
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

1	Introduction	1
1.1	Purpose	1
1.2	Scope	1
1.3	Test Environment	2
1.4	Assumptions and Dependencies	3
2	Test Items	4
2.1	Functional Features to be Tested	4
2.2	Non-Functional Features to be Tested	5
3	Test Cases	5
3.1	Database Management	5
3.1.1	Persist to Database	5
3.1.2	Query Database	7
3.1.3	Remove from Database	8
3.2	User Management	9
3.2.1	Register User	9
3.2.2	Login	10
3.2.3	Logout	12
3.2.4	Update User	12
3.3	Plant Management	13
3.3.1	Create Plant	13
3.3.2	Delete Plant	15
3.3.3	List Plants	16

3.3.4	View Plant	17
3.3.5	Update Plant	18
3.3.6	Configure Lights	20
3.3.7	Configure Pump	21
3.3.8	Configure Fan	22
3.4	Front End	23
3.4.1	Test Plan	23
3.4.2	Test Scenarios	24
3.5	Device Communications	25
3.5.1	Test Plan	25
3.5.2	Scope	26
3.5.3	Testing Strategy	26
3.6	Usability	29
4	Test Deliverables	29
5	Unit Test Report	30
5.1	Detailed Test Results	30
5.1.1	Overview of Test Results	30
5.1.2	Functional Requirements Test Results	30
5.1.3	Non-Functional Requirements Test Results	34
5.2	Other	34
5.2.1	Build Log	34
5.3	Conclusion and Recommendations	35

Table 1: Version Table

Version	Date	Description
0.1	22/05/2016	Vision, scope, architectural requirements and initial architecture design.
0.2	29/07/2016	Creation of separate documents for architecture design, software requirements, testing and user manual. Each populated with the relevant information for the project at this stage.
0.3	11/09/2016	Remake of all documents. Combination of them into a single document. Documents follow guidelines as discussed with lecturer.

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

1 Introduction

The Internet of Things (IoT) is a development of the Internet which involves the networking of every day physical devices allowing them to send and receive data. These devices are embedded with electronics, sensors, software as well as some form of Internet or network connectivity.

IoT is a relatively new development and has a large possibility of becoming a ubiquitous technology as well as allowing us to view the world from a different perspective. The potential it contains for innovation is endless.

We as a group were given the opportunity to use the Amazon Web Services (AWS) IoT platform to create an IoT project of our own desires. This document details our 'A World of Things' project.

A remote, real-time plant monitoring and environment control system is implemented using a cloud based, micro-services, Platform as a Service (PaaS) architecture, with the front-end interface using the Model-View-Controller (MVC) model.

In order to ensure that our system runs as expected and meets all of the functional and non-functional requirements, it is important to fully test the system. A testing plan has therefore been drawn up and implemented.

1.1 Purpose

This document lays out the testing plan for our system (technologies used, unit tests etc.) as well as the testing report, detailing the results and artefacts produced by the implementation of this plan.

1.2 Scope

This document contains two main sections, the testing plan (Sections 2 to 4) and the Unit Test Report (Section 5).

Inside the testing plan, the testing items have been identified (Section 2). The functional features (2.1) are the functional use cases from the system and have been taken from the

Architectural and Functional Requirements Design Specification document, where they are explained in greater detail.

Section 2.2 explains the non-functional features that will be tested. *This will be added to and expanded on later.*

Section 3 goes into greater detail about each of the use cases, listing the pre- and post-conditions of each test, and the different conditions that will be tested, including the objective, input and expected outcome of each condition.

The test deliverables are outlined in section 4.

The Unit Test Report (Section 5) includes the following:

- A detailed overview (5.1) of each of each of the tests, naming which tests passed and which failed, along with the reasons for the results achieved.
- Other artefacts generated from the testing plan (5.2)
- A final conclusion and some recommendation (5.3) for further testing.

At a later stage, this document will also include the usability report.

1.3 Test Environment

The properties of the testing environment are as follows:

- Programming Languages:
 - Java
 - NodeJS
 - AngularJS
 - JavaScript
 - HTML5
- Testing Frameworks:
 - JUnit
 - Mockito
 - Maven
 - Spring (dependency injection)
 - Travis (integration)

- Karma
 - Protractor
- Coding Environments:
 - IntelliJ IDEA
 - AWS Lambda Inline Console
 - Arduino IDE
- Operating System:
 - Java Virtual Machine (JVM)
 - OS not known
- Internet Browsers:
 - Internet Explorer
 - Firefox
 - Chrome
 - Opera
 - Safari

1.4 Assumptions and Dependencies

During testing, it is assumed that:

- A reliable internet connection is established and used
- The AWS services are up and running
- The device used for testing is configured and connected to the internet

The testing requires the following dependencies:

- AWS SDK

2 Test Items

In order to fully test our system, the testing plan needs to detail the different sections of the system that need to be tested. There are two different kinds of items that need to be tested: functional and non-functional features of the system.

2.1 Functional Features to be Tested

The following test items are the functional use cases of the system and have been taken from the Architectural and Functional Requirements Design Specification document, where they are explained in greater detail. The functional features fall into four subsystems, namely database management, user management, plant/device management and device communications.

- Database Management
 - Persist to Database
 - Query Database
 - Remove from Database
- User Management Subsystem
 - Register User
 - Login
 - Create User Session
 - Logout
 - Terminate Session
 - Update User
- Plant/Device Management Subsystem
 - Create Plant
 - Delete Plant
 - List Plants
 - View Plant
 - Subscribe to Topic
 - Update Plant
 - Create Thing
 - Configure Lights

- Configure Pump
- Configure Fan
- Device Communications Subsystem
 - Update Thing Shadow
 - Update Web Interface

2.2 Non-Functional Features to be Tested

This section has not been fully planned yet and will be completed in later versions of this document

- Usability

3 Test Cases

This section of the testing plan will detail each of the tests that will be performed, naming their pre- and post-conditions, the different conditions to be tested, and the objective, input and expected outcome of each of those conditions. The items are grouped into the different subsystems that they fall under.

3.1 Database Management

3.1.1 Persist to Database

Pre-conditions:

- DynamoDB is running
- The correct table(s) have been created

Post-conditions:

- A new entry has been entered into the correct table in the database

3.1.1.1 Condition: Correct Key Fields and Unique Keys

Objective To test if data is persisted to the database if the correct key fields have been filled in (minimum requirements for an insertion) and a unique key pair is given.

Input

- Table Name: "testTable"
- Key1 Value: "uniqueKey1"
- Key2 Value: "uniqueKey2"

Outcome A new record is created in the "testTable" table with the correct keys.

3.1.1.2 Condition: Correct Key Fields but Non-Unique Keys

Objective To test if data is persisted to the database if the correct key fields have been filled in (minimum requirements for an insertion) but a non-unique key pair is given.

Input

- Table Name: "testTable"
- Key1 Value: "nonUniqueKey1"
- Key2 Value: "nonUniqueKey2"

Outcome No new record is created in the "testTable".

3.1.1.3 Condition: Not All Key Fields Given

Objective To test if data is persisted to the database if the correct key fields have not been filled in (minimum requirements for an insertion).

Input

- Table Name: "testTable"
- Key1 Value: "uniqueKey1"
- Key2 Value: none

Outcome No new record is created in the "testTable" table.

3.1.2 Query Database**Pre-conditions:**

- DynamoDB is running
- The correct table(s) have been created
- There is a record created in the table

Post-conditions:

- One or more records is retrieved from the database

3.1.2.1 Condition: Authorised Role

Objective To test if data can be retrieved from the database via a query using the correct permission role

Input

- Table Name: "testTable"
- Key1 Value: "key1"
- Key2 Value: "key2"
- Role: "databaseQueryRole"

Outcome A record with the same key values as the input is retrieved from the database.

3.1.2.2 Condition: Unauthorised Role

Objective To test if data can be retrieved from the database via a query using an incorrect permission role

Input

- Table Name: "testTable"
- Key1 Value: "key1"
- Key2 Value: "key2"
- Role: "generalRole"

Outcome No record is retrieved from the database.

3.1.3 Remove from Database

Pre-conditions:

- DynamoDB is running
- The correct table(s) have been created
- There is a record created in the table

Post-conditions:

- A record is removed from the database

3.1.3.1 Condition: Authorised Role

Objective To test if data can be deleted from the database via a query using the correct permission role

Input

- Table Name: "testTable"
- Key1 Value: "key1"
- Key2 Value: "key2"
- Role: "databaseDeleteRole"

Outcome A record with the same key values as the input is deleted from the database.

3.1.3.2 Condition: Unauthorised Role

Objective To test if data can be deleted from the database via a query using an incorrect permission role

Input

- Table Name: "testTable"
- Key1 Value: "key1"
- Key2 Value: "key2"
- Role: "generalRole"

Outcome No record is deleted from the database

3.2 User Management**3.2.1 Register User****Pre-conditions:**

- AWS is up and running

Post-conditions:

- A new user is registered and logged in to the system

3.2.1.1 Condition: Unique User Name Email Provided

Objective To test if a new user can be registered using a unique user name email address.

Input

- Email Address: "unique@email.com"
- User Name: "UniqueUser"
- Password: "pass123"

Outcome A new user is created and logged into the system.

3.2.1.2 Condition: User Name or Email Taken

Objective To test if a new user can be registered using a unique user name email address.

Input

- Email Address: "unique@email.com"
- User Name: "UniqueUser"
- Password: "pass123"

Outcome A new user is created and logged into the system.

3.2.2 Login

Pre-conditions:

- Username must not be null
- Password must not be null

- User account associated with username must exist
- The password after being hashed and salted must match that of the provided username

Post-conditions:

- The user should be given temporary user credentials in a response object
- The user should be given access to their information and redirected to their dashboard

3.2.2.1 Condition: Correct Login Details

Objective To test if the user can log in with correct log in details (user name and password).

Input

- User Name: "User1"
- Password: "correctPass"

Outcome A user is logged in and a session is created.

3.2.2.2 Condition: Incorrect Login Details

Objective To test if the user can log in with incorrect log in details (user name and password).

Input

- User Name: "User1"
- Password: "incorrectPass"

Outcome No user is logged in.

3.2.3 Logout

Pre-conditions:

- A user must be registered on the system
- The user must be logged into the system

Post-conditions:

- The user should be no longer logged into the system and as such not be able to access any functionality from the system except to register an account or log in
- If the user wishes to access their account and the functionality of the system, they should have to log back into the system

3.2.3.1 Condition: User is logged in

Objective To test if the user is logged out (session authorisation removed).

Input

- User Session

Outcome The user session is terminated and the user cannot access any page except for the Login and Register pages.

3.2.4 Update User

Pre-conditions:

- User must be logged in
- User editing details should be the owner of those details

Post-conditions:

- The values of the edited details should be reflected in the database

3.2.4.1 Condition: User is logged in

Objective To test if the user details can be edited by the user if the user is logged in.

Input

- User Session
- Update Details Request

Outcome The user details are changed in the database and are reflected on the user details page.

3.2.4.2 Condition: User is logged in

Objective To test if the user details can be edited by a user if the user is not logged in.

Input

- Bad User Session
- Update Details Request

Outcome The user details are not changed in the database.

3.3 Plant Management

3.3.1 Create Plant

Pre-conditions:

- The user should be logged in
- The plant's name should not be null

- The plant's type should not be null
- The plant's age should not be null
- An IoT device should be selected to be associated with the plant

Post-conditions:

- The plant should be assigned a unique identifier
- It should be associated with a particular user
- An IoT device ID should be associated with the plant
- The plant should be persisted in the database with all the above information

3.3.1.1 Condition: Authenticated

Objective Test that a plant can be added while the user is authenticated.

Input

- Username: "TestUser"
- Plant name: "TestPlant"
- Plant type: "TestType"
- Plant age: 0
- Colour: "Red"

Outcome A plant is created.

3.3.1.2 Condition: Not Authenticated

Objective Test that a plant cannot be added when the user is not authenticated.

Input

- Username: "FakeTestUser"
- Plant name: "TestPlant"
- Plant type: "TestType"
- Plant age: 0
- Colour: "Red"

Outcome A plant is not created.

3.3.2 Delete Plant**Pre-conditions:**

- The user should be logged in
- A particular plant should be selected
- The user should be the owner of the plant

Post-conditions:

- The plant's entry should be removed from the database
- Any information related to the plant must also be removed from the relevant tables

3.3.2.1 Condition: Authenticated

Objective Test that a plant can be deleted while the user is authenticated.

Input

- Username: "TestUser"
- Plant ID: 1

Outcome The plant is deleted.

3.3.2.2 Condition: Not Authenticated

Objective Test that a plant cannot be deleted when the user is not authenticated.

Input

- Username: "TestFakeUser"
- Plant ID: 1

Outcome A plant is not deleted.

3.3.3 List Plants

Pre-conditions:

- The user should be logged in
- Username must not be null

Post-conditions:

- A count of the plants belonging to the supplied username should be returned
- A limit to the amount of plants returnable should be provided
- A response object containing a list of all plants associated with a user and all their individual details should be returned

3.3.3.1 Condition: Authenticated

Objective Test that a list of plants can be retrieved when the user is authenticated.

Input

- Username: "TestUser"

Outcome A list of plants belonging to the user is returned.

3.3.3.2 Condition: Not Authenticated

Objective Test that a list of plants cannot be retrieved when the user is not authenticated.

Input

- Username: "TestFakeUser"

Outcome A plant list is not returned.

3.3.4 View Plant**Pre-conditions:**

- The user should be logged in
- A valid plantId should be supplied

Post-conditions:

- All details associated with a particular plant and gathered by the associated IoT device should be displayed. These details could include:
 - Temperature
 - Humidity
 - Light conditions
 - Water flow
 - Soil moisture

3.3.4.1 Condition: Authenticated

Objective Test that plant details can be retrieved when the user is authenticated.

Input

- Username: "TestUser"
- PlantId: 1

Outcome The plant details belonging to the specified ID and user are returned.

3.3.4.2 Condition: Not Authenticated

Objective Test that plant details cannot be retrieved when the user is not authenticated.

Input

- Username: "TestFakeUser"
- PlantId: 1

Outcome The plant details are not returned.

3.3.5 Update Plant

Pre-conditions:

- The user should be logged in
- The plant ID must not be null
- The relevant updated information must be supplied
- The plant must exist

Post-conditions:

- The plant associated with the supplied plant ID must have its details updated in the database
- A response object indicating the operation was successful must be sent back to the client

3.3.5.1 Condition: Authenticated

Objective Test that a plant can be updated when a user is authenticated.

Input

- Username: "TestUser"
- PlantId: 1
- Plant name: "TestPlantUpdated"
- Plant type: "TestTypeUpdated"
- Plant age: 1
- Colour: "Pink"

Outcome The plant details belonging to the specified ID and user are updated.

3.3.5.2 Condition: Authenticated

Objective Test that a plant cannot be updated when a user is not authenticated.

Input

- Username: "TestFakeUser"
- PlantId: 1
- Plant name: "TestPlantUpdated"

- Plant type: "TestTypeUpdated"
- Plant age: 1
- Colour: "Pink"

Outcome The plant details belonging are not updated.

3.3.6 Configure Lights

Pre-conditions:

- The user should be logged in
- A particular plant should be selected
- An RGB value should be specified

Post-conditions:

- The RGB value should be communicated to the device
- The IoT device should subsequently reflect this RGB colour in the wavelengths of the LED strip, if the RGB value was (0, 0, 0) the lights should be turned off
- A response object indicating whether the action was successful should be returned

3.3.6.1 Condition: Authenticated

Objective Test that the lights can be updated when a user is authenticated.

Input

- Username: "TestUser"
- PlantId: 1
- RGB value: (255, 0, 0)

Outcome The desired state is updated to reflect the new light configuration.

3.3.6.2 Condition: Not Authenticated

Objective Test that the lights cannot be updated when a user is not authenticated.

Input

- Username: "TestFakeUser"
- PlantId: 1
- RGB value: (255, 0, 0)

Outcome The desired state is not updated.

3.3.7 Configure Pump

Pre-conditions:

- The user should be logged in
- A particular plant should be selected
- The configuration details should be specified

Post-conditions:

- The configuration should be communicated to and applied on the IoT device, the pump should run for the specified amount of time
- If the pumpTime value was 0 the pump should turn itself off
- A response object indicating whether the action was successful should be returned

3.3.7.1 Condition: Authenticated

Objective Test that the pump can be updated when a user is authenticated.

Input

- Username: "TestUser"
- PlantId: 1
- pumpTime: 5

Outcome The desired state is updated to reflect the new pump configuration.

3.3.7.2 Condition: Not Authenticated

Objective Test that the pump cannot be updated when a user is not authenticated.

Input

- Username: "TestFakeUser"
- PlantId: 1
- pumpTime: 5

Outcome The desired state is not updated.

3.3.8 Configure Fan**Pre-conditions:**

- The user should be logged in
- A particular plant should be selected
- The configuration details should be specified

Post-conditions:

- The configuration details should be communicated to and applied on the IoT device
- A response object indicating whether the action was successful should be returned

3.3.8.1 Condition: Authenticated

Objective Test that the fan can be updated when a user is authenticated.

Input

- Username: "TestUser"
- PlantId: 1
- fanSpeed: 10

Outcome The desired state is updated to reflect the new fan configuration.

3.3.8.2 Condition: Not Authenticated

Objective Test that the fan cannot be updated when a user is not authenticated.

Input

- Username: "TestFakeUser"
- PlantId: 1
- fanSpeed: 10

Outcome The desired state is not updated.

3.4 Front End

3.4.1 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.

3.4.2 Test Scenarios

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

3.4.2.2 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*

3.5 Device Communications

3.5.1 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.

3.5.2 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:

- Device Communications
 - Update Thing Shadow
 - Update Web Interface

3.5.3 Testing Strategy

3.5.3.1 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 to 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.

3.5.3.2 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

3.6 Usability

This section will be completed in later versions of this document

4 Test Deliverables

Through the implementation of this testing plan, the following artefacts are generated:

- Test Plan
- Test Report
- Maven Testing Logs

5 Unit Test Report

5.1 Detailed Test Results

5.1.1 Overview of Test Results

All of the functional requirements test cases have been implemented and each of the tests have passed. This is because of the continuous testing methodology that has been adopted, meaning that tests are constantly being written and ensures that only working code is merged with the main branch of the code (the branch that was tested).

5.1.2 Functional Requirements Test Results

For each of the functional requirements test cases, the following results (along with the reasons for the results) were generated:

5.1.2.1 Persist to Database (TC 3.1.1)

1. The correct key fields and unique keys added a new record in the correct table
2. The correct key fields but non-unique keys failed to add a new record in the table
3. Incorrect key fields failed to add a new record in the table

Result: Pass

5.1.2.2 Query Database (TC 3.1.2)

1. An authorised role is able to query the database and retrieve a record
2. An unauthorised role is unable to query the database

Result: Pass

5.1.2.3 Remove from Database (TC 3.1.3)

1. An authorised role is able to delete a record from the database

2. An unauthorised role is unable to delete a record from the database

Result: Pass

5.1.2.4 Register User (TC 3.2.1)

1. A new user is registered and logged in when the user provides a unique email address and user name
2. A new user is not registered when the user provides a non-unique email address or user name

Result: Pass

5.1.2.5 Login (TC 3.2.2)

1. A user is logged in when they provide the correct login details
2. A user is not logged in when they provide incorrect login details and cannot access any page except the login and register pages

Result: Pass

5.1.2.6 Logout (TC 3.2.3)

1. A logged in user is logged out and their session is terminated, allowing them to only access the login and register pages until they log in again

Result: Pass

5.1.2.7 Update User (TC 3.2.4)

1. The user details are updated and persisted in the database when that user is logged in
2. The user details are not updated if that user is not logged in

Result: Pass

5.1.2.8 Create Plant (TC 3.3.1)

1. An authenticated user was able to create a plant and persist it to the database.
2. An unauthenticated user was not able to create a plant.

Result: Pass

5.1.2.9 Delete Plant (TC 3.3.2)

1. An authenticated user was able to delete a plant that belonged to them and remove it from the database.
2. An unauthenticated user was not able to delete a plant.

Result: Pass

5.1.2.10 List Plants (TC 3.3.3)

1. An authenticated user was able to view all plants that belonged to them.
2. An unauthenticated user was not able to view any plant details.

Result: Pass

5.1.2.11 View Plant (TC 3.3.4)

1. An authenticated user was able to view a specific plant's details.
2. An unauthenticated user was not able to view any plant details.

Result: Pass

5.1.2.12 Update Plant (TC 3.3.5)

1. An authenticated user was able to update a specific plant's details.
2. An unauthenticated user was not able to update any plant details.

Result: Pass

5.1.2.13 Configure Lights (TC 3.3.6)

1. An authenticated user was able to update the device state for the light configuration of a plant that belonged to them.
2. An unauthenticated user was not able to update the device state.

Result: Pass

5.1.2.14 Configure Pump (TC 3.3.7)

1. An authenticated user was able to update the device state for the pump configuration of a plant that belonged to .
2. An unauthenticated user was not able to update the device state.

Result: Pass

5.1.2.15 Configure Fan (TC 3.3.8)

1. An authenticated user was able to update the device state for the fan configuration of a plant that belonged to .
2. An unauthenticated user was not able to update the device state.

Result: Pass

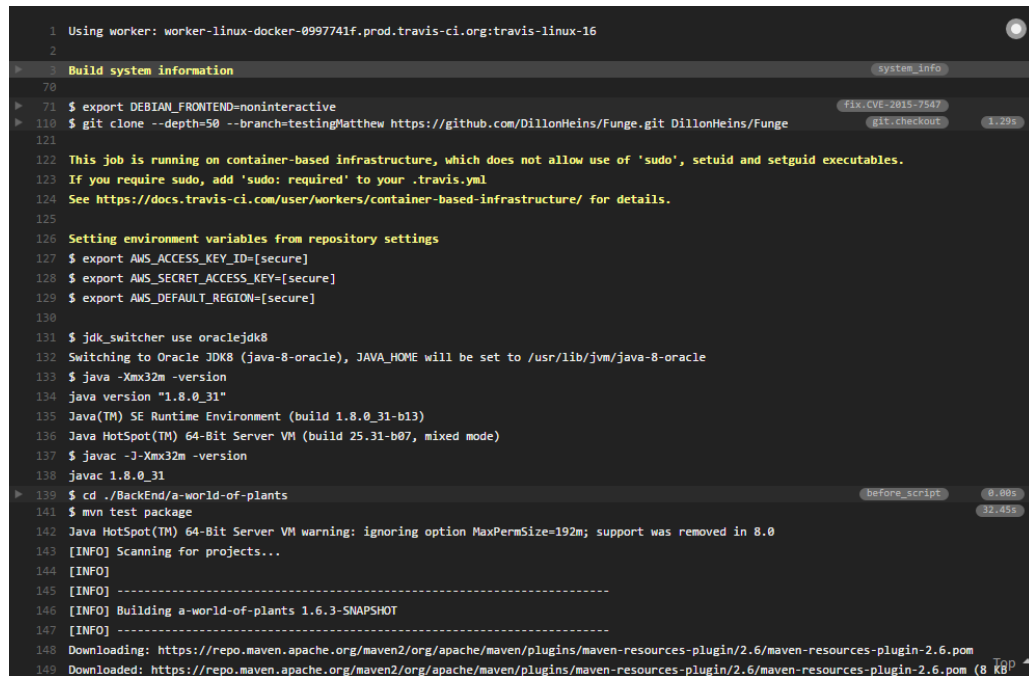
5.1.3 Non-Functional Requirements Test Results

This has not been implemented and will be completed in later versions of this document

5.2 Other

5.2.1 Build Log

The following build log was generated by Travis CI that shows the Maven build and testing process.



```
1 Using worker: worker-linux-docker-0997741f.prod.travis-ci.org:travis-linux-16
2
3 Build system information system_info
70
71 $ export DEBIAN_FRONTEND=noninteractive
72
110 $ git clone --depth=50 --branch=testingMatthew https://github.com/DillonHeins/Funge.git DillonHeins/Funge git_checkout 1.29s
121
122 This job is running on container-based infrastructure, which does not allow use of 'sudo', setuid and setgid executables.
123 If you require sudo, add 'sudo: required' to your .travis.yml
124 See https://docs.travis-ci.com/user/workers/container-based-infrastructure/ for details.
125
126 Setting environment variables from repository settings
127 $ export AWS_ACCESS_KEY_ID=[secure]
128 $ export AWS_SECRET_ACCESS_KEY=[secure]
129 $ export AWS_DEFAULT_REGION=[secure]
130
131 $ jdk_switcher use oraclejdk8
132 Switching to Oracle JDK8 (java-8-oracle), JAVA_HOME will be set to /usr/lib/jvm/java-8-oracle
133 $ java -Xmx32m -version
134 java version "1.8.0_31"
135 Java(TM) SE Runtime Environment (build 1.8.0_31-b13)
136 Java HotSpot(TM) 64-Bit Server VM (build 25.31-b07, mixed mode)
137 $ javac -J-Xmx32m -version
138 javac 1.8.0_31
139 $ cd ../Backend/a-world-of-plants before_script 0.80s
140 $ mvn test package 32.45s
141
142 Java HotSpot(TM) 64-Bit Server VM warning: ignoring option MaxPermSize=192m; support was removed in 8.0
143 [INFO] Scanning for projects...
144 [INFO]
145 [INFO] -----
146 [INFO] Building a-world-of-plants 1.6.3-SNAPSHOT
147 [INFO] -----
148 Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-resources-plugin/2.6/maven-resources-plugin-2.6.pom
149 Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-resources-plugin/2.6/maven-resources-plugin-2.6.pom (8 KB)
```

Figure 1: The initial configuration and downloading of dependencies

```

779 Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-providers/2.12.4/surefire-providers-2.12.4.pom
780 Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-providers/2.12.4/surefire-providers-2.12.4.pom (3 KB
    at 109.2 KB/sec)
781 Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-junit4/2.12.4/surefire-junit4-2.12.4.jar
782 Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-junit4/2.12.4/surefire-junit4-2.12.4.jar (37 KB at
    901.4 KB/sec)
783
784 -----
785 T E S T S
786 -----
787 Running cf.funge.aworldofplants.test.TestThing
788 Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.198 sec
789 Running cf.funge.aworldofplants.test.TestPlant
790 Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.009 sec
791
792 Results :
793
794 Tests run: 10, Failures: 0, Errors: 0, Skipped: 0
795
796 [INFO]
797 [INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ a-world-of-plants ---
798 [WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
799 [INFO] Copying 0 resource
800 [INFO]
801 [INFO] --- maven-compiler-plugin:3.5.1:compile (default-compile) @ a-world-of-plants ---
802 [INFO] Nothing to compile - all classes are up to date
803 [INFO]
804 [INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ a-world-of-plants ---
805 [WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
806 [INFO] skip non existing resourceDirectory /home/travis/build/DillonHeins/Funge/BackEnd/a-world-of-plants/src/test/resources
807 [INFO]
808 [INFO] --- maven-compiler-plugin:3.5.1:testCompile (default-testCompile) @ a-world-of-plants ---
809 [INFO] Nothing to compile - all classes are up to date

```

Figure 2: The running of tests

5.3 Conclusion and Recommendations

The system passes all functional requirements that were tested, however there are still some requirements that have not yet been implemented and therefore are not tested yet. The non-functional requirements still need to be tested and will appear in later versions of this document.

It is believed that the tests are sufficiently accurate, but more conditions will be added at a later stage to ensure that the tests are robust enough to find any bugs that there might be in the system.