Architectural and Functional Requirements

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

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.

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.[1]

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.

2 Vision

This project aims to showcase the capabilities of the Amazon Web Services Internet of Things platform in conjunction with other AWS technologies, such as events processing and cloud computing. The focus of the project will be the education of students with regard to agriculture and plant sciences. We plan to motivate students to become interested in agriculture by creating an "Internet of Plants".

The 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.

This involves the networking and monitoring of living plants in order to analyse aspects of their environment - such as water intake, lighting, humidity, moisture, and temperature. It also involves the controlling of the environment of the plants through the adjustment of the amount of water they receive, the wavelengths of their lights and air flow through the use of fans.

Users should be able to view data gathered about their plants through the use of graphs so that they can make informed decisions about what adjustments to the environment they should make.

The main purpose of the platform is to encourage younger generations to become interested in agriculture and, in doing so, help stimulate South Africa's agricultural industry. We plan to implement gamification on our platform as a way to encourage users to participate.

Given enough time, AI learning could be employed to optimise the conditions under which plants grow. By combining automation and AI learning, we could create an interesting challenge for the users: Grow a plant better than our control plant grown with the help of AI.

3 Background

This project was created by Julian Hambleton-Jones (AWS) for the purpose of utilising the AWS IoT platform in an interesting and innovative manner. After exploring many IoT options and projects, it was decided that the project would use IoT to manage and analyse a network of living plants through the use of IoT devices and an events-based server.

3.1 Future Business/Research Opportunities

Through the use of sensors to continuously monitor the environment of plants a large amount of important information can be gathered. This information can create opportunities for the learning of how particular actions affect the environment of plants.

This data can also be used as training data for AI so that the controlling of the environment can be fully automated such that the AI attempts to tend to and grow a plant as best as possible.

These activities could help agricultural industries to improve crop yields for an increasingly growing human population.

3.2 The Project's Problem

The project is to use the AWS IoT hardware platform in an interesting and innovative manner to solve an existing problem. It was decided that the problem to be tackled is the growing lack of interest of young South Africans in agriculture.

4 Important Terminology

Some important words and phrases used in this document are detailed below:

- **A World of Plants** Name of the entire system.
- **AWS** Amazon Web Services. A secure cloud services platform offering compute power, database storage, content delivery and other functionality.
- **Plant Box** The enclosure which contains the plants, with all the monitoring and automation equipment.
- **Achievements** Gamified awards given to users for completing certain tasks.
- **IoT** Internet of Things
- **Thing** An IoT device that has a thing shadow, publishes to topics and is connected to sensors
- **Thing Shadow** A JSON string that is associated with the state of a thing
- **IoT Topic** A stream of JSON strings that are published to. A stream can be subscribed to, where the subscriber will receive all JSON messages published to that topic

5 Functional Requirements

5.1 Use Case Prioritisation

The use cases for each subsystem have been sorted into *Critical* (the system could not work without it), *Important* (the system would work, but would be very different without it) and *Nice-to-have* (not very important; low priority).

5.1.1 User Management Subsystem

- Critical
  - Register account
  - Login
  - Logout
- Important
  - Update User Details
- Nice-to-have
  - View gamification information/get user details

5.1.2 Plant/Device Management Subsystem

- Critical
  - Create plant
  - List plants
  - Create and associate a virtual 'Thing' with a plant
  - View plant status
  - Configure light settings
- Important
  - Update plant details
  - Configure pump settings
  - Configure fan speeds
  - Delete plant

5.1.3 Device Communications Subsystem

- Critical
  - Send readings summary to Lambda
- Nice-to-have
  - Complex event processing on device communications

5.2 Use Case/Service Contracts

The system is split into three distinct subsystems, mainly the User Management subsystem, the Plant/Device Management Subsystem and the Device Management Subsystem.

In this section, the use cases and service contracts for each subsystem will be laid out and explained.

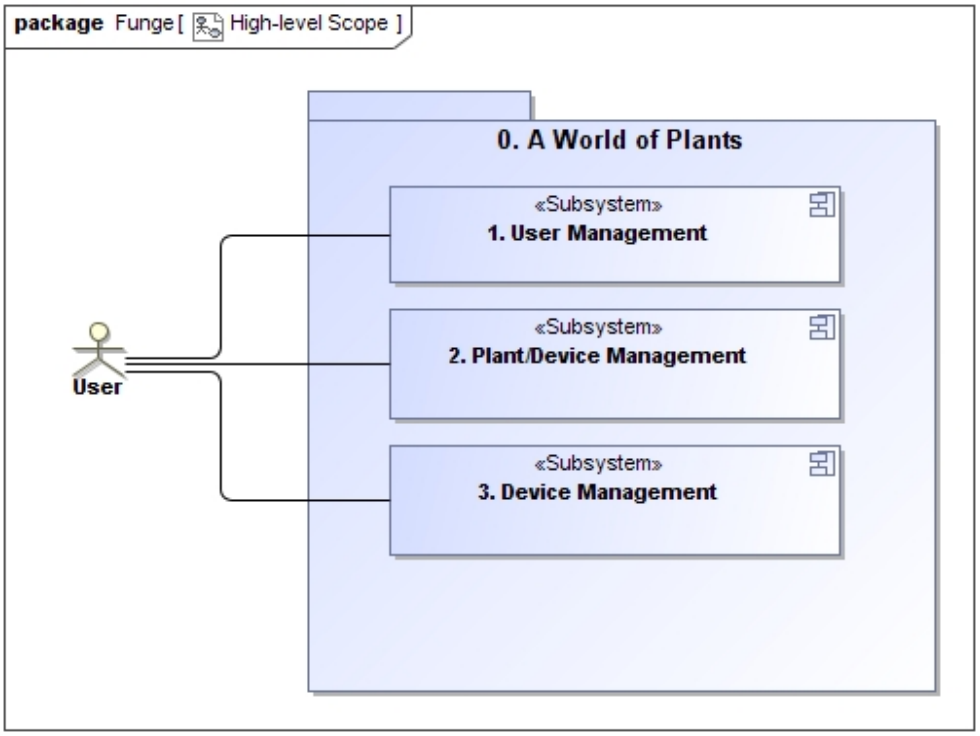

Figure 1: High-level scope, showing the three subsystems

5.2.1 User Management Subsystem

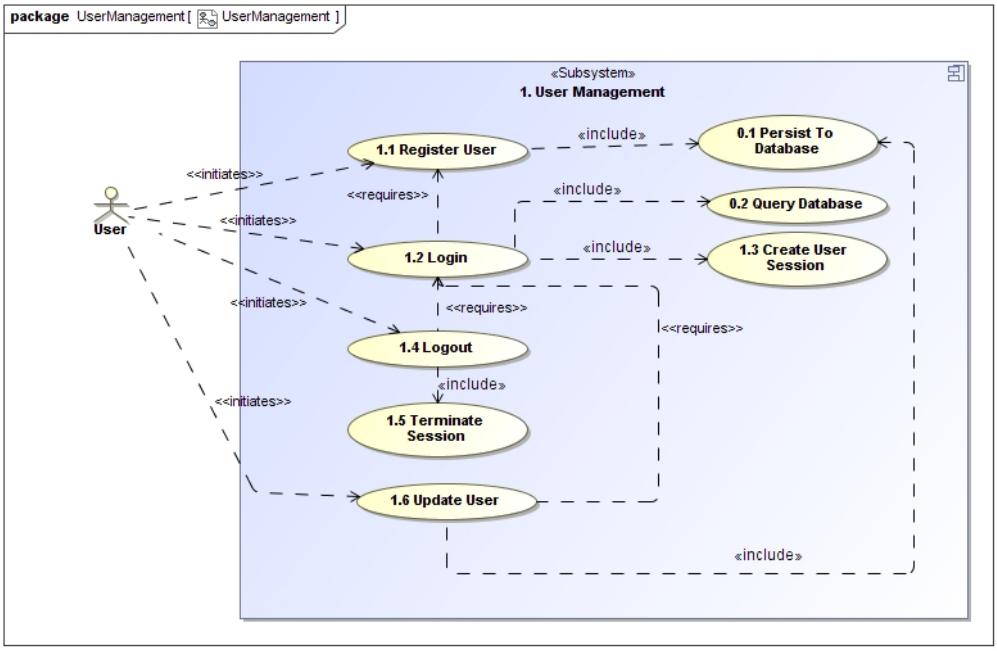

Figure 2: Use Case Diagram for the User Management Subsystem

5.2.1.1 Register Account

Description: A user must register an account before being able to access the system so that they may associated devices and plants with their account.
Prioritisation: Critical
Pre-conditions:

- Email must be unique, obey email standards and not be null
- Username must be unique and not null
- Password must be longer than 8 characters and must not be null

Post-conditions:

- The password must be encrypted using PBKDF2WithHmacSHA1 and salted
- An identity ID must be created for the user

- All the above information must be associated with the username provided and stored in a database

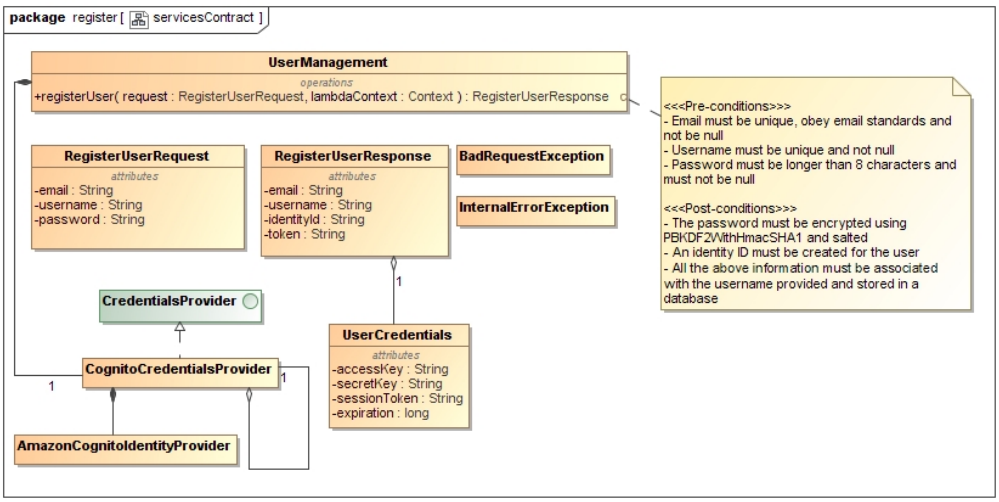

Figure 3: Services contract diagram for registering an account

5.2.1.2 Login

Description: A registered user must be able to login so they may securely view their account and all associated details.
Prioritisation: Critical
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

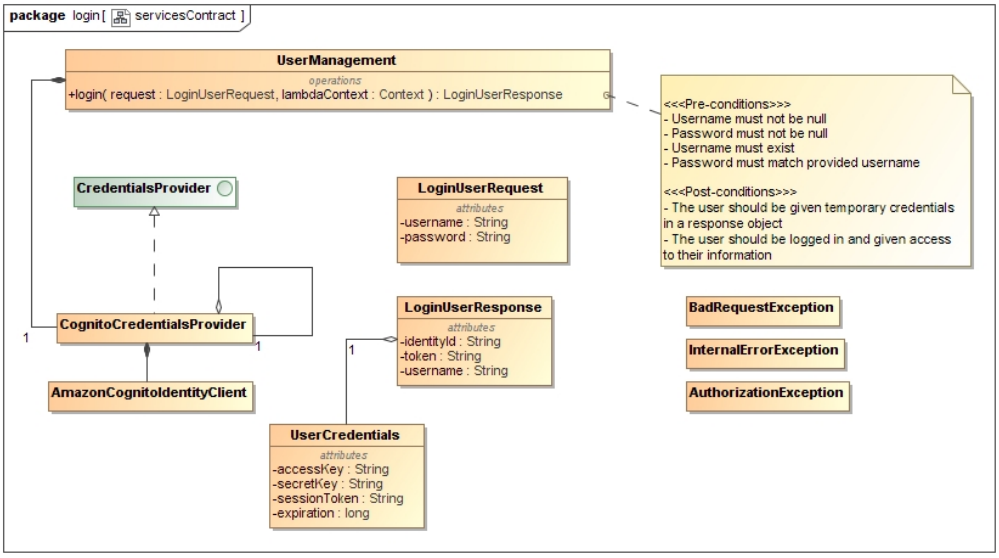

Figure 4: Services contract diagram for login

5.2.1.3 Logout

Description: A registered user must be able to logout securely from the system.
Prioritisation: Critical
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

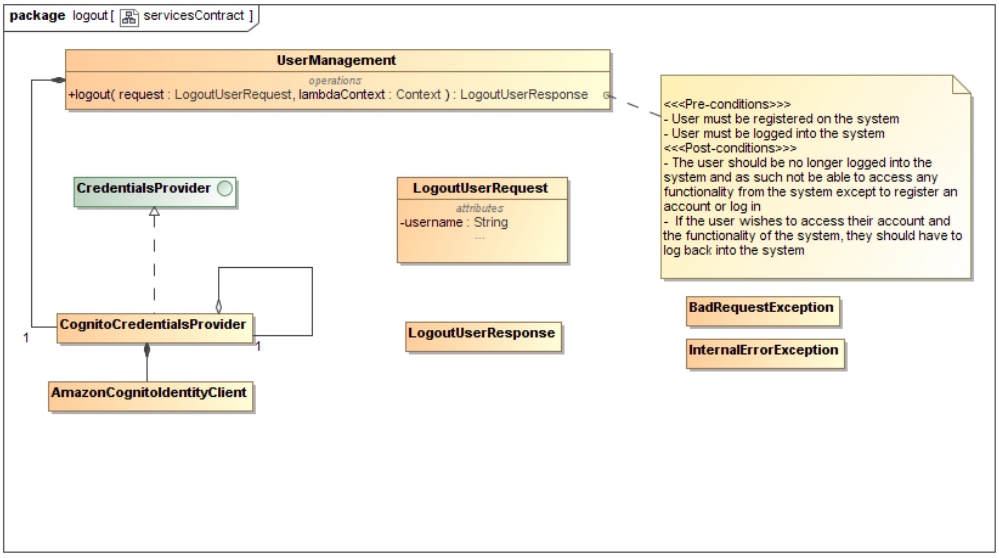

Figure 5: Services contract diagram for logout

5.2.1.4 Update User Details

Description: A registered and logged in user should be able to edit details such as their email and password.
Prioritisation: Important
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

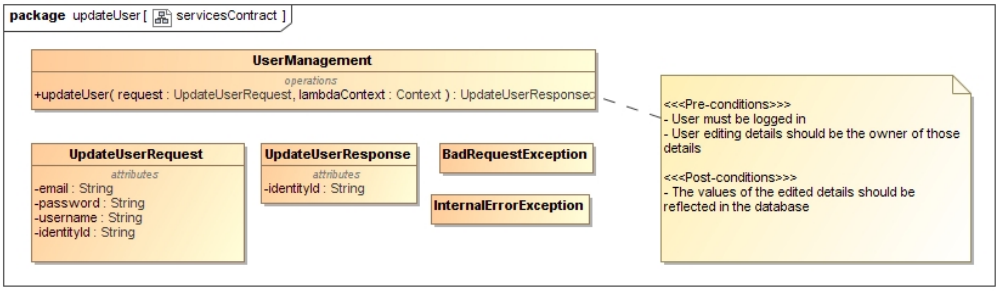

Figure 6: Services contract diagram for update user details

5.2.1.5 View Gamification Details/Get User Details

Description: A registered and logged in user should be able to view details with regard to the progress they have made. This includes details such as their ranked level and experience points for completing tasks.
Prioritisation: Nice-to-have
Pre-conditions:

- User must be logged in

Post-conditions:

- The user's personal details should be returned (username & email)
- The gamification values associated with the user should be displayed to the user (level and experience)

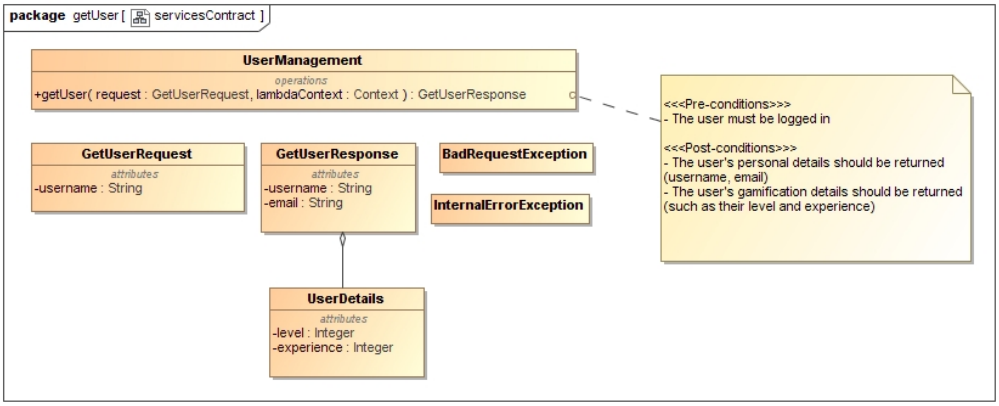

Figure 7: Services contract diagram for view gamification details/get user details

5.2.2 Plant/Device Management Subsystem

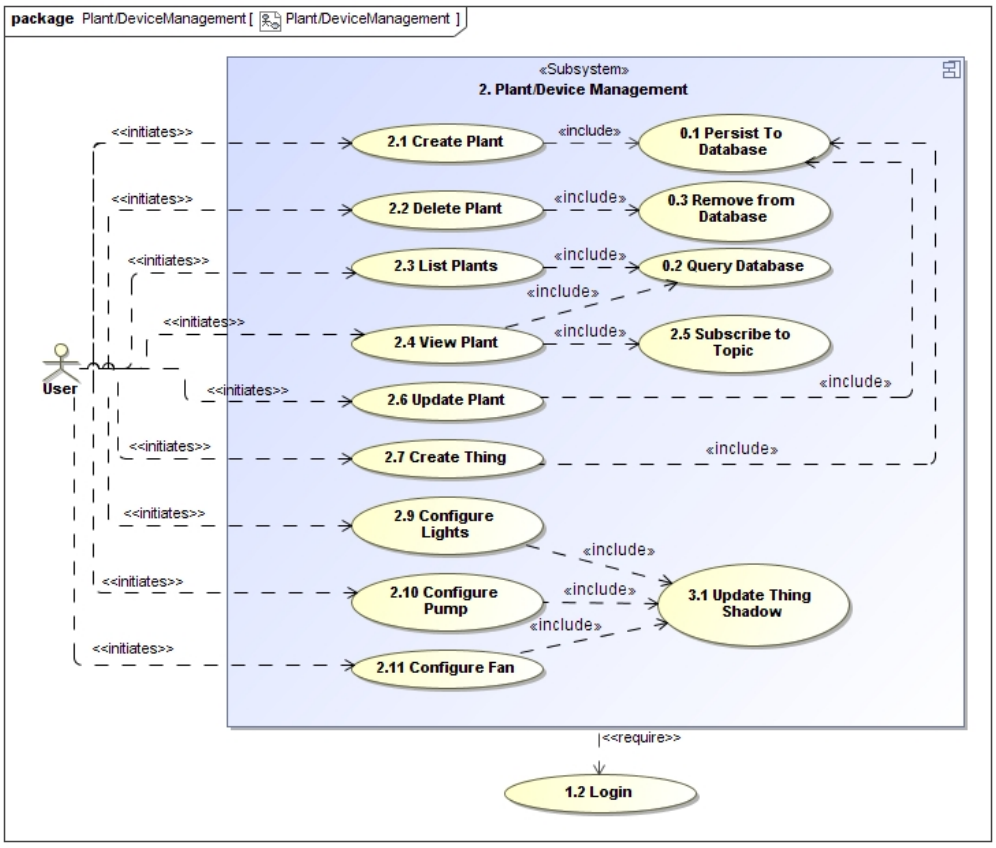

Figure 8: Use Case Diagram for the Plant/Device Management Subsystem

5.2.2.1 Create Plant

Description: A registered and logged in user should be able to create a virtual plant to be associated with a device monitoring a plant. This plant should have a name as well as a plant type.
Prioritisation: Critical
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

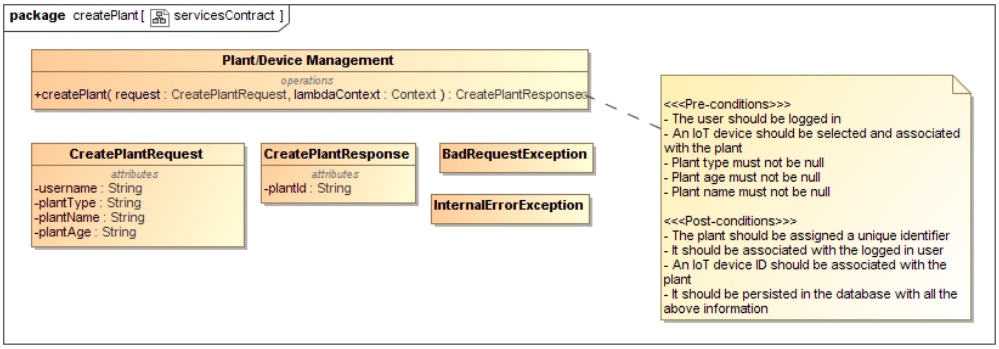

Figure 9: Services contract diagram for create plant

5.2.2.2 List Plants

Description: A registered and logged in user should be able to retrieve a list of all the plants associated with their account.
Prioritisation: Critical
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

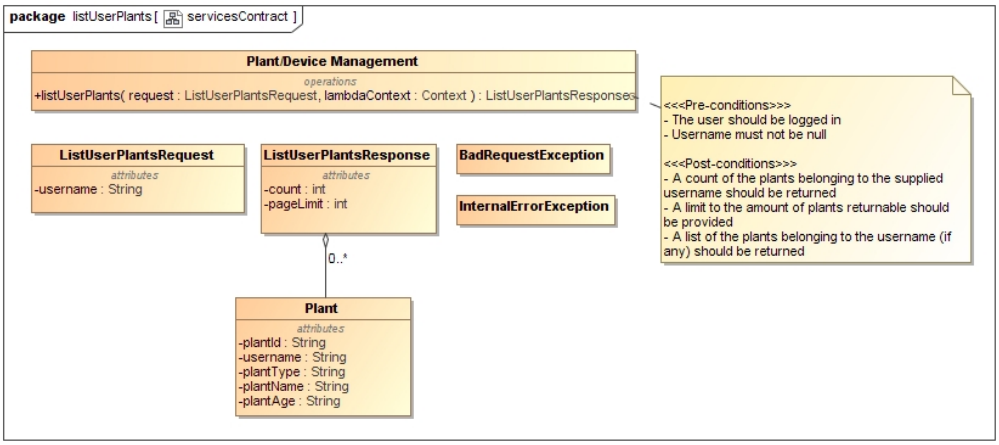

Figure 10: Services contract diagram for list plants

5.2.2.3 Create and Associate a Virtual 'Thing' With a Plant

Description: A registered and logged in user should be able to create a virtual 'Thing' and associate it with a particular plant.
Prioritisation: Critical
Pre-conditions:

- The user should be logged in
- A plant ID should be supplied
- The virtual 'Thing' should be associated with a particular plant

Post-conditions:

- A newly created 'Thing' should be created
- A relationship between the 'Thing' and a plant should be created and persisted in the database

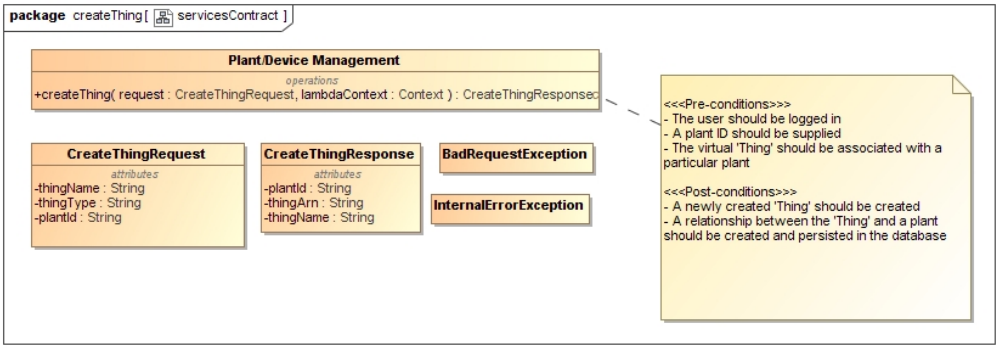

Figure 11: Services contract diagram for create and associate a virtual 'Thing' with a plant

5.2.2.4 View Plant Status

Description: A registered and logged in user should be able to view the details associated with a particular plant in terms of the readings the associated IoT device has stored in the database.
Prioritisation: Critical
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

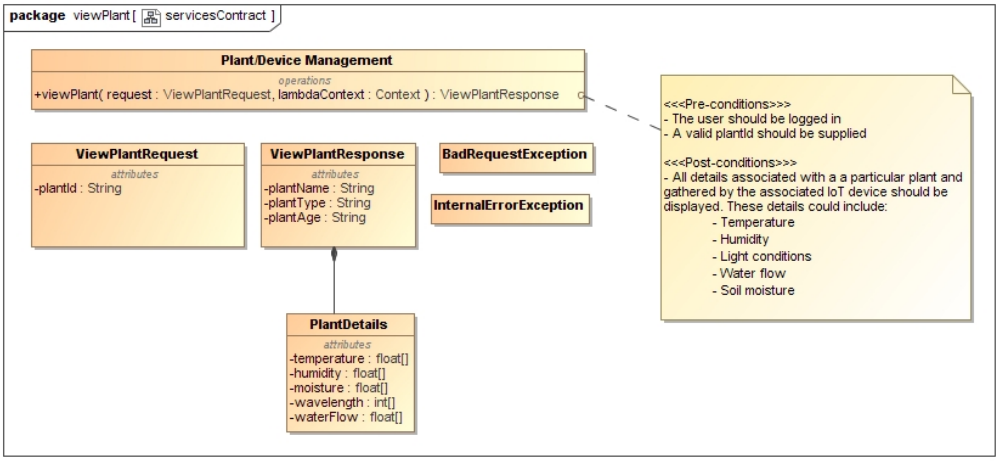

Figure 12: Services contract diagram for view plant status

5.2.2.5 Configure Light Settings

Description: A registered and logged in user should be able to adjust the RGB values of the LED strip associated with their IoT device via the frontend interface.
Prioritisation: Critical
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

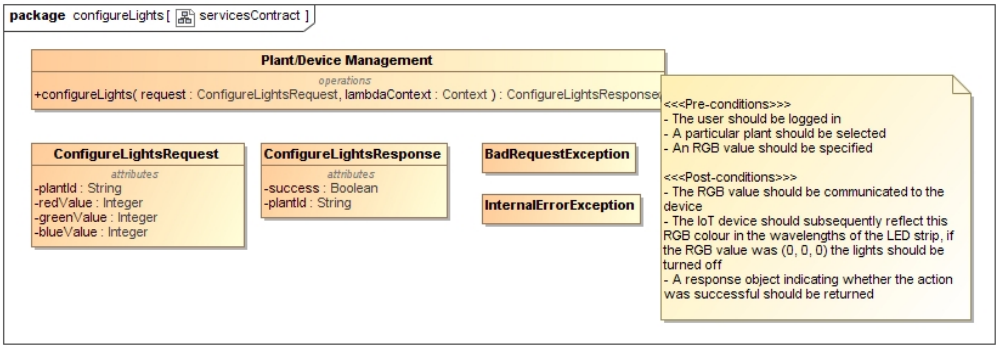

Figure 13: Services contract diagram for configure light settings

5.2.2.6 Update Plant Details

Description: A registered and logged in user should be able to edit the name and type of a particular plant.
Prioritisation: Important
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

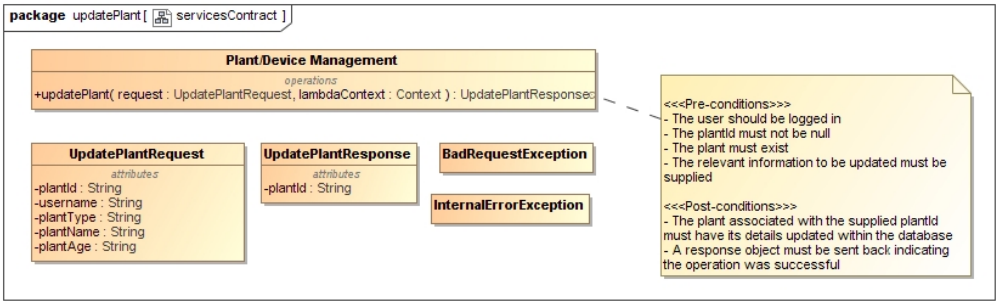

Figure 14: Services contract diagram for update plant details

5.2.2.7 Configure Pump Settings

Description: A registered and logged in user should be able to configure the settings of the water pump associated with an IoT device.
Prioritisation: Important
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

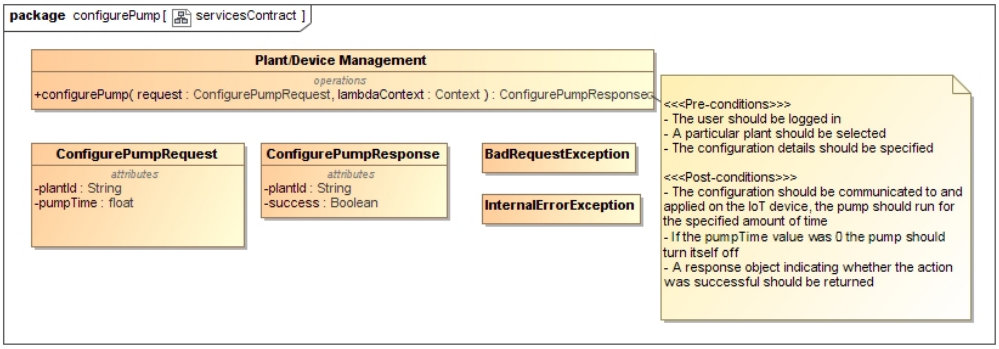

Figure 15: Services contract diagram for configure pump settings

5.2.2.8 Configure Fan Speeds

Description: A registered and logged in user should be able to configure the settings of the fan associated with an IoT device.
Prioritisation: Important
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

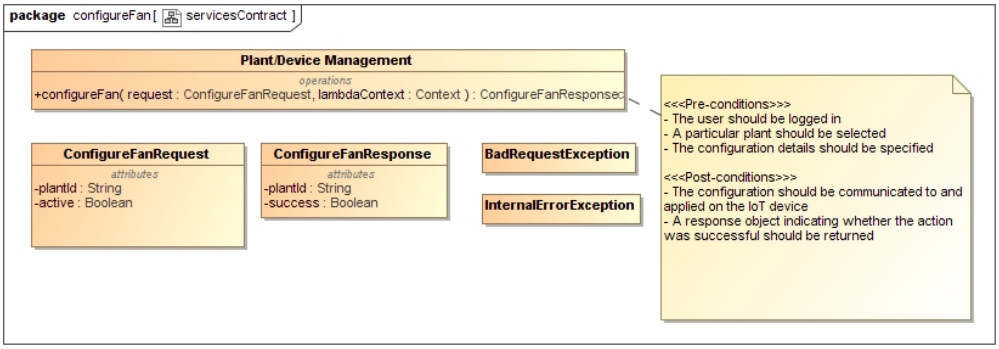

Figure 16: Services contract diagram for configure fan settings

5.2.2.9 Delete Plant

Description: A registered and logged in user should be able to remove a plant associated with their account.
Prioritisation: Important
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

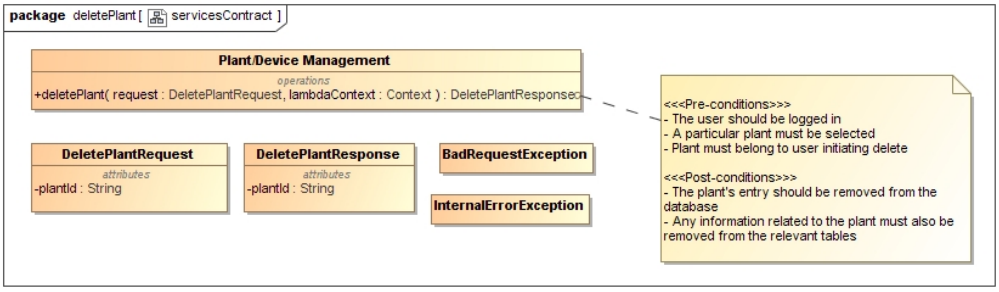

Figure 17: Services contract diagram for delete plant

5.2.3 Device Communications Subsystem

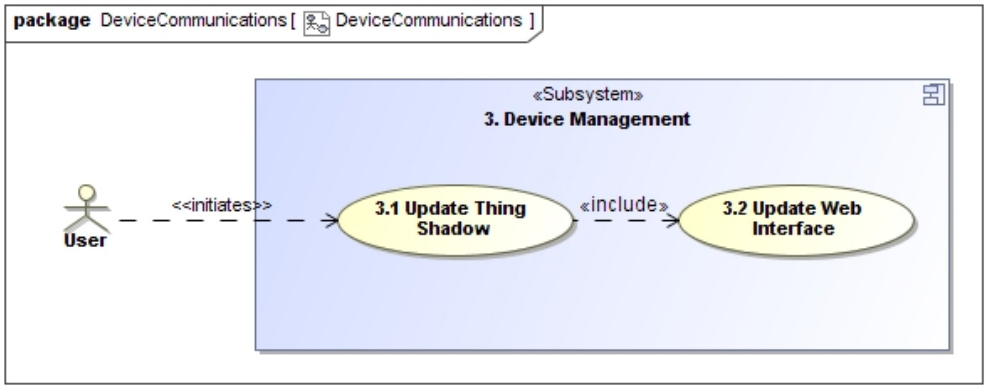

Figure 18: Use Case Diagram for the Device Communications Subsystem

5.2.3.1 Send Readings Summary to Lambda

Description: An IoT device should send details about the information it has gathered using its sensors to the serverless provider.
Prioritisation: Critical
Pre-conditions:

- A connection should be established with the IoT platform
- The device should publish to a particular topic

Post-conditions:

- The information gathered should be stored in the database and associated with the plant associated with the IoT device

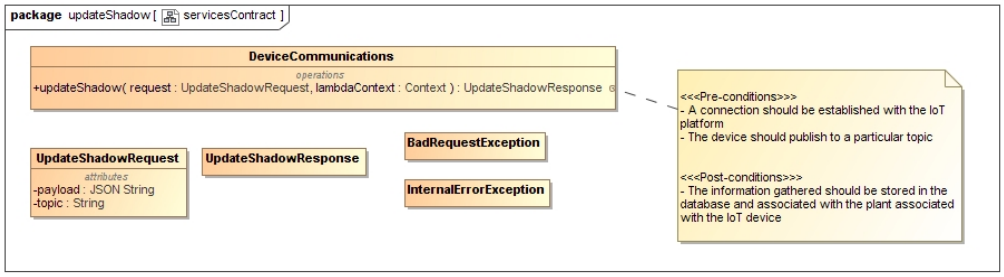

Figure 19: Services contract diagram for send readings summary to Lambda

5.2.3.2 Complex Event Processing on Device Communications

Description: An IoT device's readings should be intercepted and analysed to determine whether they indicate a complex event has occurred.
Prioritisation: Critical
Pre-conditions:

- A connection should be established with the IoT platform
- The device should publish to a particular topic
- The information should be intercepted and analysed in real time

Post-conditions:

- According to the event the information indicates the appropriate automated action should take place in terms of controlling some aspect of the device.

6 Architecture Requirements

6.1 Access Channel Requirements

6.1.1 Human Access Channels

Users will be able to register and log into a Web interface. From the Web interface, they can associate their own Plant Boxes, monitor conditions such as temperature and humidity, control the Plant Box conditions and view their Achievements. The Web interface should be accessible from any modern browser, such as Google Chrome, Mozilla Firefox, Microsoft Edge, Apple Safari and latest Internet Explorer. The Web interface will be designed under strict UX, UI and World Wide Web Consortium (W3C) standards. This includes a responsive design that will work on any screen size. Plant Boxes do not need to be connected to the Internet at all times - the Web interface will store the latest settings and sync them with the Plant Boxes when a connection can be established.

6.1.2 System Access Channels

AWS API Gateway will be used expose the system's API and map it to the serverless backend provided by AWS Lambda. MQTT messages will be used to communicate between the system backend and the individual Plant Boxes. MQTT is an ISO standard (ISO/IEC PRF 20922) publish-subscribe-based "lightweight" messaging protocol for use on top of the TCP/IP protocol. It is often used to facilitate IoT communications.

6.2 Quality Requirements

6.2.1 Usability

Because our system might be used by persons not familiar with complex systems, the user interface should be intuitive and efficient to use. Users with basic computer literacy should be able to easily use the core functionality of the system with little to no extra training required. Beyond the user interface, the user manual should be clear and well laid out in order to assist with learning of the system. To further ease of use, input validation should be done on the client side as much as possible to ensure that validation is fast and does not require a server response.

Error messages generated by the system while in use should be clear and informative, helping the user understand and fix any issues that may have occurred. The messages should not be too technical in nature and attempt to provide a solution to the error.

In order ensure usability is high:

- usability tests with non-technically inclined people.

6.2.2 Scalability

It is important that our system be scalable in order to ensure that the system can handle a very high number of users as well as all their devices. By using cloud computing, the system will grow and shrink as required by the users, providing a smooth experience no matter how many users or devices are connecting to the system. We must be able to handle a minimum of 10 000 concurrent users without noticeable latency issues.

- Processing power scales dynamically based on demand
- Storage capacity scales dynamically as items are added and removed
- Database read and write speeds scale dynamically

6.2.3 Flexibility

It is important that the system has support for many hardware components such as sensors. These technologies are advancing rapidly, therefore it should be easy to add new components to the system, without making any large changes. A user should be able to add/remove any number of devices from their account, as well as add/remove any number of sensors from each device without any affect on the system.

In order to fulfil these requirements, the following tactics will be used:

- Hot pluggable hardware
- Two-way binding of real-time updates between Web interface and hardware
- Contract-based-development, which allows easy pluggability of software components via interfaces

6.2.4 Auditability

The system will have a very high level of monitoring and audit control. The state of the devices is constantly polled and a historical record is kept of every device. The system can keep access and event logs for later auditing. All system calls will be logged and log files will be kept for a minimum of 5 years. After 5 years, old logs will be summarized

and only the most vital information (such as user creation and user detail editing) will be kept for auditing purposes.

- Logging of user activity and resource usage
- Public access to source code and version control, which also provides transparency to users and other developers
- Cloud-based storage of log files to ensure longevity

6.2.5 Integrability

The three main parts of the system (web interface, server and devices) are highly integrable between each other. The web interface communicates with the server and devices through exposed RESTful services, allowing it to be independent and interact with any other system so long as the systems use the same RESTful calls.

The device communicates with the server and the web interface through the MQTT protocol and topics, which allows it to be decoupled from the server. Any server system that can receive and send MQTT messages can be coupled with the device subsystem.

The server consists of multiple services that cannot be decoupled from one another as they all use the same SDK and are designed to be used in unison.

6.3 Integration Requirements

- **HTTP**

  This is the foundation of data communication for the World Wide Web. It will be used to serve the web pages to the users via a web browser.

- **REST**

  The system's RESTful API will be exposed via AWS API Gateway. This will map API calls to the serverless backend, AWS Lambda.

- **TCP**

  The TCP protocol will be used to establish network communications between the users' computers and the system's backend, as well as between the system's backend and the users' Plant Boxes.

- **MQTT**

  MQTT is a lightweight messaging protocol that will be used as a way to facilitate two-way communication between the backend and the Plant Boxes, via TCP as described above.

7 Architecture Constraints

The following architecture constraints will be applicable:

- **Programming Languages:**
  - The backend must be programmed using Java
  - The Arduino devices must be programmed using Processing
  - The frontend must be programmed using JavaScript
- **Web Services:**
  - AWS IoT hardware platform
  - AWS cloud services

8 Architecture Design

8.1 Infrastructure

We will be using a cloud-based, serverless, event-driven micro-services architecture. By using a micro-services approach, the system will be able to develop different parts of the system concurrently, due to the loose coupling between components. This also allows more efficient unit testing because different parts of the system can be mocked independently.

8.2 Services

8.2.1 Data gathering service

Figure 20 shows the components of the data gathering service. The Plant Box consists of the plants, sensors and the physical IoT device. The sensors send readings to the IoT device every 10 seconds, which in turn updates its virtual representation on the AWS IoT platform.

8.2.2 Processing service

After the data has been gathered, it will be processed by means of calculating necessary statistics over a set time period. For example, every 12 hours the max, min and average readings from the temperature sensor are calculated and stored using the Database service (see section 8.2.4). This is shown at a high level in Figure 21

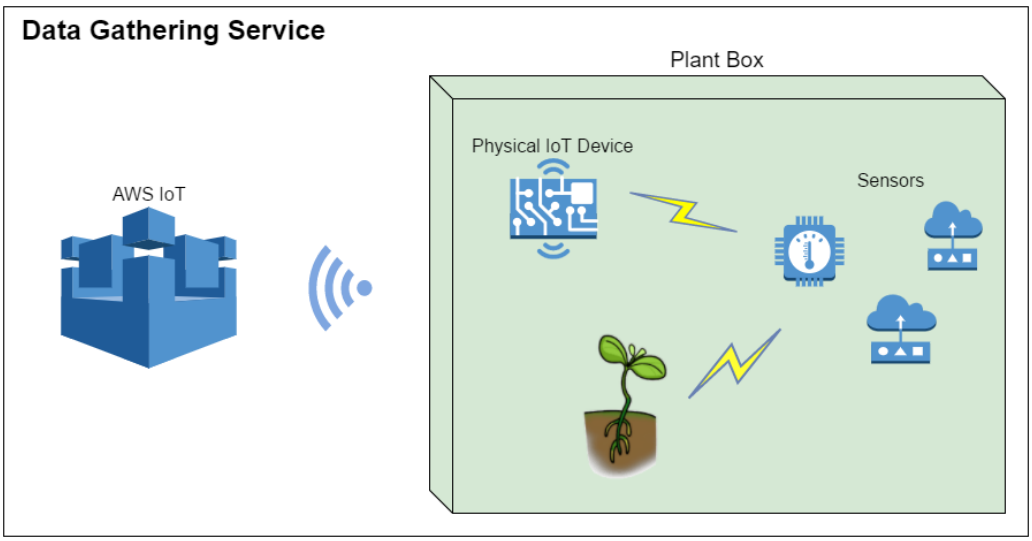

Figure 20: Data Gathering Service

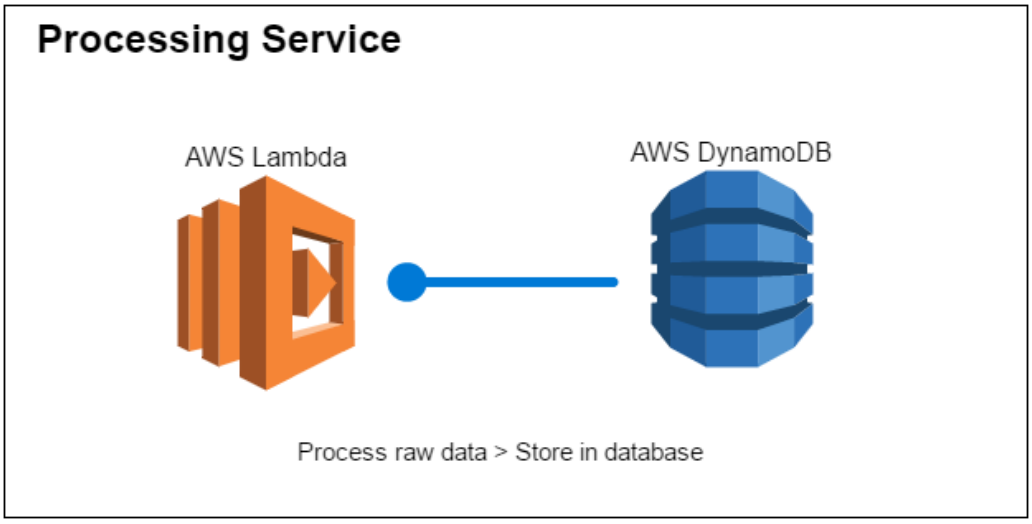

Figure 21: Processing Service

8.2.3 Business service

The Business Service, shown in Figure 22, handles all user interaction and delegates it to other services. This includes all Web interface functionality such as graphing, gamification and plant management. AWS API Gateway is used to map the Business service to the

frontend.

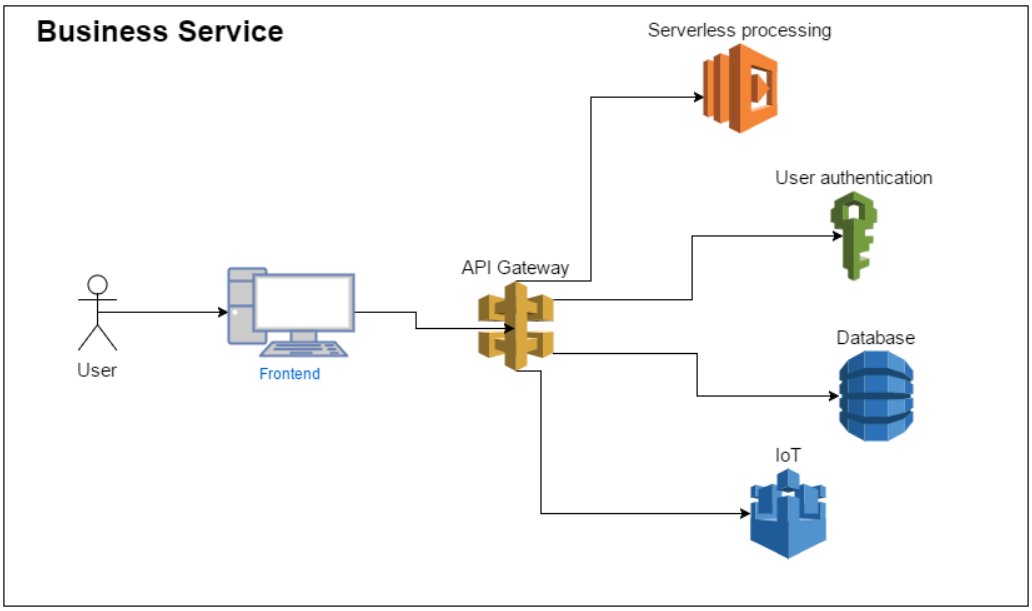

Figure 22: Business Service

8.2.4 Database service

The Database Service will use the DynamoDB persistence API to persist user information and Achievements as well as plant statistics. This is illustrated at a high level in Figure 23

9 Database and Persistence

The database which the system is using is AWS DynamoDB. It is a managed, NoSQL database service. Some of the reasons we decided to choose a managed, NoSQL database service include, but are not limited to:

- Database operations are able to be triggered and are able to trigger other cloud services based on particular events. This suits our event-driven architecture well
- Automatic data replication over three availability-zones in a single region which ensures fast and redundant access

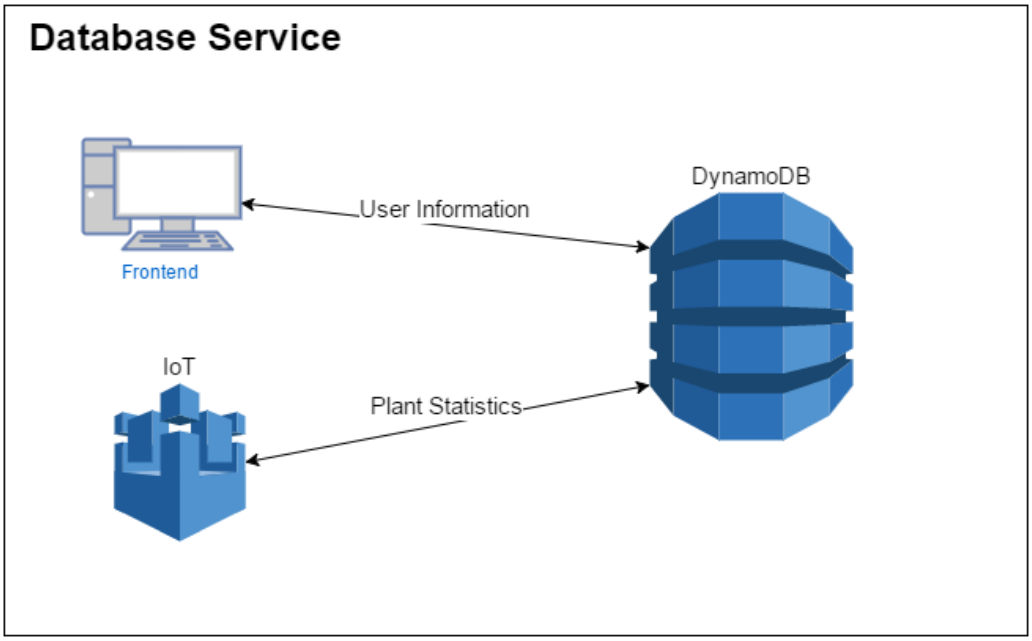

Figure 23: Database Service

- It obeys a pay-per-use model
- Through the use of other AWS services security and access control is easy to manage
- ”Infinitely” scalable read/write I/O running on IOPS-optimised solid state drives

10 Process Specification

In this section, there is an overview of the three main processes of the system, where each process is explained in a broad manner.

10.1 Backend

The backend consists of three main parts, mainly the database, the events processor and the Internet of Things (IoT) manager.

10.1.1 Database

The database stores the user details, which IoT device is associated with which account, as well as all the details sent by the IoT device in the updates (explained below) into the corresponding sensor detail tables. When a sensor table reaches a certain capacity, the database will trigger the events processor to collect and concatenate data into a more summarized form: hourly data is transformed into daily data, daily into monthly etc.

10.1.2 Events Processor

Instead of the server running continuously, waiting for input, the main functions of the system are triggered by certain events. This means that the server uses a lot less processing power as it does not run when idle and it can handle simultaneous parallel requests without any explicit multiprocessing.

The events that trigger the server include messages sent from the device through IoT and any API RESTfull service calls such as registering a user, adding a new thing or editing plant details.

10.1.3 IoT Manager

In order to control the MQTT messages and Internet of Things devices in our system, is using an IoT manager. The manager authenticates all of the devices (things), controls all of the IoT topics that the things publish to, any rules to be processed on the incoming MQTT messages etc.

10.2 Frontend

By using the web interface, the user can add plants and devices to their account. Once added, they can view a live stream of the status of the plant, graphing the sensor readings. They can also use the interface to update their desired configuration settings for the plant, such as light colours.

10.3 IoT Device

The IoT devices are connected to sensors that take in readings. The sensor readings are added to JSON strings and sent via MQTT to the IoT manager periodically. The device also receives updates from the IoT manager in the form of shadow updates which will tell the IoT device what configuration to save the configurable devices in (e.g. light colours).

11 Technologies

11.1 Overview

The system uses the Amazon WebServices SDK and APIs.

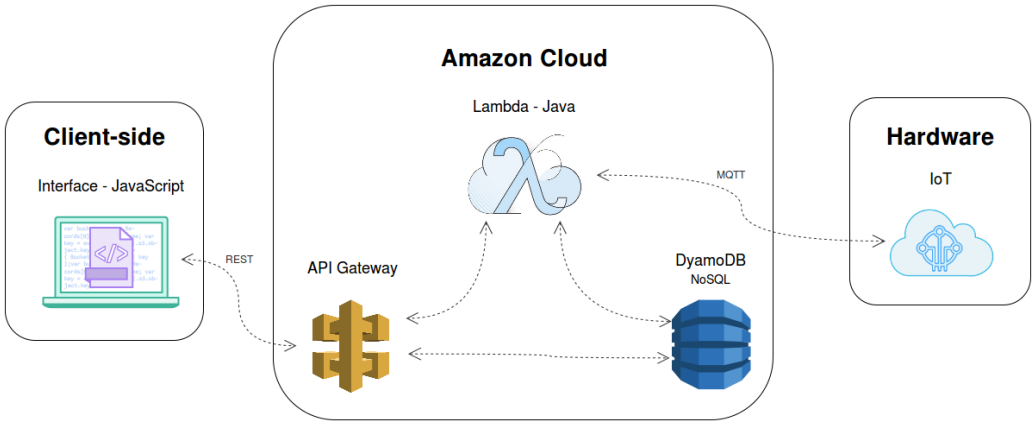

Figure 24: Communication between AWS services

11.2 Frameworks

11.2.1 Internet of Things

AWS IoT is a managed cloud platform that lets connected devices easily and securely interact with cloud applications and other devices.

The system uses the IoT platform to connect to Lambda (explained below) using the MQTT protocol. As part of the project, the system is required to use the AWS IoT platform.

11.2.2 Lambda

AWS Lambda lets the system run code without provisioning or managing servers. You pay only for the compute time you consume - there is no charge when your code is not running. With Lambda, you can run code for virtually any type of application or backend service - all with zero administration. Code can just be uploaded and Lambda takes care of everything required to run and scale the code with high availability. The code can be

set up to automatically trigger from other AWS services or to be called directly from any web or mobile app.

Lambda will form the largest part of the backend. All messages will be sent from IoT to Lambda, using MQTT, and that event will trigger a response and the web interface can access Lambda through the API Gateway, negating the need for a server to be constantly listening. Lambda can then perform operations on DynamoDB, but can also respond to events triggered when data in DynamoDB is changed or added.

11.2.3 DynamoDB

Amazon DynamoDB is a fast and flexible NoSQL database service which has built in Java support. Because it is a NoSQL database, Java objects can be pushed or pulled using the API. It intergrates with AWS Lambda, allowing Lambda to respond to events in the database. It can also be easily accessed through the API Gateway, allowing for fast read-write operations from the client interface.

11.2.4 API Gateway

Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale. The system can easily and quickly create an API that acts as a front door for the user interface to access data from DynamoDB or trigger events in Lambda by exposing RESTful services to the API. Amazon API Gateway handles all the tasks involved in accepting and processing up to hundreds of thousands of concurrent API calls, including traffic management, authorization and access control, monitoring, and API version management.

11.2.5 JavaScript SDK

The AWS SDK for JavaScript enables the system to directly access AWS services from JavaScript code running in the browser. The system can authenticate users, store data in DynamoDB and trigger Lambda events. It is simple to deploy and uses the same API as the other components, meaning function migration becomes trivial.

11.2.6 CloudWatch

CloudWatch is a monitoring device for all AWS services. The project uses this to log access to the API Gateway, IoT, and Lambda functions, so that a historical audit of all

connections is kept. CloudWatch also has an alarm service, which can trigger custom events (e.g. a Lambda function fails)

11.2.7 Cognito

Cognito is used for user authentication and to manage user pools.

11.3 Build Tool and Continuous Integration

Travis CI is used for the build and continuous integration pipeline. As code is pushed through version control, Travis CI pulls the latest code, runs all tests and if successful, deploys the code live.

11.4 Languages

For the frontend:

- HTML5, JQuery, Bootstrap
- AngularJS

For the backend:

- Java 8
- NodeJS for mocking of devices

For the hardware:

- Processing for Arduino

12 Initial Design

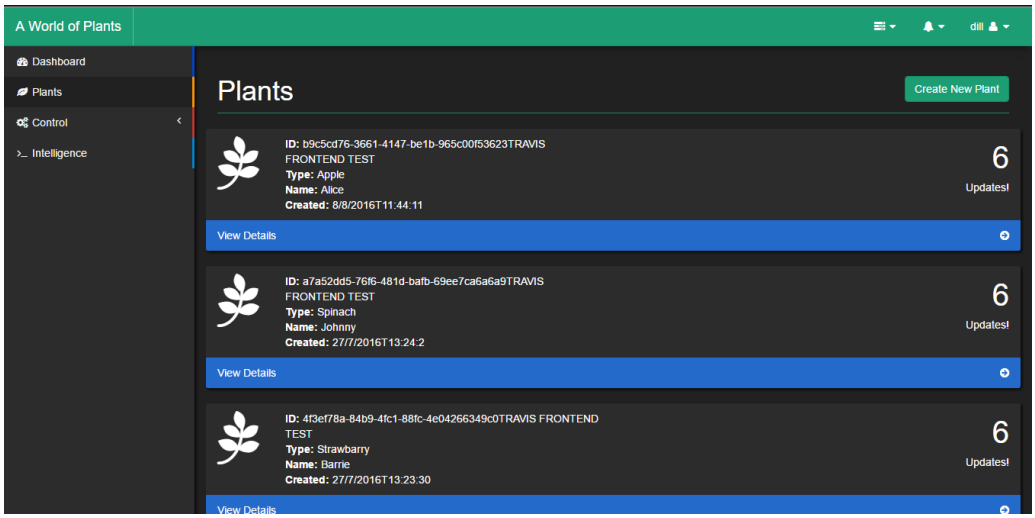

Figure 25: Initial Design of Main Interface

The initial design of the plant management section of the Web interface. Website navigation appears on the left sidebar. Notifications, Achievements and user actions appear on the top navbar. In the main section, users can add a plant and view a list of their current plants with a short overview.

13 References

[1] Internet of Things Global Standards Initiative. 2016. IOT. [ONLINE] Available at: http://www.itu.int/en/ITU-T/gsi/iot/Pages/default.aspx. [Accessed 1 September 2016].