently being worked, then it should be resolved prior to passing code review.
- If an issue discovered during testing is unrelated to the bug or feature currently being worked, then it should be documented, analyzed, and assigned severity and priority.

2.6 Test Completeness

Testing shall be considered complete when the following have been accomplished:

- 100% test coverage
- All Unit & Functional Test cases executed

3 Resource & Environment Needs

3.1 Testing Tools

- Unit testing framework (JUnit, pytest)
- Functional testing framework (Selenium, Cucumber, Jubula)
- Jenkins (or similar CI/CD tools)
- Test monitor system for functional testing
- Test database (if necessary) to manage alert data

3.2 Test Environment

Testing should be performed within a system configured as closely as possible to a production system. If the UI is a web application, then testing should be performed against a list of approved browsers.

4 Test Deliverables

- Test results from CI/CD tools
- Test metrics
- Bug reports
- Customer/user sign off