Testing Document

Project:
A World of Things

Client:
Julian Hambleton-Jones

Team:
Funge

Team Members:

14214742 - Matthew Botha
14446619 - Gian Paolo Buffo
14027021 - Matthias Harvey
14035538 - Dillon Heins

Contents

Disclaimer: The system is still under development. All documentation will be updated as the system progresses

1 Introduction

In order to test our system, we need to do unit tests, integration and system tests and performance and stress tests. There are two main halves, namely the back end and front end.

2 IoT Back End

2.1 Test Plan

In order to fully test the IoT back end component of the project, we need to ensure that a number of connections are in place and are working as expected:

- From the device to IoT
- From IoT to Lambda
- From Lambda to DynamoDB

- From API Gateway to Lambda
- From Lambda to the device's shadow via IoT
- From the shadow to the device

These connections are made up of the device (Arduino), IoT, Lambda and the API Gateway. Each of these parts must be tested individually and then together as a whole. This will be laid out in the sections below.

Currently, there is no automated processes to test the use cases below, and the testing has to be done manually. The test are performed whenever a change is made to the respective subsystem, so the testing is done during development and not on a separate schedule. Later on in development, the tests will be automated.

2.2 Scope

The scope of this section of the testing encompasses the above mentioned parts and how they interoperate. This includes the following subsystems and use-cases:

- Plant/Device Management
  - Edit Plant Configuration/Settings
- Device Communications
  - Send Updated Settings/Configuration to Device
  - Send Readings Summary to Lambda

2.3 Testing Strategy

2.3.1 Unit Testing

Device Mocking: In order to test the functionality of the device, we have created a mock device using the AWS IoT Device SDK for JavaScript, and have coded the mock device using NodeJS. The mock device is not currently handled by a testing environment and has to be managed manually via the terminal. The mock device makes use of npm to retrieve the AWS IoT Device SDK for JavaScript so that the mock device can connect to IoT.

- Pre-conditions:

  - The device should have the correct certificates created and linked
  - A corresponding thing should have been created on IoT for the device

- The device should connect
- The device should send an update message
- The device should listen for a response
- The device should retrieve any shadow updates from IoT
- Post-conditions:
  - The device should have connected successfully
  - The package should have been delivered
  - Any shadow updates should have been applied ot the device

IoT rule test:
When the mock device test is run, it will send an MQTT message to the IoT platform. From here, we can check whether the device shadow has been updated and if the rule has been triggered. If the shadow is updated, then the connection was successful. If the connection was successful but the rule was not triggered, then the rule is not working. We still need to automate this process. Because IoT is on the AWS servers, there is not (so far as we are aware) any testing framework that can trigger and test this use case, and the test has to be performed manually.

- Pre-conditions:
  - A test thing must have been created
  - An IoT rule that corresponds to the test thing must have been created
- An MQTT message should be sent on the corresponding topic to the rule.
- The rule should react to the topic
- Post-conditions:
  - The rule has been executed

Lambda Function Test:
The AWS Lambda console has testing integrated into the platform. We have written a test package to send to Lambda that simulates the messages sent from an IoT device through an IoT rule. The test runs a mock DynamoDB database and will return if the test was successful or not. We can also use the logging feature to check the test results. We still need to automate this process.

- Pre-conditions:
  - The Lambda function has been created
  - A test case has been written and stored in the integrated testing component of AWS Lambda
- The test case is run using the AWS Lambda testing interface on the Lambda console
- Post-conditions:
  - The Lambda function should have run

API Gateway and Thing Shadows:
Because we have not yet implemented the thing shadow interface and functionality, there is no testing for the API Gateway connection or thing shadow yet.

2.3.2 System and Integration Testing

- Communication between a device and IoT
  - Pre-conditions:
    - A device exists and is connected to the internet
    - A thing has been created that relates to the device
    - The device has the correct credentials
    - There is an IoT rule subscribed to the topic related for the thing
  - The device should be able to send a message to IoT
  - IoT should be able to pick up the message and fire a rule
  - IoT should be able to change the shadow of the thing
  - The thing shadow should update the device
  - Post-conditions:
    - An IoT rule has been fired by the device
    - The thing shadow has been updated
    - The device is synchronised with the thing shadow
- Communication between IoT and Lambda
  - Pre-conditions:
    - There is an IoT rule that triggers a Lambda function
    - The Lambda function that is to be triggered exists
  - An IoT rule should be able to be triggered
  - The rule should be able to trigger a Lambda function
  - The data from the rule should be sent to the Lambda function
  - The Lambda function should receive the data from the rule
  - The Lambda function should execute correctly
  - Post-conditions:
    - An IoT rule triggers a Lambda function
    - Data is sent from an IoT rule to a Lambda function
    - A Lambda function retrieves data from an IoT rule
- Communication between Lambda and DynamoDB
  - Pre-conditions:
    - There is a Lambda function that uses DynamoDB
  - A Lambda function is triggered
  - The Lambda function saves an item in a DynamoDB table
  - The Lambda function correctly retrieves a saved item
  - Post-conditions:
    - An item is saved in a DynamoDB table from a Lambda function
    - An item is retrieved by a Lambda function from DynamoDB

2.3.3 Performance and Stress Testing

There is no testing for performance or stress testing as of yet.

3 Front End

3.1 Test Plan

The focus of front-end tests will be to ensure that all AngularJS components integrate correctly with their dependant components, as well as with the Java backend (which in turn will communicate with the AWS services). Protractor will be used for end-to-end (e2e) integration tests, and Karma will be used for unit testing each Angular component. Integration tests form the backbone of any frontend system and will be as extensive and complete as possible. User scenarios will be simulated and run via Protractor to test all possible user interaction. Unit tests will be run on all critical frontend components.

Test creation will form part of the daily design and implementation workflow. It is not a separate ”phase” - as soon as new functionality is added or changed, accompanying tests should be created and executed. If a bug or error is detected outside of testing, the error or bug will be immediately fixed and an accompanying test will be added. Test automation will be implemented to ensure that all tests are executed every time a new commit is made.

3.2 Test Scenarios

3.2.1 Unit Testing

Unit tests will only be written for critical frontend components if the need arises

3.2.2 Integration (end-to-end) Testing

Basic application necessities:

- Application should have a title
- Application should not allow a user unauthorised access to the site

Landing Page:

- *Pre-conditions:*
  - User accesses web application via frontend
  - User is not logged in
- Login
  - *Pre-conditions:*
    - User must not be logged in
    - User must already have an account
  - *Post-conditions:*
    - User must be logged in
  - User should be able to switch to registration form if they do not have an account
  - User should be able to log in if they have entered the correct credentials and be redirected to the main site dashboard
  - User should not be logged in if they enter no credentials or the incorrect credentials and an appropriate error message should be displayed
- Registration
  - *Pre-conditions:*
    - User must not be logged in
    - User must have navigated to the registration form

    - User must not have an account
    - User's email or username must not already be in use
  - *Post-conditions:*
    - User must have an account registered on the system backend
    - User must be logged in
  - User should be able to switch to the login form if they already have an account
  - User should have their details saved in the backend and be automatically logged in if they have entered all their details and their details are not already on the system
  - User should not have their details registered if they do not enter all the required information or if their username or email is already on the system

Main site:

- Navbar
  - *Pre-conditions:*
    - User must be logged in
  - If the user clicks "Logout", the user's credentials are immediately discarded and the user is redirected back to the landing page. *Postcondition: The user is logged out and cannot access the site anymore*