Visualising plants and metadata

Final Report for CS39440 Major Project

Author: Siôn Griffiths (sig2@aber.ac.uk)
Supervisor: Dr. Hannah Dee (hmd1@aber.ac.uk)

18th of April 2016
Version: 1.1 (Draft)

This report was submitted as partial fulfilment of a BEng degree in
Software Engineering (G600)

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB
Wales, UK

Declaration of originality

I confirm that:

- This submission is my own work, except where clearly indicated.

- I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.

- I have read the regulations on Unacceptable Academic Practice from the University's Academic Quality and Records Office (AQRO) and the relevant sections of the current Student Handbook of the Department of Computer Science.

- In submitting this work I understand and agree to abide by the University's regulations governing these issues.

Name ....

Date ....

Consent to share this work

By including my name below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name ....

Date ....

Acknowledgements

Abstract

Visualising plants and metadata is a project delivering a web-based system which enables the convenient exploration of plant images and associated metadata captured as part of experiments carried out at the National Plant Phenomics Centre(NPPC).

CONTENTS

LIST OF FIGURES

LIST OF TABLES

List of Listings

Chapter 1

Background & Objectives

1.1 Background

1.1.1 Introduction

This project uses data and images collected during the course of experiments conducted at the National Plant Phenomics Centre (NPPC) [21]. The NPPC, based near Aberystwyth, houses a state-of-the-art, automated greenhouse and imaging system. During experiments, plants are housed on moving conveyors and are carried one by one through measurement chambers where images of various modalities (such as infra red and visible light) are captured from multiple angles. Physical and environmental measurements, such as plant weight, water usage or greenhouse temperatures, can also be captured automatically. More specific or specialist data (such as targeted phenotype or genotype traits) are captured following observation by staff at the facility.

The experiments at the NPPC are capable of investigating large sets of plants including whole breeding populations in order to inform how physical characteristics are affected by genes. Phenotyping experiments will often conduct a quantitative trait locus (QTL) analysis where sections of DNA corresponding to certain desirable phenotypes are identified, mapped and recorded for consideration in future breeding. The NPPC is capable of supporting a wide range of plants, from food-security critical cereal crops such as wheat and oats to plants that are promising sources of bio-fuel or biomass such as Miscanthus. The results of these experiments can directly affect the yield and robustness of future generations of important crops.

There is a wealth of data collected at the NPPC that is often never considered again following the conclusion of an experiment. This project seeks to provide an ongoing use for some of these data and images.

1.1.2 Initial project topic

The initial title for this project was 'Building a plant Atlas from real images'. In this context an Atlas is a term used to describe a visual reference for developmental stages within a subject of interest. Such an approach is commonly attributed to Tanner et al [25] for their work at numerically scoring the stages of bone development within the hands of infants and providing a visual reference for each stage.

Biologists use numerical growth stage indices to chart the developmental milestones in the life cycle of a given plant. In cereals a popular scale for these growth stages is the 0 to 100 scale defined by Zadoks [27] in 1974. These scales are often presented in an Atlas style with hand drawn, stylised images used to display detail of the characteristics of a particular growth stage. Figure 1.1 shows an example taken from the BBCH scale which is based on the Zadoks cereal scale.

Skala BBCH
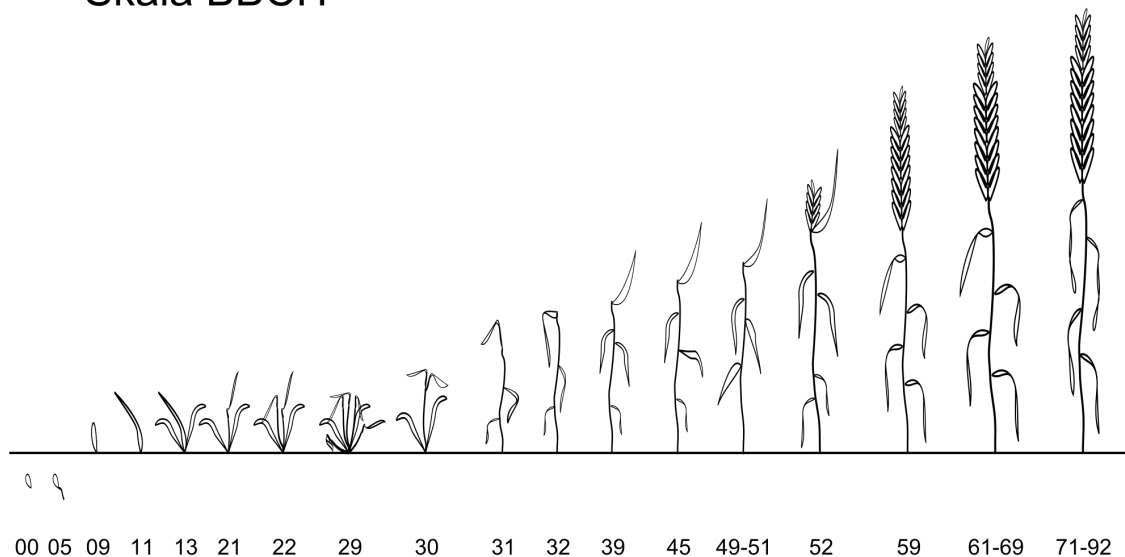
Figure 1.1: Example of 'atlas' style visualisation of wheat plant development stages. Source : Wikimedia Commons [17]

The aim of the original project was to utilise the images and data collected during a particular experiment at the NPPC in order to provide an atlas style visualisation using real plant images and providing a web based interface onto this visualisation with the intention of it being used as a reference for the biologists or a teaching aid and to sort or align the experiment population on growth stage and compare their physical characteristics. Further suggested work on this topic was to leverage some machine learning capability to attempt to draw conclusions or infer useful correlations from the datasets collected over the course of running experiments.

This topic was selected for the subject of this dissertation since it provided a chance at building a system that may have some practical use and application after the dissertation was complete. The topic also provided the opportunity to learn about certain plants, their life cycles and their importance as a research subject, this seemed like a more interesting problem domain when compared to other available choices.

1.1.3 Change of project topic

In the initial weeks of the project, meetings were arranged with Dr Roger Boyle, a researcher specialising in computer vision based applications at the NPPC. The purpose of these meetings was to discuss the direction and background of the originally proposed project and arrange visits to the NPPC itself in order to gain an understanding of the facility and its functions. Spike work, proto-

typing and research into plants, growth stages, atlases and investigations into agreement between expert opinions [26] were the focus of these early weeks as opposed to investigations into the data being collected at the NPPC.

In order for the original topic to be successful, the recording of key growth stage information during the course of an experiment was necessary, without these data points it would not be possible to build the proposed atlas visualisations. Unfortunately, where it was assumed that such data was being recorded as a matter of course during NPPC experiments, it became apparent that the recording of plant characteristics data during experiments was extremely sparse. When experiment data was provided for analysis it became clear that the approach to data collection was very different from what was expected.

The interdisciplinary differences in attitude to data between the computer scientists and biologists involved was a key factor in the erroneous assumptions regarding the quantity and quality of collected data. The biologists who collect experiment data are often concerned only with growth stages which are directly associated with traits that are being targeted as part of the experiment and from the data analysed are content with error margins of up to three days.

With only a handful of growth stages captured it became clear that the original project topic would not be possible. Following this discovery a meeting was arranged with various staff at the NPPC including the director, Professor John Doonan, in order to discuss what data was available and what would be a useful project. The outcome of this meeting was a new project topic, a system would be implemented that would make use of available images and data for a given experiment and present it in an easy to navigate fashion. The produced system should also natively support the means to add additional data to an experiment such that a more complete dataset could potentially be achieved and saved in a web-accessible database.

1.1.4 Existing solutions

Two alternative solutions were investigated during the discovery stages of this project. The first is the current means for experiment images to be viewed at the NPPC. Figure 1.2 shows an excerpt from an internal webpage hosted by the NPPC that provides access to the plant images associated with a particular experiment. Here we can see the available image modalities and can select between the various image angles. Following one of these links will display the entire time series(days) of captured images for the plant corresponding to the chosen modality and angle. Whilst this system allows browsing of the images captured it does not allow a simple way to compare or switch between the different modalities of images that correspond to the same time (day) in the life of the plant. No means to view or add associated plant data is available in this solution.
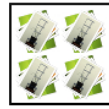
Figure 1.2: Current means of NPPC image exploration

The second existing solution investigated is a commercial product called Zegami [11] developed in part by Oxford University. Figure 1.3 shows the Zegami system in action on the Australian Plant Phenomics Facility [2] website. Zegami is a web based tool that allows the browsing of large amounts of images in a fairly intuitive and responsive way and also provides the means to search, sort and filter images using associated data attributes. Zegami allows the visualisation of data in graphical formats and provides the means to select subsections of the images based on selections made in the graphical views via drawing boxes or circles around the desired datapoints.

Being positioned as a image and data exploration tool, data addition is mostly done from a single datasource file in Zegami with no out-of-the-box facility for users to add supplementary data to an experiment.

Zegami is a feature-rich and robust solution for exploring large collections of images and associated data, however, being a commercial venture it is not free and can be considered prohibitively expensive a solution for a project of this nature.
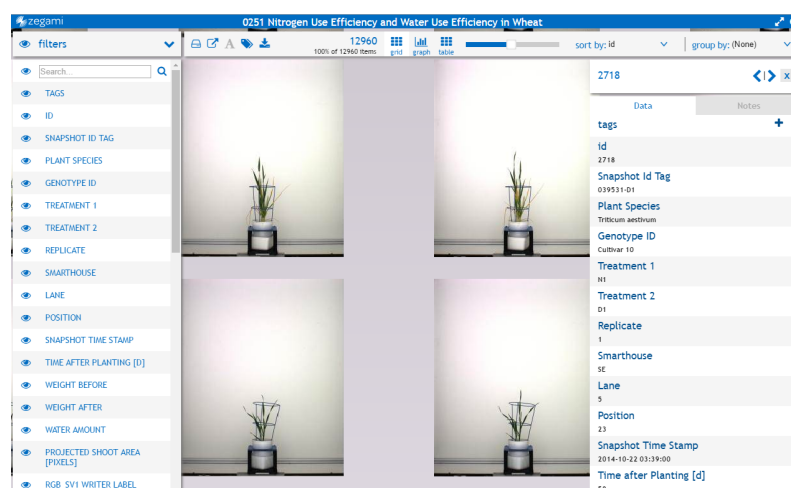
Figure 1.3: Zegami system as used by the Australian Plant Phenomics Facility. Source : `https://zegami.plantphenomics.org.au`

1.2 Analysis

The core problem that this project aims to solve is the need for a web-based system that enables the exploration of images and associated data that are collected during experiments at the NPPC.

Following the domain research and familiarisation with the NPPC gained during the discovery period for the initial project specification it was felt that the new direction of the project was well understood and that certain requirements could be identified fairly quickly. The following list will detail these and further additional requirements identified during the course of the project to provide a complete view of targeted functional specifications.

1.2.1 Requirements decomposition

1.2.1.1 Web based system

In order to provide a practical solution to the problem, access to the system should be available from a wide array of devices and systems. The system should also be accessible from a variety of different locations. The most practical solution to these considerations is a web based approach. Most devices and systems natively support some kind of web access and a centrally hosted solution is far more accessible than any alternative approach.

1.2.1.2 Integrate with NPPC data repository

The system needs to display plants, plant images and associated data. Data and images captured at the NPPC during experiments are stored in a central data repository hosted on the Aberystwyth University network. In order to provide access to these images and data via a web based approach, it is necessary to integrate the system with the data repository such that the images or data can be pulled directly from the repository as opposed to re-hosting the same content within the delivered

system itself. The sheer quantity of image data in the repository itself makes a re-hosting solution highly impractical.

1.2.1.3 Browse plants and plant images

Users will be able to view plants, associated data and plant images. In order to provide a means of exploring the data and images captured at the NPPC, the system needs to provide some interface onto these data that allows easy navigation between plants, images and associated data. Currently there is no consolidated interface onto both experiment images and the data observations captured during the course of experiments. Providing such an interface makes the exploration of past and current experiments convenient and simple.

1.2.1.4 Import experiment data

Experiment data will be imported into the system from file and associated with the plants in an experiment. The biologists conducting experiments will invariably use spreadsheets in order to capture observations and data on plants in the experiments. The system needs to be able to incorporate these data with minimal manual processing such that the system can be used to link these data to the plant images.

1.2.1.5 Add meta data to plants and plant images

Users will be able to add supplementary data to individual plants and images. In order to provide the opportunity to supplement experiment data with further information, a facility to add data to the system is required. The addition of data in this manner allows the creation of richer, more complete datasets for a given experiment with potential for future use in other applications and analyses.

1.2.1.6 Persist data in local database

Image data is sourced directly from the NPPC data repository. Data imported or input into the system needs to be stored in a persistent way. Access granted to the NPPC data repository is read only for security and data integrity purposes. A database local to the delivered system provides a solution that allows fast queries of contained data, avoiding any network over heads, and provides the means to store experiment and supplementary data. Exports can potentially be taken from the database in order to migrate the data to a separate system or facilitate its use as a dataset for future work.

1.2.1.7 Display graphs of data in system

Visualising the data within the system is key to facilitating the quick understanding and digestion of captured data. Using graphical visualisations provides the means of quickly comparing certain data attributes or deducing whether there are interesting correlations or outliers in the data. Graphical visualisations can be used in order to quickly determine whether further statistical analysis of subsets of captured data is worthwhile or likely to produce interesting results. Providing the means

to quickly create arbitrary visualisations from the experiment data allows the experimenters to see the data in ways that would previously require a much larger degree of manual effort.

1.2.1.8 Administrator panel or page to manage the system

A means of managing the experiments in the system will be provided via a web page with restricted access. Providing the means to control the enabled experiments and data within the system is vital for an easy to use and efficient solution.

1.2.2 User Roles

For this system there were two identified user roles, an administrator or admin role and a general user role.

1.2.2.1 Admin

Figure 1.4 shows a use case diagram for the admin user. The admin manages experiments in the system with the facility to initialise new experiments or delete previously initialised experiments. The admin user can also manage the data associated with an experiment by importing experiment data from file or resetting the data associated with the experiment back to its newly initialised state.

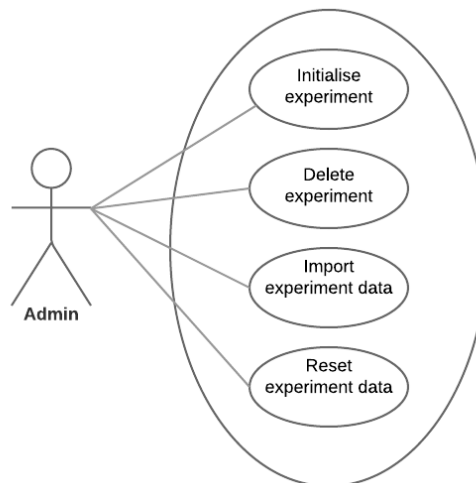

Figure 1.4: Use cases for admin user

1.2.2.2 User

Figure 1.5 shows the use cases for the general user role. Users of the system are able to select between the various available experiments and browse the plants and images associated with them. Users are able to add data to plants in the form of tags or key value pair attributes. Users are able to generate graphs from the available data.
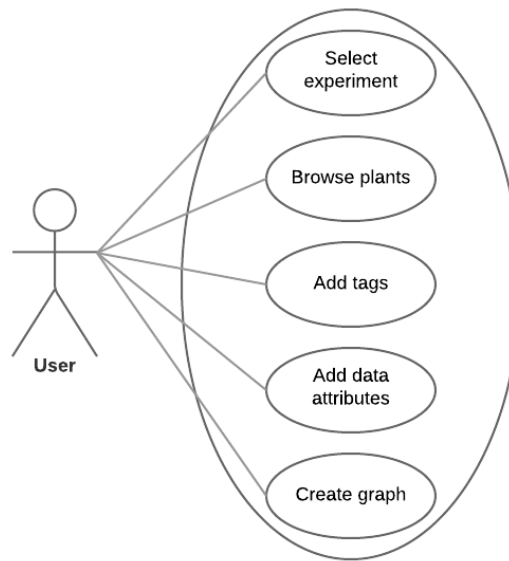
Figure 1.5: Use cases for generic user

1.3   Process

Plan driven approaches traditionally associated with software development projects usually expect that all system requirements are understood and collected prior to any further work on design or implementation. A number of factors made such an approach unsuitable for this project, chiefly a lack of domain knowledge made up-front requirement gathering difficult and the requirements themselves were likely to be poorly defined and subject to change.

The methodology used for the delivery of the project was an agile approach based on the popular SCRUM methodology. For the duration of the project, work would be carried out in time-boxed iterations or 'sprints', each a week long. Sprints would begin with a planning session and end with a release of the system software. At the conclusion of each sprint a short retrospective analysis of the sprint would take place, looking at what went well and what could improve for the next iteration. The focus on incremental delivery of working software allowed the project to evolve in an emergent fashion whilst remaining continuously functional as features were prototyped, designed and implemented.

System requirements were broken down into user stories which in turn were broken down into individual tasks if necessary. As in SCRUM, these stories were held in a backlog until being added into a current or future sprint depending on priority and the goal for a particular sprint. Emergent issues such as priority bugs could easily be incorporated into the wider context of the current sprint if necessary which allowed work to be focused on the most pressing issues. The Jira [13] issue tracking application was used in support of this process providing an environemnt in which to specify and track user stories,task and sprints, see section 2.7.3 for further details.

During a sprint, each day would begin with a quick overview of tasks in the sprint, replicating the 'stand-up' meetings common in SCRUM. Work would be commenced or continued on the task deemed highest priority at the time. At the end of each day a short update would often be posted on the project blog available at `https://siongriffithsblog.wordpress.`

`com/` summarising the days activity. The blog itself has proven to be a useful part of the process, helping to document certain aspects of the implementation and design that may have otherwise been forgotten.

1.3.1   Time Management

Effective management of time is a key consideration of any reasonable development process model. Partway through the project it was decided to fully adopt the Pomodoro technique [15], working in blocks of twenty five minutes with complete focus on the task at hand, referred to as Pomodoros. A five minute break is taken after each successful twenty five minute work block in order to avoid the mental fatigue of attempting to remain focused and productive for an extended amount of time. Taking these regular, short breaks allowed for a higher degree of productivity over the course of a work day.

Having distinct blocks of time in which to complete work compliments the SCRUM approach to effort tracking and estimation. Although not part of the initial process, towards the latter half of the project after gaining a sufficient feel for the possible output of a single Pomodoro, all work was estimated in terms of the Pomodoros required to complete the task. The goal for a given sprint was to achieve sixty Pomodoro and use this figure as the budget for work that could be done. It was fairly difficult to estimate in terms of Pomodoro and often fairly inaccurate, although the productivity aspect certainly works, the more abstract and popular 'story-point' method of effort estimation is what would be used if the project was repeated.

Figure 1.6 shows the early Pomodoro tracking during two iterations. A successful Pomodoro would result in a sticker being allocated to that day. It was preferable to have Pomodoro goals for a given sprint rather than concrete work times (for example nine-to-five) since this allowed a great deal of flexibility whilst also maintaining that a weeks worth of work was to be done. Effort could be expended in the beginning of the week in order to have more time later on for example.
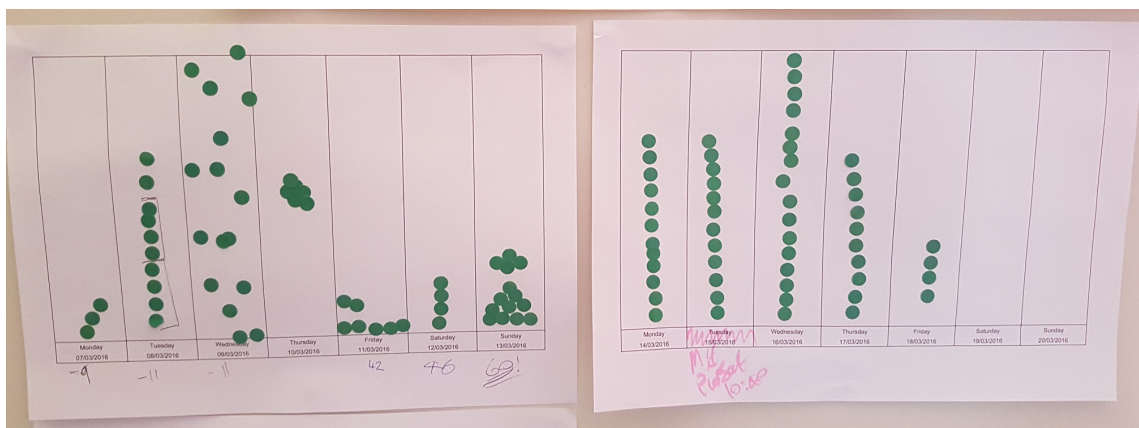
Figure 1.6: Tracking pomodoros

Chapter 2

Design

2.1 Overall Architecture

The overall architecture of the system can be described as a mixture of two well known architectural patterns, Model View Controller(MVC) and 3-tier architecture.

2.1.1 MVC

The Model View Controller paradigm is synonymous with web application development these days and is often employed without much consideration for alternatives. The fact is that the MVC pattern is so ubiquitous and well supported and understood that it is difficult to make a case against its use for a project of this nature, where MVC fulfils all expectations for a data driven front end to a web application. Many of the alternative approaches share the same primary goal as MVC, separation of concerns, keeping the display and data components separate. However these alternatives lack the penetration that MVC currently has and as such their comparative obscurity makes them less desirable from a general maintenance point of view since it can be expected that a greater number of developers will already be familiar with MVC. Many mature and well maintained frameworks offer MVC as default out of the box in a well supported and easy to understand manner, for these reasons not much consideration was given to other possible solutions although some were briefly looked at.

Figure 2.1 provides a high level overview of the way in which the MVC pattern is structured. For this project, each web page will be represented as its own view with its own dedicated controller, providing a high degree of modularity and helping to keep the individual controller classes thin in order to aid code navigation, maintainability and scalability of the solution.
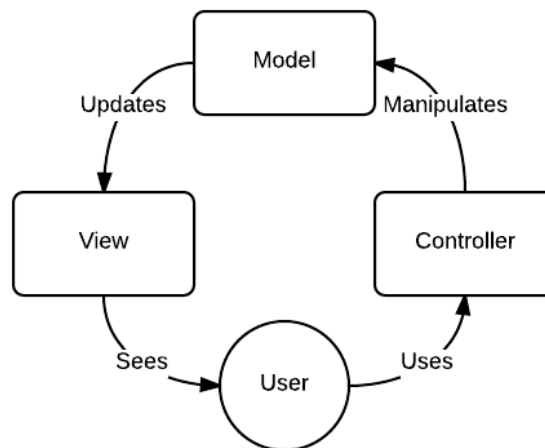
Figure 2.1: Model View Controller pattern overview

2.1.2   3-tier Architecture

The second architecture pattern making up the system design is 3-tier architecture. The goal of the 3-tier pattern is to separate the system into 3 distinct, modular tiers. These layers are the presentation layer, the business logic or service layer and the persistence layer.

In this project the presentation tier encapsulates the entirety of MVC pattern described in section 2.1.1 above, the MVC controller for a particular view will make a request to a service layer class and use the resulting data to populate a model for returning to the view.

The service layer is where the business processes are carried out. Logical processes, data transformations and calculations are all carried out within this tier. The service layer also acts as a go-between for the presentation and persistence layer, translating requests and results into compatible forms for the other tiers.

The persistence layer, or data layer, is concerned with data storage and retrieval, usually, and for this project entirely, from a database system. All crud operations and queries on database entities are performed within this layer allowing the service layer and its containing logic to be abstracted away from the database.

Figure 2.2: 3-tier architecture overview

The 3-tier pattern is a useful design within this project because it allows modularised and compartmentalised code to be written and maintained. Using the 3-tier pattern allows a developer to modify one of the distinct layers without having to rewrite the entire system. For example, all the queries and database calling methods could be changed to use totally different technologies and provided the same hooks are available for the service layer to gain access then the system would function as expected w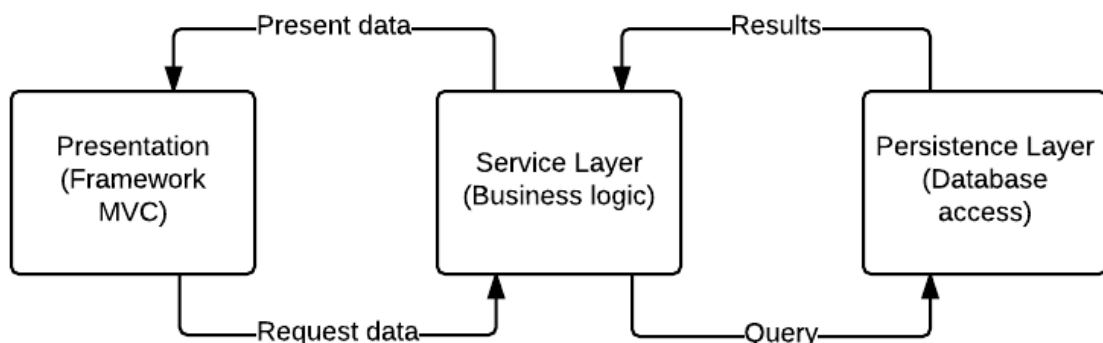ith no modification to service or presentation layers. Figure 2.3 shows the relationship between layers using a subset of the system components. Service layer dependencies are wired into each controller that require access to the processes in that service or the data in the database access object (Dao) coupled to a given service.
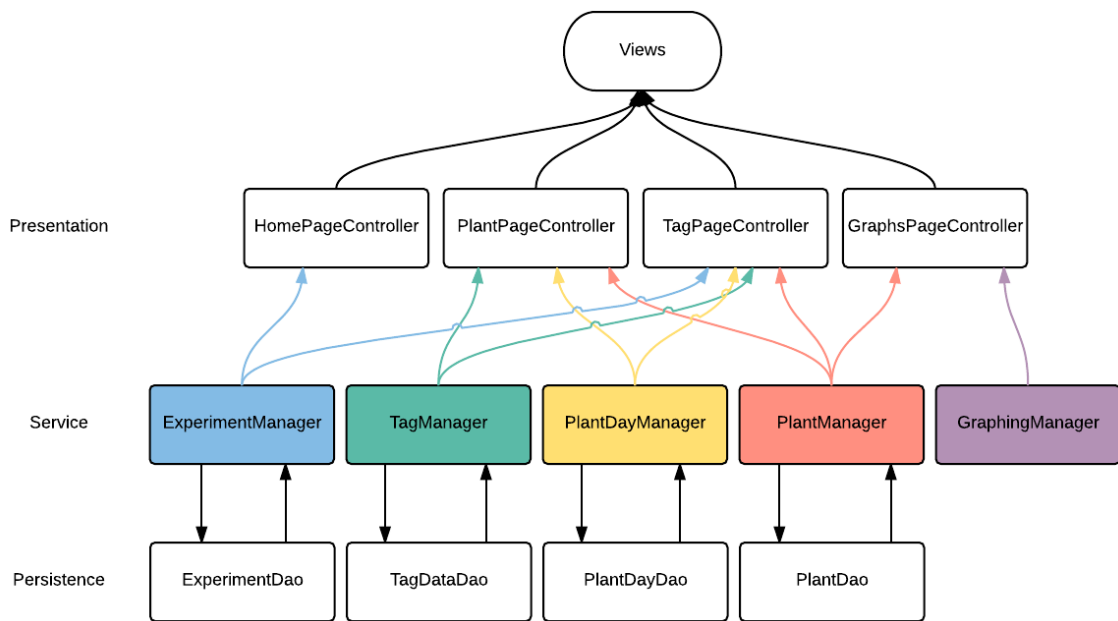
Figure 2.3: Architecture and dependency wiring

2.2   Framework and Programming Language

The sheer range of MVC frameworks available to developers is incredible and the decision of which to use is potentially difficult. It was not within scope to review a large amount of potential choices and to research which were mature, well supported solutions rather than a 'flavour of the month' framework. The two main contenders considered for this project were Ruby-on-Rails and the Java based Spring. Both are mature, well supported technologies with a large range of compatible libraries. Both have large and active communities surrounding and supporting them and both have well maintained official documentation. Both have convenient front-end templating technologies. Both share a 'convention over configuration' approach and importantly both are capable of supporting the designs discussed in section 2.1.

For the purposes of this project, Spring was eventually selected. Specifically the Spring Boot [9] bundle which greatly reduces initial configuration time at the start of a project and allows simple integration of complex packages (such as the Spring-Data package discussed in section 2.4) via so called 'starter' packages. External dependencies are minimal and the system can be run on a wide variety of hardware configurations without the need to rebuild. Even though both

frameworks offer inversion of control and dependency injection, the annotation based Spring approach is preferable, especially when coupled with a fully featured IDE capable of understanding the wired dependencies and annotations.

Using a framework based on Java has some arguable advantages, the fact that Java is compiled provides an extra level of checking during development and provides a quick notification of overlooked errors that may take time to uncover in an interpreted environment such as that provided by Rails. Java has built in security and type-safety providing peace of mind both in guaranteed resolution of variable types and built in access control at the virtual machine level.

Initially the project was developed using the latest version of Java (version 8), however, in order to take advantage of the available hosting environment at the NPPC it was necessary to use version 7 as this is what was installed on the systems. Fortunately the change was relatively straight forward since much of the work carried out prior to this

2.3   Domain modelling

Designing a domain model representation for the project was fairly straight forward. It was clear from initial investigations that a simple relationship existed between the primary domain entities that were going to be represented by the system. Essentially, there are Experiments, Experiments have a number of Plants associated with them and these Plants never belong to more than one Experiment at a time. Each Plant then has a number of images associated with it. There is a clear one to many relationship between these entities that is intuitive and can be modelled easily.

Further consideration of this domain model design following some initial implementation led to the inclusion of the PlantDay class in order to better represent the time serried nature of the images and data associated with a Plant. The Plant class now has many PlantDays which has many PlantImages. Each unique date that has images of a Plant is represented as a PlantDay. PlantImages with the same date are grouped together within the same PlantDay. This gave rise to the final design for the relationship between these domain entities, figure 2.4 shows how this relationship is modelled within the system.
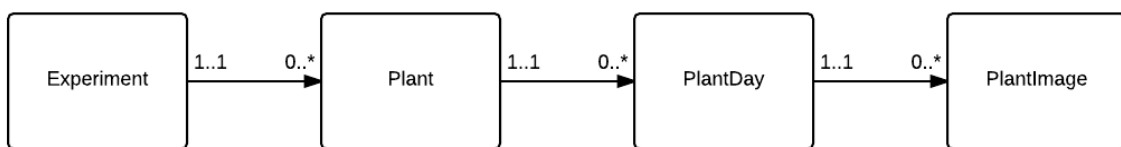
Figure 2.4: A simplified class diagram showing the relationship between primary domain entities

Having modelled the entities in the system, consideration was given to the best approach to adding data to the entities. There are two primary modalities to the data expected within the system, data directly associated with a top-level Plant and time-serried Plant data which would be associated with a date or PlantDay. After investigating different examples of data collected during experiments it was decided that there would be two primary data classes. A Metadata class which would hold a map structure and share a one-to-one relationship with unique Plants and PlantDay instances, and a TagData class which would be unique to the tag content it contained and potentially referenced by many Plant or PlantDay instances.

Since it was clear that experiment data is possibly very different from one experiment to the

next, the system had to be permissive in how the data is associated with domain objects. Arbitrary attributes and values had to be supported and this is why the approach with the Metadata class is to have a map structure holding string values for attribute key/value pairs. Using strings meant that both numerical and text data could be represented within the same structure, leaving conversion and/or checking requirements to other parts of the system if required. This was deemed acceptable since for the most part the system is only storing and displaying these data rather than using the values directly for calculations for example. Both the Plant class and PlantDay could have held the metadata map as instance variables, and indeed they did initially, but it became apparent that porting it out into a single class would enable the data to be queried natively on the database (discussed further in section 2.4) and cut down on the number of unique queries required in the Java code in order to find metadata instances.

Whilst the Metadata class represents general experiment data for each Plant and PlantDay, the TagData class was designed to hold sparsely associated data. The majority of Plants would be untagged, tags are used as supplementary comments against the occasional Plant to note some interesting information (common examples seen were 'dead' or 'small'). The approach with the TagData class was to have a unique instance for each unique tag, for example, all Plants with the tag 'dead' shared a reference to the same tag instance with content 'dead'. This provided a simple way to enable returning all Plants sharing a tag within an experiment via queries against the content of the associated TagData. Figure 2.5 shows the complete relationship between the domain model classes. Essentially these classes are all Plain Old Java Objects (POJO) which is why such a simple diagram suffices, the methods they contain are all getter/setter type methods and can be omitted from the relationship diagrams.
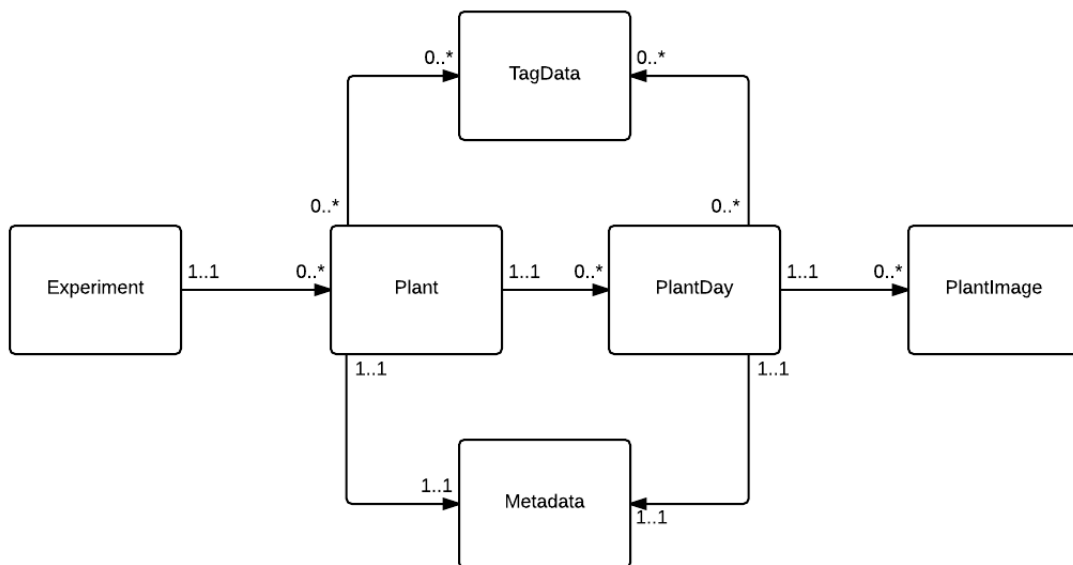

Figure 2.5: A simplified class diagram showing the domain model

2.4   Database

For this project the database structure is entirely derived by the Hibernate [5] object relational mapping (ORM) which is included in the Spring framework within the Spring-Data project as part of its Java Persistence API (JPA) support. The ORM system allows a developer to annotate Java code with keywords that inform the ORM system of how to represent a given class and persist it in the database. This technique allows the developer to manage the persistence element of a system from within the same object-oriented paradigm that the rest of the system is written in. It provides a level of abstraction away from the managing of the database itself leaving the developer to define only the structure of the data rather than its precise representation within a specific database system.

Listing 2.1 shows an example of these annotations within the Plant class. The class is annotated with @Entity to inform the ORM that it is a managed class to be persisted and table constraints are declared. The getter methods for the instance variables in the class are annotated with relationship definitions if applicable, including foreign key mappings and what manner of database instructions should cascade through the relationship to the related entity. The annotations can also define a fetch type which can take the value LAZY or EAGER, this defines whether the related entity objects should be fully initialised when the parent is called or whether, in the case of a LAZY fetch, a proxy object with no instance variables initialised should be returned. For this project, the use of LAZY fetch is preferred in all situations since it allows greater control over the performance of the system. If for example, the Plant class made an eager fetch for it's associated list of PlantDay objects, each PlantDay would be fully initialised at the time when the Plant object is retrieved from the database via a query, resulting in extra queries to the database in order to fully populate each PlantDay. Using this fetch technique allowed the PlantDays to be initialised when a method like .size() was invoked on their containing list or a getter method was invoked on the PlantDay itself.

```
1  @Entity
2  @Table(uniqueConstraints = @UniqueConstraint(columnNames =
       {"bar_code"}))
3  public class Plant {
4
5      @Id
6      @GeneratedValue(strategy = GenerationType.AUTO)
7      public long getId() {
8          return id;
9      }
10
11     @OneToOne(cascade = {CascadeType.ALL})
12     @JoinColumn(name="plant_meta_data_id")
13     public Metadata getMetadata() {
14         return metadata;
15     }
16
17     @OneToMany(mappedBy = "plant",cascade = {CascadeType.ALL},
           fetch = FetchType.LAZY)
```

```
18      public List<PlantDay> getPlantDays() {
19          return plantDays;
20      }
```

Listing 2.1: Detail of ORM annotation in Java class

Figure 2.6 details the database schema resulting from the ORM annotated relationships within the system. As discussed previously in this section, the relationships are a direct result of the structure of the Java code and the choices made with certain annotations, such as which side of a relation should hold a reference to the other.

The main deliberate change made to the default ORM mapping onto the database was to pull the dataAttributes map (a `Map<String, String>` representation in Java) out from the Metadata object into its own table, 'metadata_data_attributes'. The default would have been to map this as a blob type column within the metadata table, however, in order to ensure that all data within the database can be queried via native queries on the database itself, this extra table approach was taken.
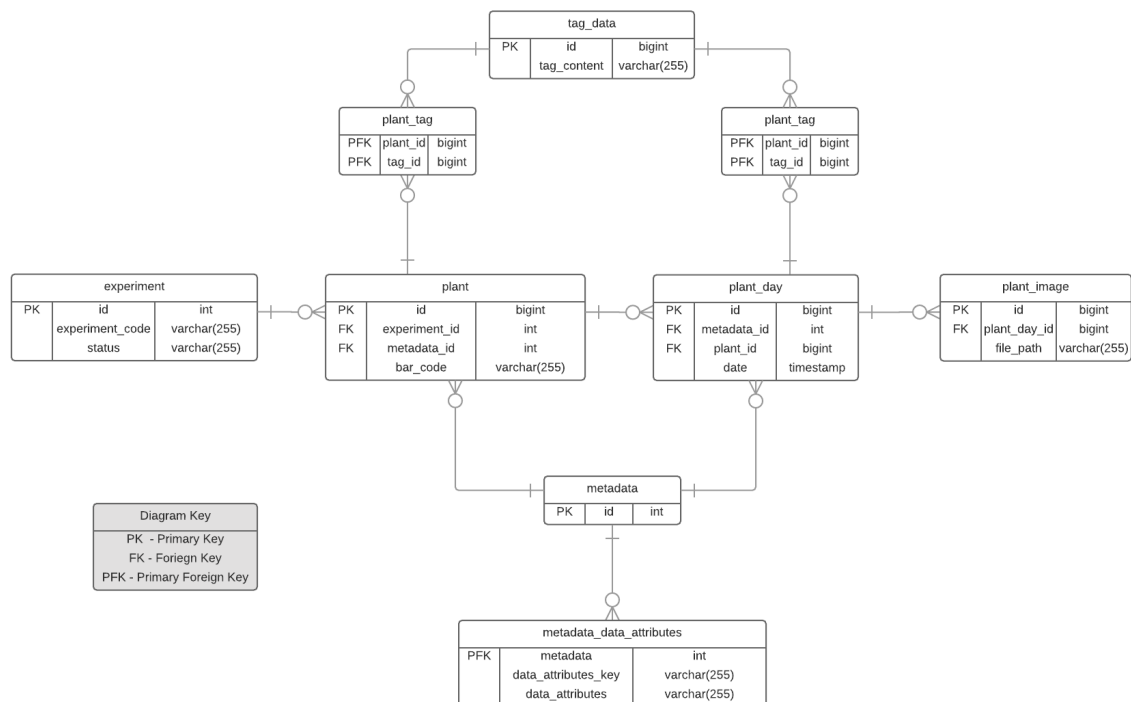
Figure 2.6: An entity relationship diagram for the database schema

From a developer based standpoint, when using an ORM such as the Hibernate system provided in Spring, the underlying database technology is mostly transparent. Provided the database is of a type supported by the ORM, the developer need not make any changes to the code in the system in order for the underlying database technology to change. Custom queries implemented within the system are defined using JPA syntax and are abstracted away from the underlying database. However, consideration was given to the database system to ensure that the best choice was made both to support efficient representation and to ensure compatibility with a wide range

of possible hosting environments and potential future maintenance needs.

It was clear from very early in the project that the data to be persisted within the system had structured relationships between entities which favours the more traditional SQL type databases over NoSQL solutions. It is forecast that the increased scalability offered by a NoSQL solution would not be required for this system, traditional SQL management systems are capable of scaling up to many millions of entries which should be more than sufficient for this system. With this in mind a MySQL solution was chosen, it is widely supported and well understood in terms of potential maintainers along with being a default technology on many operating systems including the hosted environment provided for the system.

2.5 Data Import

When designing the data import system the primary objective was to try and preserve, as much as possible, the format and structure of the original data files produced as part of an experiment. Figure 2.7 shows an excerpt from a spreadsheet provided as an example of real experiment data.

	genotype	barcode	comment	Tillering 04/01	FL	GS51 1st spikelets days from 01/01/15	GS55 panicle 50% emerged	cease watering	Height 10/02/16
2	43	9887	O7-01111		1	0	18	18	04/03/2016	98
3	124	9887	O7-01112		1	1	13	18	01/03/2016	93
4	205	9887	O7-01113		1	1	13	18	11/03/2016	92
5	74	9887	O7-01121		1	0	25	29	11/03/2016	98
6	155	9887	O7-01122		1	0	29	33	11/03/2016	106
7	236	9887	O7-01123		1	0	33	36	11/03/2016	106

Figure 2.7: A sample of provided experiment data in its original form

For representing these data, the CSV format is the obvious choice and is available as one of the default export options for spreadsheet applications. The design challenge was routing the data from the CSV file into the system whilst enabling arbitrary values for identifiers to allow for the vast potential range of different data to be captured by the system. However, the format of the files needed to change as little as possible to simplify the use of the system and to minimise the amount of work required on the data to get it ready for import.

The solution was to use a simple annotation based method to identify how each column of the CSV should be routed. Listing 2.2 shows how these annotations are represented and used within the header of the CSV file.

```
1  {{plant-a}}genotype,{{bc}}barcode,
       {{plant-t}}comment,{{day-a}}Growth Stage~~51,
```

Listing 2.2: Excerpt showing annotated CSV header for data import

There are four supported annotations:

1. {{**bc**}} - Identifies the column as the barcode for a Plant. This serves to uniquely identify the Plant to which the data in the row should be assigned.

2. {{**plant-a**}} - Identifies the column as a Plant attribute. The content of the header column is used as the identifier and the content within the row is used as value.

3. {{**plant-t**}} - Identifies the column as a Plant tag. The content of the header column is ignored and the content of the row is added as TagData to a Plant.

4. {{**day-a**}} - Identifies the column as a PlantDay attribute. Uses a specific delimiter token ˜˜ in order to split the header column into a key value pair. The column in the row in this instance needs to be a date such that a specific day can be allocated the data.

Other methods investigated included using XML and JSON within the CSV to identify certain fields in the header and columns. However, even though both XML and JSON are convenient ways to represent and parse semi-structured data, they are not particularly quick to use in terms of the characters needed to be typed and neither are they especially readable when crammed into the header of a CSV file. In keeping with the goal of simplicity these alternatives were quickly discounted in favour of the simple annotation method.

2.6   UI

The UI design process used within the project is fairly straight forward and relatively basic. Essentially the process used would first involve a wire frame mock up of the page, an example for the Plant Detail page is shown in figure 2.8. This would then be prototyped and the design would evolve iteratively as implementation proceeded and user feedback testing was conducted.
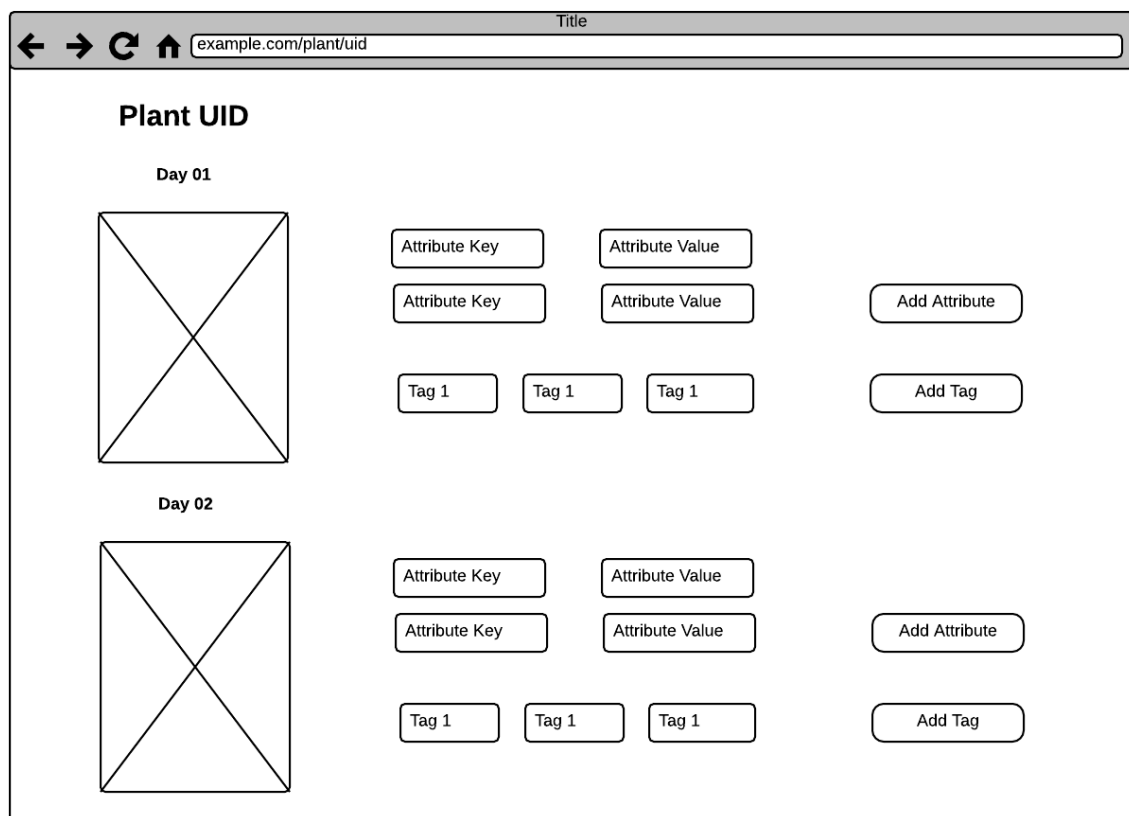
Figure 2.8: An early wire-frame design for the Plant Details page

Figure 2.9 shows the final look of the Plant Detail page, it's clear to see how it relates to the initial wire frame and also where implementation decisions have resulted in some minor tweaks to the design. This same process was followed for all the pages within the system, where the focus has been on a simple yet usable design with minimal clutter.

Figure 2.9: A screen shot showing final design of Plant Details page

A design consideration for UI interaction was that user interactions within a page should always use an asynchronous method to update the page with results of the interaction, such as crating a graph, tagging Plants or editing attributes. This asynchronous approach was achieved via the use of ajax based Javascript functions invoked when users would click buttons or links. The view controller class within the framework would process these ajax requests and return the HTML for a partial page fragment which would then be injected back into the page in order to provide updated content without the need of a page refresh.

2.7 Tools and third-party services

2.7.1 Intellij

IntelliJ [6] is the core development tool used during the completion of this project. It is a fully featured Java integrated development environment (IDE) that has support for a wide range of features including Spring and Github (see section 2.7.2) integration right out of the box. Its code completion and debugging tools are significantly more refined in comparison to the most popular alternative Eclipse, allowing for faster writing of code and easier debugging. As with any reasonably modern IDE, IntelliJ comes with the facility to run sophisticated test suites, providing code coverage metrics and providing auto-generated method stubs in implementation or test classes further speeding up development time, especially in boiler-plate heavy languages such as Java.

IntelliJ also provides in-built static analysis tools that run automatically has part of committing changes to version control via the IDE. This is useful as it is configured to highlight warning level issues which include code style along with potential logical mistakes within sections of code and even spelling errors. Having these checks at commit time enables the developer to review any potential problems before the code gets checked into the repository, although the results are often a little too pessimistic they are still useful. Figure 2.10 shows the IntelliJ IDE with static analysis results displayed.
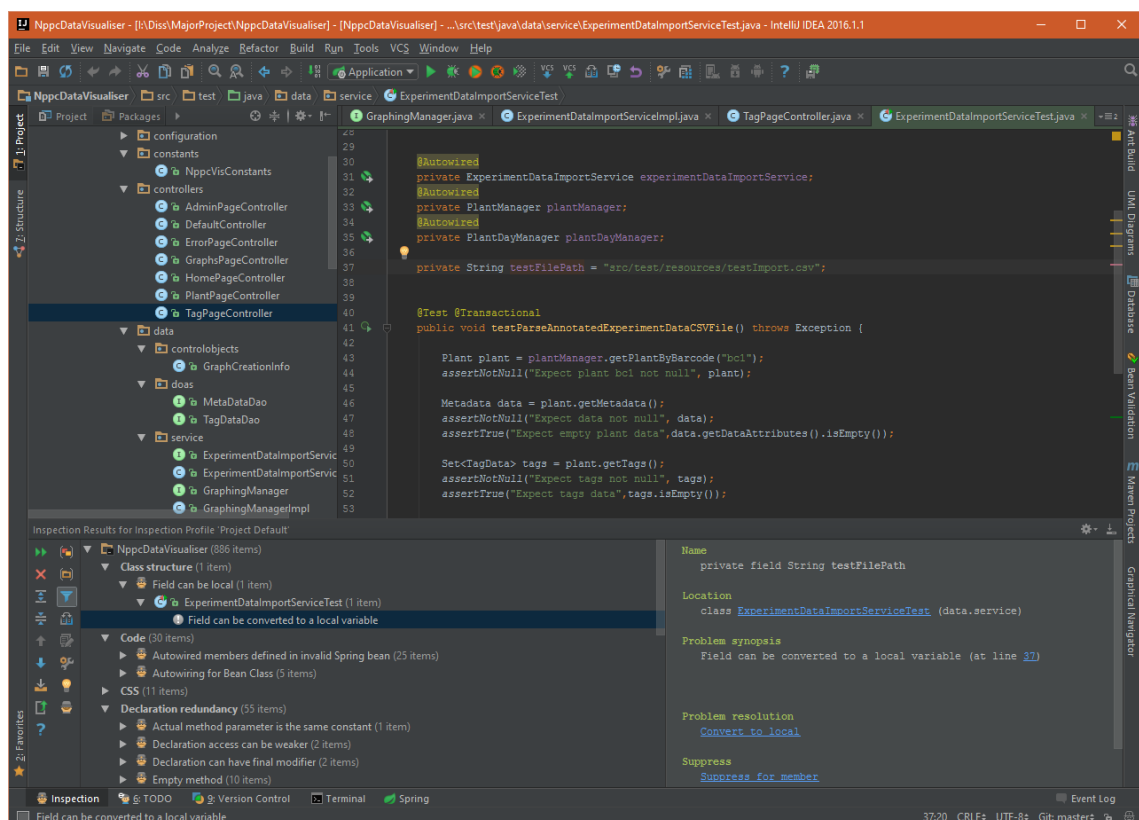

Figure 2.10: The IntelliJ IDE showing result of static analysis

2.7.2 Git and Github

The use of version control is invaluable in modern software development. It has become a necessity in even the smallest of hobby projects since it allows the developer to be confident in making changes without having to worry about rescuing previous version if things go wrong and provides development teams with the means to work concurrently and collaboratively on the same code base.

The version control system selected for this project was Git [4], having previously used alternatives such as Subversion I chose Git for its integration with more numerous, modern services and the fact that it allows local copies of a repository which is synced with a remote repository as opposed to the remote-only approach taken by Subversion.

The Git repository for this project is hosted on Github [18], a web based service dedicated to providing git repository hosting and related facilities, such as commit history tracking, release visioning and integration with third party services. There are alternatives to Github available but due to familiarity brought on by hosting all previous projects and the fact that Github is now an industry leading solution, it was decided to use Github for this project without any real evaluation of alternatives since it was well known that Github could provide all facilities required for the purpose of this project. Figure 2.11 details one of the tagged release versions of the system within Github. Having releases tagged in this way allows easy rollbacks to previous release versions in the event of any major issues in a new release.
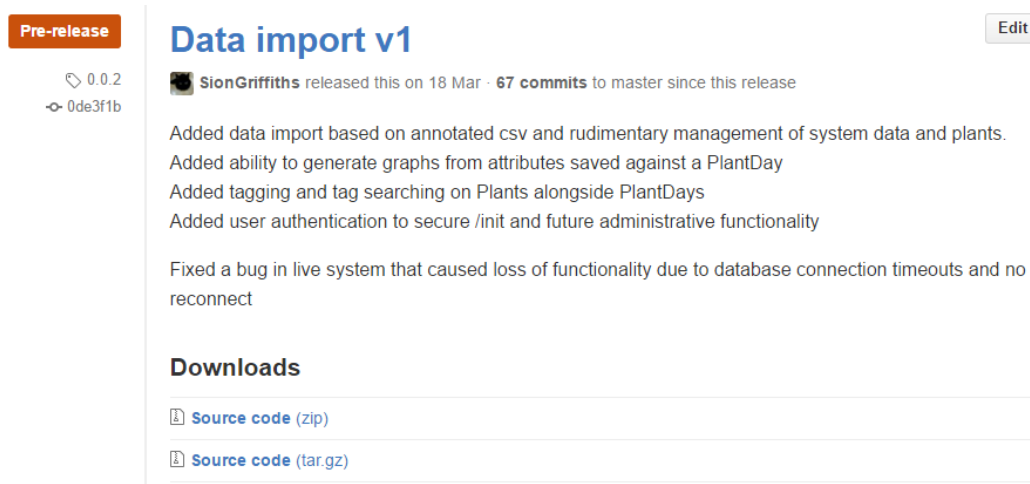
Figure 2.11: A tagged release of the project within Github

2.7.3 Jira

Jira [13] in an issue tracking and project management tool provided by Atlassian, an Australian software company. It is an industry leading product used by many companies for tracking their projects and the issues within them. Its use on this project was in support of the agile approach to project development, allowing the specification of user stories, development tasks and their inclusions within configurable sprints or development iterations. Figure 2.12 shows the current sprint view in Jira, user stories are grouped into 'lanes' corresponding to their status, allowing a simple way to track the work completed and left to do within in the current development iteration.

Figure 2.12: Current sprint screen in Jira

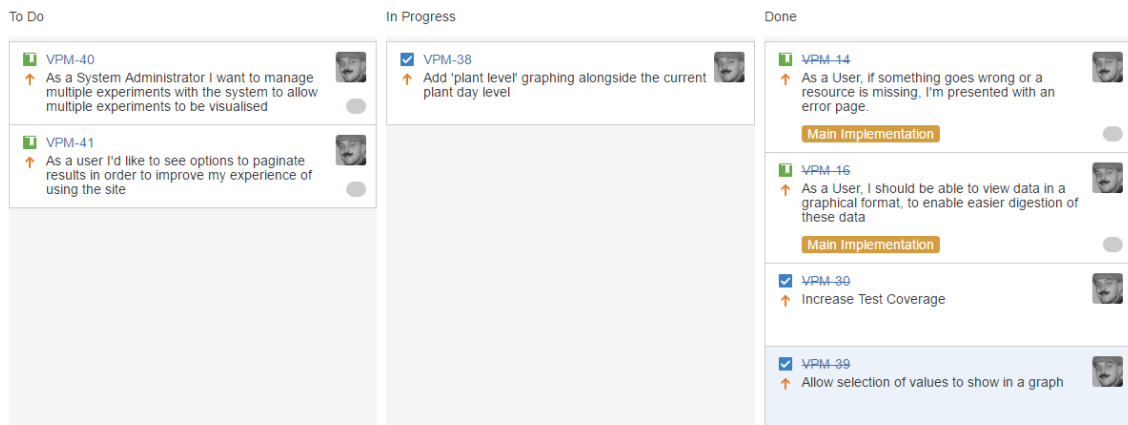Bugs could also be tracked as issues within Jira and added to the current sprint if necessary, I found this to be a valuable way to deal with emergent issues during development as it allowed a simple way to assign priority to urgent issues and keep track of less urgent bugs in the project backlog to be worked on in a future sprint. Figure 2.13 shows a selection of bugs raised as part of development, Jira provides simple methods for filtering all issues against a project by type or status allowing quick access to screens such as this.

Figure 2.13: A sample of bugs raised in Jira

Another helpful feature was the integration with the version control repository hosted on Github. Referencing the issue ID in Jira in a commit message linked the commits with the issue within Jira. This provided a handy way to track development against particular issues over time and allowed a quick way to navigate between the issues in Jira and the commits on Github.

Figure 2.14: The version control commit tracking within Jira

There are a vast array of alternatives that could have been used for issue tracking within the project, many provide the full array of features that were used in Jira during the development of this project. However, Jira being the industry leader, provided an opportunity to gain further valuable experience of its use in a day to day, agile development project. Having previously been involved in the running of a Jira system during my time in industry provided me with familiarisation in configuring a project for my needs and confidence in being able to do so quickly. This was enough to chose Jira over the alternatives that were evaluated such as Waffle.io and the native issue tracking feature provided with Github.

2.7.4 Codeship

Codeship [16] is a web based Continuous Integration(CI) service. Working in conjunction with the version control repository, Codeship will detect up any commits made to the repository hosted on Github and execute build and test scripts defined as part of the initial setup of the CI service. Use of a CI system within the project provided assurance that each incremental change made to the system integrated correctly and that all tests continued to pass. A notification would be sent in the event of build or test failure.

The build script for the project can be seen in figure 2.15 showing how the project databases are setup and the environment is configured prior to executing the project build and test commands.

The scripts are invoked within small Docker [12] based environments which allow build dependencies to be modularised and configured quickly. The initial integration of the CI system into the project environment was extremely simple, linking the Github repository for the project was a couple of mouse clicks and the script below is the entirety of the extra configuration required to get the CI system fully up and running. It was because of this speed and simplicity of configuration that Codeship was chosen over rival offerings such as TravisCI [10] which appeared to have a much more complex initial setup during evaluation.

Figure 2.15: The project build script on Codeship

Figure 2.16: A sample of the build history in Codeship

2.7.5 Plotly.js

Plotly.js [7] is an open source graphing library built on top of technologies such as d3.js, a Javascript data manipulation and visualisation library, and stack.gl, a wrapper around WebGL and other associated technologies. There are numerous alternatives libraries that could have been chosen, including d3.js itself and Google Charts amongst others, but the ease of integration with the system and the look and feel of the output from Plotly.js made it the superior choice for this project. Figure 2.17 shows a Plotly.js generated graph embedded within the Graphs page in the system.

Graphs

Select X axis attribute :

genotype ▾

Select Y axis attribute :

Height 10/02/16 ▾

⦿ **Scatter** ○ **Box**

| Swap axis | Create Graph |
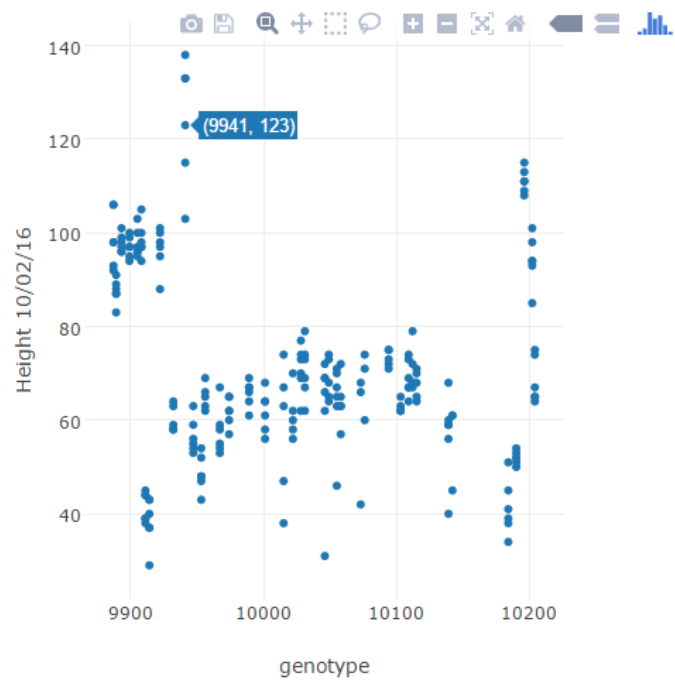
Figure 2.17: Project graph page featuring a Plotly.js generated graph

Chapter 3

Implementation

3.1 Integration with NPPC data repository

Initial implementation and spike work was conducted with the assumption that a network connection to the NPPC data repository would be established and maintained by the system. After a meeting with Dr Colin Sauze, it became apparent that the project would be hosted on a machine at the NPPC and that the hosting environment (discussed in section 3.5) would have the data repository mounted as a network drive as default. This meant that the system could treat the repository as a local drive on the hosting environment, removing the requirement for the system to manage a network connection to the repository.

In order to avoid transferring and re-hosting images from the repository it was necessary to add the repository as a static content resource within Spring such that the Tomcat web server driving the system could serve the images. This was simple to achieve within Spring by providing a custom implementation of the `WebMvcAutoConfigurationAdapter` class and its `addResourceHandlers` method where a file location could be defined as a resource and assigned a URL based resource handler. In the case of the images within the repository, they are handled from the base URL `/images`.

The initial design involved using a recursive method to parse the various directories within the repository and create the various plants for an experiment from the directory structure. Performance was a noticeable issue with experiment initialisation taking an unreasonable amount of time to read plants from the repository. During troubleshooting this performance issue, an iterative approach that was tightly coupled to the file system structure was implemented for comparison purposes. This iterative approach was over twice as quick as all the tested and recommended recursive methods especially when checks on specific directories were required such as those needed to check for filtered image modalities. It was decided that the iterative approach offers enough of a difference in performance for the tight-coupling to be a fair trade off. However, changes were made to the directory structure of the repository soon after the implementation of the iterative approach. Discussions followed with Dr Colin Sauze who has oversight of the repository and it was confirmed that no further changes were planned since utilities he had written to interact with the repository were now also tightly coupled to the new structure. In the event that further changes need to be made, a recursive solution is provided within the source code (commented) of the project ensuring the project can function without much in the way of code change.

3.2   Graphing System

As part of the planning and design phases for the graphing based functionality in the system it was necessary to investigate and decide upon a framework or library that would allow the generation and display of graph based visualisations within the system. Having selected Plotly.js it was necessary to integrate the library into the system. Being a Javascript library made this fairly simple. However, in order to allow the user to configure certain settings that control the way the graphs are generated, a number of helper functions had to be implemented in Javascript to provide a simple interface onto the Plotly library from within the Graph pages on the site. The flexibility of Plotly and the wide range of graphing options that it provides meant that implementing a means for the user to configure it completely was impractical.

The pages in the system are built using a responsive CSS design (responsive functionality provided by Bootstrap), in order to be compatible and maintain the responsive resizing of elements within the page, the graphs generated via Plotly would resize in proportion to the html element in which they were positioned. Unfortunately, even though the documentation for Plotly mentions dynamic resizing it was not possible to get the feature to work correctly within the system and the exact reason for this is still unknown. The result of this is that each graph generated currently has fixed dimensions.

Having integrated the Plotly library into the system, the next implementation step was to provide it with data. This was achieved by providing a number of asynchronous Javascript functions that would send the user selected parameters for the graph to a method in the `GraphsPageController` class. The `GraphsPageController` takes advantage of the convenience annotation `@ResponseBody` provided by the Spring framework that converts the object returned by a controller class (specified using the `@Controller` annotation) into JSON format. This made the data returned to the asynchronous function compatible with Plotly and could be passed directly to the function creating the graph. The graph data itself is retrieved by the `GraphingManager` class using the user selected attributes to query the data layer for resulting Plants or PlantDays.

Implementing functionality that would return plant results into the graph page when clicking on data points in the graph provided to be more complex than initially estimated. The added complexity was a direct result of a separate design decision to move metadata attributes to their own table within the database (discussed in section 2.4). The difficulty was in building the correct query in JPA query language in order to return plants based on the key/value attribute pairs for two separate attributes, gaining familiarity with the syntax and experimenting with the various join query options eventually solved the issue.

3.3   Domain model implementation and ORM

The design for the domain model was relatively straight forward and usually implementing simple relationships between classes is trivial, however, since the project uses Hibernate ORM it was necessary to become familiar with some of the inner workings of such a framework during the implementation phase.

The entity classes (`Experiment`, `Plant`, `PlantDay`, `Metadata` etc) within the system are mapped directly to database tables using annotations as discussed in section 2.4. In order to retrieve information from the database it was necessary to provide methods to query the database,

the preferred way of doing so is to use dedicated repository classes (database access objects or DAOs) to access the database. Initially the implementation of these DAO classes used a manual approach that had been tested during the prototyping phase, listing3.1 shows this manual approach when retrieving Plant objects via their barcode. A query is build using a standard parametrised technique to avoid injection attacks since the barcode comes from the front end of the system in some cases. The Query object itself is built by invoking a Hibernate 'session' retrieved using a static accessor method, the query is then run on the database and results returned.

```java
public Plant getPlantByBarcode(String barcode) {

    String hibernateQuery = "FROM Plant WHERE bar_code
        =:bar_code";
    Query query = getSession().createQuery(hibernateQuery);
    query.setParameter("bar_code", barcode);

    return (Plant)query.uniqueResult();
}
```

Listing 3.1: Manual Hibernate session handling

Unfortunately the approach used in listing 3.1 was not compatible with more complex database interactions especially, those involving lazy initialisation(as discussed in section 2.4). Using the `getSession()` approach to retrieve a Plant would return a Plant proxy object with a lazy, non-initialised list of the PlantDays associated with the Plant. Attempts to read or manipulate PlantDay data within this lazy list would result in Hibernate exceptions where the session attempting to access the database is now closed since the session was scoped only within the `getPlantByBarcode` method. Attempts to maintain the same session for multiple queries and lazy initialisation proved impractical. Research showed that using the `@Transactional` annotation would enable the Spring framework to manage transactions within classes or methods using the annotation, in order to take advantage of the framework managed transactions it was necessary to completely refactor the persistence layer of the system.

Whilst investigating the options for the persistence layer the `Repository` interfaces provided by Spring were discovered which provide a means to define queries in an extremely simple and efficient manner whilst also hooking into the transaction and session management provided by the `@Transactional` annotation. A particular extension of the `Repository` interface, the `JpaRepository`, was chosen as the base for all the domain DAOs in the system. The `JpaRepository` interface provides both CRUD functionality and provides a simple pagable function that allows pagination queries to be easily defined. Interestingly using these interfaces provides a means for the framework to infer database queries from method signatures alone, no implementation class is necessary beyond the custom implementation of the interface.

```java
Plant findByBarCode(String barCode);
```

Listing 3.2: Interface query definition

Listing 3.2 shows how the same functionality as in listing 3.1 is achieved using the `JpaRepository` interface. There's no override or concrete implementation of the method, it is called directly from the interface and all functionality is inferred by Spring. For more complex queries the query can be built in the interface directly using the `@Query` annotation as shown in listing 3.3. Queries can be parametrised using the `@Param` annotation as shown.

```
1  @Query("select p from Plant p join p.experiment e where
       e.experimentCode = :experimentCode ")
2  List<Plant> findByExperimentCode(@Param("experimentCode")
       String experimentCode);
```

Listing 3.3: Building custom hibernate queries by decorating interface methods

Changing to use the `JpaRepository` interfaces allowed abstraction away from the database sessions and individual transactions and allowed queries to be defined in a much more developer friendly and maintainable fashion. Using the `@Transactional` annotation instructs Spring to manage the commit and rollback functionality on the database further providing utility to the developer by providing innate rollback in the event of any error during a transaction.

In solving the lazy initialisation issue discussed previously, it became apparent that the domain design calling for bi-directional relationships between one-to-many related entities would not be workable or necessary. The bi-directionality could be achieved using standard Java getter methods alone without needing to have this translated to the database via ORM. Having the bi-directionality at a database level caused circular instantiation to occur during initialisation of lazy collections within an entity object, this would manifest in stack overflow exceptions showing evidence of a poorly thought out design and was quickly rectified.

3.4   Data Import

Implementing the design for the experiment data import system was fairy straightforward. The design called for the use of custom annotations in the header of a source CSV file in order to efficiently route the contained data correctly. When the header of the CSV file is parsed, the header is separated from the body of the file, the columns corresponding to each annotation type have their index number added to a list corresponding to that type. For example, if the second column in the header was annotated with the {{plant-t}} (see section 2.5 for annotation details) annotation, then the index '2' would be added to the list holding plant tag column indices, further columns with the same annotation would have their indices added to the same list. In a similar way, the column containing the identifying barcode for the Plants is identified and its index saved.

The body of the experiment CSV file is processed line by line. A line in the file is represented by a list of String objects. For each line in the file, the identifying barcode is extracted by reading the value in the line list occupying the index corresponding to the barcode, the correct plant can then be found in the database. A similar approach is taken for the other column types, essentially their values are extracted by directly reading the value in the line that corresponds to the index in a given list. Listing 3.4 shows how this is achieved for plant data attributes, the indices corresponding to plant attribute columns are held in the list `plantAttribIndex`, for each index in this

list the values are read directly from the line. In this instance the key/value attribute pair consists of the header and the column value, this attribute pair is then saved into the plant corresponding to the barcode found in the line.

```
1  for(Integer i : plantAttribIndex){
2      if(line[i] != null || !line[i].equals("")){
3          Metadata data = plant.getMetadata();
4          data.addDataAttribute(header[i], line[i]);
5          plantManager.savePlant(plant);
6      }
7  }
```

Listing 3.4: Detail of routing data during import

Issues encountered during the implementation of this feature were mostly generic issues concerning the reading in of data from file, notably date formatting and certain special characters causing issues when attempting to display them as attributes or tags in the front end of the site since Spring and the page templating framework Thymeleaf have some inbuilt special character sanitisation.

3.5   IBERS hosted environment

As discussed in section 2.2, a consideration during the selection of framework within which the project would be implemented was the number of dependencies it would place on the environment it was hosted. The less dependencies then the more attractive the framework since this directly affects the time required to configure and maintain the environment. This project required that three dependencies be present on a given environment to enable the system to function, the Java runtime environment, MySQL database server and the Apache Maven build tool which is bundled into and required to run Spring-Boot applications.

Initially the Java version used for the project was version 8, however, it was more convenient for the NPPC to provide a server with Java version 7 forcing the change in targeted language level. Fortunately, this occurred sufficiently early in the implementation that not much of the codebase was dependant on new features not available in the version 7 Java runtime and the required changes were small and trivial.

The provided hosting environment was provisioned and set up by Dr Colin Sauze, the data manager at the NPPC. The hosting environment is a virtual machine running Ubuntu v14.04 hosted on a Intel CPU based server with 8GB of RAM. It is hosted with the Universities firewall and as such is only accessible from within the University network (or via VPN). The network restriction meant that the deployment of a release build of the system could not be completed via the project continuous integration platform. The deployment process was not fully automated for the project. For a given release version, the source code would be checked out from the version control repository and the system rebuilt and restarted on the server itself. Since this required only three commands at the end of each week to be entered into the system terminal, it was deemed that automation was not necessary.

Chapter 4

Testing

4.1 Overall Approach to Testing

The overall approach to testing was to have high test coverage of system features and functionality and to automate these tests wherever it was feasible to do so. Automated tests would run often as part of the normal development workflow and provide continuous assurance of functionality and system environments. Where automation was impractical, alternative approaches were taken to ensure that the system was fully tested in a robust manner.

4.2 Automated Testing

For the purposes of automated testing, a separate database was used. The database would be completely recreated for the start of each run of the test suite and dropped at the end. Prior to the tests running, the database would be seeded with test data that is similar to the real-world data expected in the production database. By using this method the tests would more closely mirror the real-world behaviours of the system and each run could be insulated from the data changes made in previous runs.

A Continuous Integration(CI) system was used in order to facilitate the convenient and regular running of all automated tests in the project. The CI system would build the project from source each time a commit was made into the version control repository. As part of this build process the full test suite would be run. Any issues encountered during this process, from compilation errors to test failures, would result in the build being rejected by the CI system. In the event of a rejected build, the CI system would notify via email of the build failure. This feature turned out to be invaluable since it highlighted a configuration issue that did not affect my local development environment but would have affected the server the project is hosted on. Because the tests were automated and I was notified of a failure, I saved what is likely to have been a significant amount of debugging time at the next release of the project to the server. Time was also saved since the full test suite didn't need to be run locally at development time, single tests could be run and the full test and integration suite would be invoked on commit to the version control repository.

Figure 4.1 shows the test results page for the automated tests as generated by Intellij when the full test suite is executed locally. Certain tabs within the page are expanded for display purposes.

Figure 4.1: Automated test result page generated by Intellij

F

4.2.1 Unit Tests

When implementing most of the service layer classes for the system a TDD approach was employed in order to ensure high test coverage of the parts of the system which incorporate the business logic. Using TDD helped evolve the design of these service classes by ensuring that nothing was built in a way that was difficult or convoluted to test. Tests are implemented on a method by method basis for the most part, that is, each method in a service will have its own unit test to ensure functionality.

A simple example is shown in listing 4.1 detailing a test for the tags reset functionality in

the PlantManager service class that is invoked as part of deleting the data associated with an experiment.

```java
@Test
public void resetTagsForExperiment()  {
    Long id = 10L;
    Plant plant = plantManager.getPlantByID(id);
    Experiment experiment = plant.getExperiment();

    assertEquals("Expected number of tags to be 2", 2 ,
        plant.getTags().size());

    plantManager.resetTagsForExperiment(experiment);
    assertEquals("Expected number of tags to be 0", 0 ,
        plant.getTags().size());
}
```

Listing 4.1: Unit test for the PlantManager service

Most of the classes not covered by unit testing are tested via integration testing. The overall coverage for automated tests in the system is 79 % of all lines written in Java.

4.2.2 Integration Testing

Integration testing for this project was achieved primarily through testing of the MVC controller classes. The goal behind these tests is to make requests to the various available routes within the system and verify that the correct results are returned. Being a web based system, all functionality is in some way linked to a request mapping or route in a controller class. Testing these routes provides a convenient method to ensure the distinct layers and components that make up the system are working as intended and the interactions between them are as expected.

Integration tests for this project take advantage of features provided by the Spring framework in order to simplify the configuration of the tests and the mocking of certain aspects of the system, such as the application context in which the tests are running. These mocked dependencies and use of the same static data and database each run, ensure that results can be verified consistently.

The example test in listing 4.2 shows how a mockMvc object is used to simulate the web application context and perform requests against the application, in this case a HTTP GET to the path '/plants' which should return the plants page. The HTTP session object can be managed as part of the tests and injected into individual requests to ensure compatibility with real-world usage. Following the HTTP request the results can be verified. In the case of the example test the HTTP status is checked to ensure that the server returned status code 200 (Ok). The content of the response is verified then finally, a check against the view() method is made to ensure that the correct page has been returned as a result of the request.

```java
@Test
```

```
2  public void testShowPlants() throws Exception {
3      String testBarCode = "bc1";
4      this.mockMvc.perform(get("/plants").sessionAttrs(sessionattr))
5              .andDo(print())
6              .andExpect(status().isOk())
7              .andExpect(content().string(containsString(testBarCode)))
8              .andExpect(view().name("plants/show"));
9  }
```

Listing 4.2: Simple integration test example

A similar approach is adopted for all integration tests throughout the system. For each tested route, the request is simulated and results verified in much the same way as in the example test. In more complex tests or those testing functionality which require more robust verification there are extra steps taken such as asserting the existence of certain page model attributes or objects being passed to the front end views.

4.2.3 Stress and Performance Testing

Performance and stress testing was carried out through the use of Apache Jmeter [1], an open source Java application built to measure site and application performance under controlled loads. Jmeter enables the simulation of a number of concurrent users accessing a given site. These simulated agents follow a defined sequence of actions as specified in the test script. Unless otherwise stated the tests run with ten concurrent agents and the tests are repeated thirty times in order to smooth out any outliers in the data.

The tests were all carried out against the project hosted on the remote server provided by Ibers. The machine used to run the Jmeter scripts is a powerful desktop machine using a recent generation of Intel i7 processor featuring 4 cores and able to process 8 concurrent threads. It is necessary that the test machine be connected to the Aberystwyth University VPN in order to reach the target server although the impact of the extra overhead appears minimal and is considered for the sake of comparing results. Although the ten concurrent users may seem low, the throughput on average is over 100 requests per second when run from the test machine which is significantly more than ten real users would be able to generate.

For the purposes of this project Jmeter was used to assess whether pages in the site would load within defined time limits and whether implementation decisions have an adverse effect on performance. In general the goal was to have pages served within 300ms with a hard limit of 1000ms, or one second, although this does not include image load times. A target of 300ms is well under the 1 second limit for keeping a users flow of though as identified as part of a study conducted by Nielsen [22]. Running the tests regularly could also help highlight issues that may not be uncovered under other forms of testing such as intermittent problems that could result in request errors that would be difficult to reproduce otherwise.

General results as output by Jmeter are included in figure 4.2 for an experiment which has been initialised with data. The initialisation is an important distinction because the amount of experiment data significantly affects the initial page response time for the Graphs page, other pages are affected somewhat but to a much lesser degree. Figure C.8 displays the results of running the

same test without the data having been added to the experiment and it's clear to see the effect on the load time for the Graphs page. Having these results available during development informed some of the design choices within the Graph page such as having the graphs themselves and any plant objects loaded via Ajax following user interaction as opposed to being populated into the page on load.
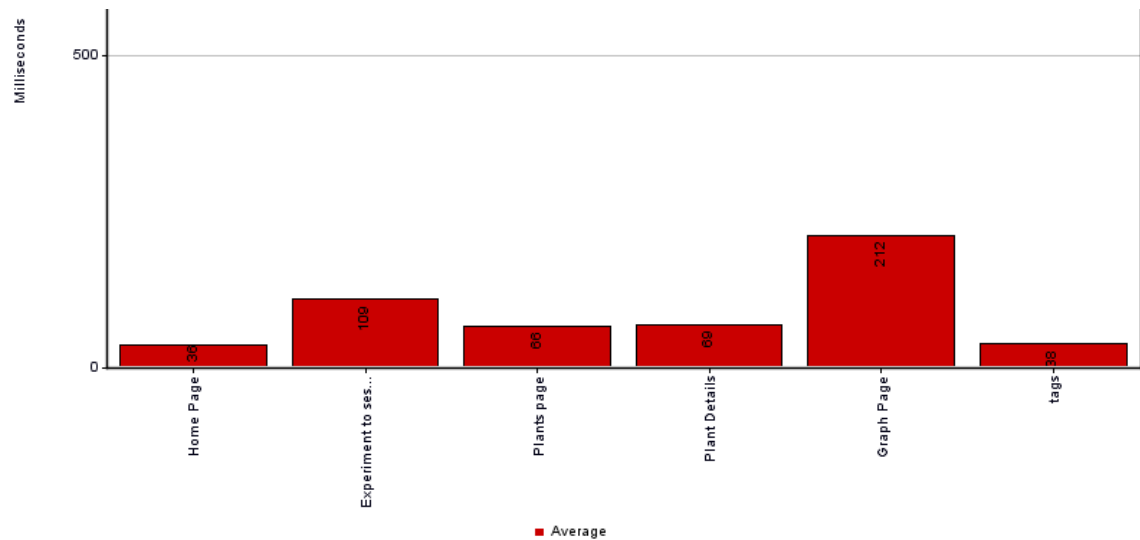
Figure 4.2: Visulisation of Jmeter test result of a fully initialised experiment
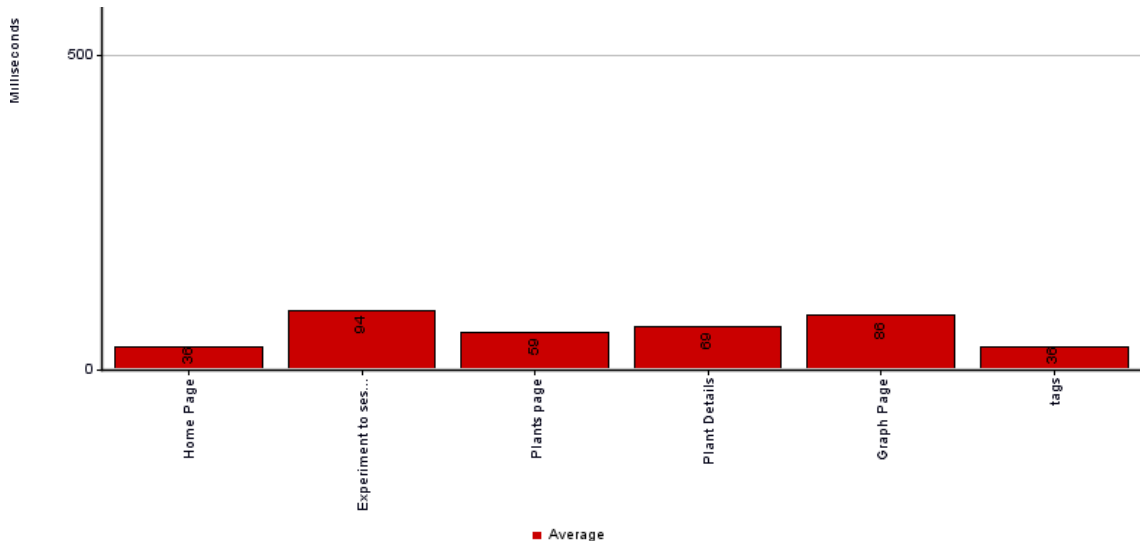
Figure 4.3: Visulisation of Jmeter test result of a partially initialised experiment

The effect of choosing pagination defaults for the plants and plant detail pages could also be measured although in the case of pagination the real limiting factor is the bandwidth and render time required. However, the effect on request time and loading on the server could be seen and monitored for any potential issues. In an experiment with many plants or a large amount of data the response time would increase significantly with page sizes of over 50 or so plants but no other adverse affects were noticed on the system even with a significant number of requests.

4.3   Manual Testing

For areas of the system where automated testing was impractical or insufficient to verify results, a manual approach was taken and test tables used to verify functionality was as expected.

4.3.1   Admin Page Test Table

Much of the functionality on the Admin page relies on an active network connection to the NPPC data repository and as such is unsuitable for automated testing. There was no feasible way to establish a connection between the continous integration server and the NPPC data repository therefore the manual verification of functionality was necessary.

Test	Input	Expected Output	Pass
Attempt to access admin area without login	Go to /admin without login	Redirected to administrator login page	✓
Attempt to access admin area with correct login	Go to /admin with login	Admin is page is displayed	✓
Attempt admin login with incorrect credentials	Submit admin login form with incorrect credentials	Error displayed to user.	✓
Admin log out	Click logout button from admin page	Redirect to home page and authorisation cleared from session	✓
Initialise Experiment	Click initialise button for uninitialised experiment	Experiment begins initialising - plants are created	✓
Update experiment	Click Update button on initialised experiment	Experiment begins update, plants are updated or created	✓
Import data with valid csv	Click Init Data button on initialised experiment	Data is imported from csv	✓
Import data with invalid csv	Click Init Data button on initialised experiment	Invalid csv data is ignored	✓
Delete data	Click delete data on experiment	Data is deleted from the experiment	✓
Delete plants	Click delete plants button on experiment	Plant data and images are deleted	✓

Table 4.1: Test Table for Admin page functionality

4.3.2   Graph Page Test Table

Although most of the functionality within the Graph page is verified via automated testing, certain aspects require visual verification and as such a manual approach is taken to verify functionality within the page.

Test	Input	Expected Output	Pass
Test view graphs with no experiment	Go to /graphs with no selected experiment	No data' page is show with back button	✓
Test view graphs with experiment that has no data	Go to /graphs with experiment in session that has no data	No data' page is show with back button	✓
Test view graphs with experiment that has data	Go to /graphs with experiment in session that has data	Graph page is shown with graph creation options	✓
Test create graph	Click create graph button on /graphs page	A graph is displayed in the page with selected axis attributes	✓
Test box plot	Select 'Box' and create graph	Nodes in the graph are represented as box plots	✓
Test scatter plot	Select 'Scatter' and create graph	Nodes in the graph are represented as scatter plot	✓
Test swap axis	Click swap axis button	Selected axis attributes are swapped, x value becomes y value and vice versa	✓
Test plant results on graph node click	Click on or near a node in the graph	A clickable list of plants corresponding to the values of the clicked node appear in the page	✓
Test click result plant	Click on a plant link generated as result of clicking on a graph node	User is redirected to the detail page for the clicked plant link	✓

Table 4.2: Test Table for Graph page functionality

4.4 User Testing

When development was near completion, a small sample of volunteer test users were recruited to use the system and give feedback on usability and the system in general. An online form was provided with a number of questions and a section for general feedback the responses to which can be found in Appendix C

Following the user testing, a number of changes were implemented according to the feedback given. Namely, adding pagination controls to the bottom of the plants and plant detail pages for more convenient page navigation and fixing an overlooked issue on the plant detail graph page. If a plant has no attributes recorded against individual plant days then the page should make the user aware that graph generation is not possible and provide a means to return to the previous page. Prior to user testing the page was confusing and mostly blank in the event that no graphable data were available.

Chapter 5

Evaluation

5.1   Requirements

In terms of meeting the requirements as detailed in Section 1.2.1 the system is entirely successful. Each requirement is implemented in a fairly robust and well tested manner. In terms of the initial identification of gathering of requirements there was room for improvement. Essentially the requirements for the system were fairly straight forward to accomplish and did not provide a large degree of challenge or the opportunity for significantly interesting work. The reason for this was essentially a reaction to the change in project topic, these final requirements were fairly rushed into agreement and should have been discussed in more exhaustive detail with the stakeholders involved. However they were sufficient in defining a system that solved the problem objective.

5.2   Implemented System

5.2.1   Strengths

The delivered system is well tested and developed in a robust way with a focus on maintainability. The choice of framework and system architecture provide an environment which is scalable, expandable and highly portable.

The system allows users to easily browse through collections of plants and associated data for given experiments. Navigation through, and interaction with the system is simple and intuitive. Users are able to create graphs of data in a flexible way and are not limited by system imposed restrictions on the data due to implementation choices such as object typing and data structures.

A convention over configuration approach makes managing and administering the system simple and efficient whilst also providing the emans for more detailed manipulation of system parameters if required.

5.2.2 Weaknesses

5.2.3 Future Work and Improvements

The project brief was fairly open ended and the delivered system fairly narrow in scope, this leaves many avenues for potential improvement and expansion of the system, potential examples of which are discussed below :

5.2.3.1 Search and filtering

Further utility could be provided by the system if some extensive search and filtering functionality was implemented. Filtering the time series of images or plants in an experiment by attribute values, ranges etc would enhance the ability of the user to find specific information of interest or aid in inputting such specific data to particular entities within the system. Similarly, searching for plants or attribute values would be a helpful utility in both exploration of the experiment data and input of data to specific entities of interest within the system.

One approach which was partially investigated during the implementation was to use the Elasticsearch [3] integration provided by Spring. This implementation provides a means to index certain attributes and classes within the system to provide a convenient and fast solution to searching and filtering. The approach follows a similar principle to that implemented for the persistence layer, that is, the entity classes are annotated and a 'data repository' interface is used to query these entities, further details can be found in the Spring documentation [8]

5.2.3.2 Comparisons and statistical analysis

As the interesting results of experiments carried out at the NPPC are often statistical in nature it would be convenient to the scientists using the system to be able to conduct or view the results of simple statistical analysis alongside the visualisation of data provided by the delivered system. As discussed in previous sections, the scientists conducting experiments often analyse only very narrow ranges of data correlations that could be present in an experiment. Providing a correlation measure for graphed attributes could indicate to users of the system that interesting correlations exist in previously overlooked data.

When generating a graph visualisation from two attributes, a correlation measurement could be taken between the two attributes using the entire set of examples within the experiment. One such method would be to use Spearmans rank correlation coefficient [24] to display the correlations between attributes. There are various libraries available within the Java ecosystem that provide implementations of such statistical analysis algorithms and methods.

5.2.3.3 Machine learning

With the potential for large datasets of meaningful, well understood data that is often statistical in nature it is an ideal scenario to leverage some machine learning capability in order to produce potentially interesting correlations and conclusions from system data. The main obstacle to providing such a facility is the current lack of collected data within the context of experiments. If the system was used mroe extensively then the provision of machine learning would become more and

more feasible. Data for ongoing experiments recorded within the system could be used to provide the basis of predictions of certain traits and attributes which could, if sufficiently accurate, provide the scientists with the means to focus effort on the most interesting, promising or relevant subset of plants within an experiment.

5.2.3.4 Image processing and computer vision

The NPPC captures vast quantities of image data for each experiment. Different modalities and angles add to the richness of the images as a potential data source in themselves. There is a wide array of potential improvements to the system that could be made to take advantage of such data. The NPPC themselves are concerned with the leverage of computer vision approaches to image analysis with ongoing research into useful automated procedures such as the detection of specific growth stages in oats as developed by Boyle et al [14]. As a result there are automated processes that provide support to the deployment of image based processes such as segmentation of images to provide masks of plant pixel location within images.

The system could be extended to use popular image processing libraries such as ImageJ (integration of which was tested briefly during spikework and verified as viable) which can provide many useful functions such as pixel colour maps, counts and so forth which could be used to infer the health or age of a given plant, detect fungus and so on.

It is possible to invoke external programs using system calls from within the delivered system, it is not infeasible to run compatible experiments through detectors such as that developed by Boyle et al [14] in order to automate the detection of certain growth stages and record these results within the persistence layer of the system

5.2.3.5 Comprehensive user management

Within the delivered system there is only the concept of an admin user and generic users. Providing the means for users to register and create account opens up many interesting options for further development of the system. In providing individual user accounts it is possible to then extend the attribute system to allow a concept of ownership, for example if a user adds attributes to a plant then that user has ownership of that attribute and gains the ability to delete or edit the attribute. Other users could raise disputes against the attribute or provide a competing value for the same attribute, this would allow the system to score agreement on particular assessments of certain traits or characteristics as identified by the different stakeholders within an experiment. Providing an agreement score for attributes in such a manner would provide a more robust dataset for use in other applications such as machine learning or vision systems.

A popular approach to encouraging user participation within such systems is gamification (bringing elements of game-design into non-game contexts), a system where users can gain points or extended privileges based on participation could be used to encourage scientists to add data and utilise the system more fully.

5.3   Process

It is believed that an agile approach as chosen for this project was the correct choice and that the process followed during the course of this project was successful for a number of reasons. It provided an environment in which work could be effectively prioritised based on emergent or short term needs and goals without having to reorganise more than the current day or iterations work. Being able to react to change and bugs in this way made development more comfortable especially when requirements were not concrete and the domain was not fully understood at the beginning of iterations. Other approaches may not offer such a degree of flexibility.

The use of user stories and the ability to move them into and out of the scope of iterations along with progressing their status made the project work flow more easily and allowed the focus to be on the goal of an iteration and the next release rather than a generic list of un-organised tasks. As discussed in Section 1.3, the estimation of tasks in terms of time could have been improved to provide a more complete understanding of developer throughput which would have led to more efficient allocation of pomodoros to the correct tasks. However, estimation is notoriously difficult to do with a high degree of accuracy especially when working with technologies that developers aren't fully experienced in using.

A key part of the process followed is the incremental and frequent release of the product. Providing a deployment at the end of each iteration offered a source of reassurance to stakeholders that work was progressing and also provided, via the use of version control facilities, the opportunity to rollback if a release was broken or problematic.

The use of a blog to keep up-to-date with project work was a great benefit, especially during the writing of this report where it served as a reminder of specific implementation or design details. Having to write a blog post acted as a source of accountability, encouraging the hour or so of extra work it would take to make a worthy post.

5.4   Student Evaluation

There are a number of areas in which personal improvement has been made during the course of delivering this project and a number of skills have been gained or enhanced as a direct result of the work undertaken. These will be discussed in this section as well as a general evaluation of performance and approach during the course of the work.

In terms of preparation for the project there are two facets worthy of discussion. Firstly the provision of supporting applications and technical environment in which to conduct the work was well chosen and provided a great deal of utility and efficiency when designing, developing and testing components of the project. Use of testing frameworks and continuous integration provided a continuous assurance that errors would be found during the development process which provided more time to focus on design and implementation rather than manual testing and verification. Secondly, the attitude towards resolving unknowns in the discovery phase of the project could have affected the general results in a fairly negative way, a key lesson was learned as a result and that is a more pro-active approach early in a project is almost always going to be preferable and cheaper in terms of effort in the long run.

In evaluating potential technologies and designing the system, experience has been gained in a wide range of useful applications and frameworks. Experience gained in Spring and supporting

technologies is of direct relevance to post-graduation employment and many of the skills acquired are desirable in many industries. Evidence of automated testing and appreciation of stress testing frameworks are in-demand and widely transferable skills within the software industry.

The project saw opportunities to practice many skills and apply knowledge that has been gained during the course of previous university study, notably project methodology choices and good software engineering practises such as a focus on developing for maintainability and scalability. The experienced gained in development using the Java and Javascript languages during my time in university provided a solid foundation of understanding that allowed development time to focus on new technologies such as framework specific implementations as opposed to the core technologies.

The nature of the project and its concurrency with certain external events, along with the nature of the student, has provided a unique opportunity to appreciate stress management in ways that have not been required or considered previously.

If the project were to be repeated the biggest difference would be the integration with the staff at the NPPC, being more proactive in this regard would have ensured that all the involved stakeholders would have a better understanding of the project and the expected output. Implementing the pomodoro technique from the beginning would have resulted in higher productivity. It would have been a much more interesting project had the focus been on images as data rather than something to display. In general the approach to development and design would remain consistent but differences in personal attitude and outlook would have a significant effect on execution of the project.

A key and unexpected take away from this project is what I learned about myself personally, rather than in any professional or academic sense and I am grateful for the opportunity to learn this lesson.

Appendix A

Third-Party Code and Libraries

plotly jquery bootstrap spring? opencsv

Appendix B

Ethics Submission

AU Status
Undergraduate or PG Taught

Your aber.ac.uk email address
sig2@aber.ac.uk

Full Name
sion Griffiths

Please enter the name of the person responsible for reviewing your assessment.
Reyer Zwiggelaar

Please enter the aber.ac.uk email address of the person responsible for reviewing your application
rrz@aber.ac.uk

Supervisor or Institute Director of Research Department
cs

Module code (Only enter if you have been asked to do so)
CS39440

Proposed Study Title
MMP Visualising Plants and Metadata

Proposed Start Date
25th Jan 2016

Proposed Completion Date
4th May 2016

Are you conducting a quantitative or qualitative research project?
Quantitative

Does your research require external ethical approval under the Health Research Authority?
No

Does your research involve animals?
No

Does your research involve human participants?
No

Are you completing this form for your own research?
Yes

Does your research involve human participants?
Yes

Institute
IMPACS

Please provide a brief summary of your project (150 word max)
Building an interactive web based system for visualising experiment data and images of plants provided by the National Plant Phenomics Centre. Human participants will only be involved in anonymous usability studies.

I can confirm that the study does not involve vulnerable participants including participants under the age of 18, those with learning/communication or associated difficulties or those that are otherwise unable to provide informed consent?

Yes

I can confirm that the participants will not be asked to take part in the study without their consent or knowledge at the time and participants will be fully informed of the purpose of the research (including what data will be gathered and how it shall be used during and after the study). Participants will also be given time to consider whether they wish to take part in the study and be given the right to withdraw at any given time.
Yes

I can confirm that there is no risk that the nature of the research topic might lead to disclosures from the participant concerning their own involvement in illegal activities or other activities that represent a risk to themselves or others (e.g. sexual activity, drug use or professional misconduct). Should a disclosure be made, you should be aware of your responsibilities and boundaries as a researcher and be aware of whom to contact should the need arise (i.e. your supervisor).
Yes

I can confirm that the study will not induce stress, anxiety, lead to humiliation or cause harm or any other negative consequences beyond the risks encountered in the participant's day-to-day lives.
Yes

Please include any further relevant information for this section here:
Anonymous usability study is the only area with human participants

Where appropriate, do you have consent for the publication, reproduction or use of any unpublished material?
Yes

Will appropriate measures be put in place for the secure and confidential storage of data?
Yes

Does the research pose more than minimal and predictable risk to the researcher?
Not applicable

Will you be travelling, as a foreign national, in to any areas that the UK Foreign and Commonwealth Office advise against travel to?
No

Please include any further relevant information for this section here:

If you are to be working alone with vulnerable people or children, you may need a DBS (CRB) check. Tick to confirm that you will ensure you comply with this requirement should you identify that you require one.
Yes

Declaration: Please tick to confirm that you have completed this form to the best of your knowledge and that you will inform your department should the proposal significantly change.
Yes

Please include any further relevant information for this section here:

Appendix C

User testing results

Introduction

Thank you for agreeing to participate in my user evaluation study for my undergraduate dissertation project: **Visulising plants and metadata.** This is for the module CS39440 Major/Minor Project as part of my **BEng Software Engineering** degree at Aberystwyth University.

The aim of this study is **to provide usability testing feedback for the site developed as part of this project.**

Confidentiality

This is an anonymous study and no information about your identity will be kept. The data will be held **within my personal Google Drive account** and I, **Siôn Griffiths,** am the only person with the password to the account that has permission to access the data.

The data that is gathered during this study will be analysed and used in relevant sections of my dissertation. Copies of the raw response, without any analysis, will be provided in an appendix in my report, as evidence that I have conducted this study. However, the data will be checked and anonymised if necessary before it is included in the report.

The reports are read by members of staff in the University as part of the assessment process. Typically, this will mean approximately 3 members of staff. An external examiner from another University might also read the report. Further, the department keeps reports for approximately 1 year and might be shared with other students and members of staff as an example of work from this year.

The original data completed by participants will be deleted when the module result is released in June/July 2015.

Agreement

Please note that any data you provide does not enjoy legal privilege, i.e. cannot be used in a court of law.

In continuing to complete and submit the form on the next page, you agree to the above terms about the storage and processing of the data.

Figure C.1: Disclaimer form as shown to user testing participants

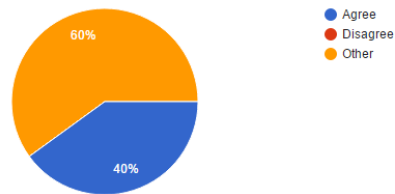It is easy and intuitive to navigate between experiments (5 responses)

Figure C.2: User testing results for question 1

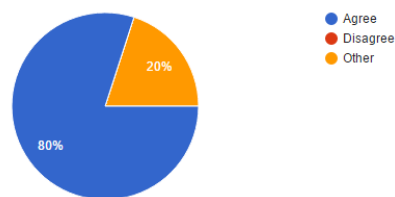It's easy to add data or tags to plants (5 responses)

Figure C.3: User testing results for question 2

Pagination options are easy to understand and use (5 responses)
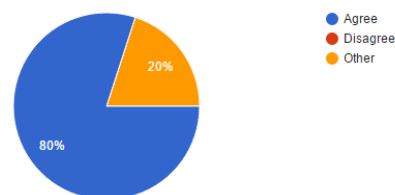

Figure C.4: User testing results for question 3
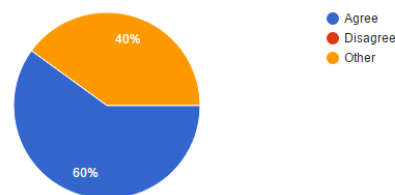
Creating graphs is simple (5 responses)

Figure C.5: User testing results for question 4

Navigating the different plant images is intuitive and simple (5 responses)
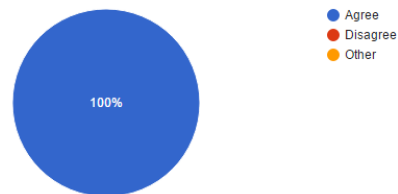

Figure C.6: User testing results for question 5

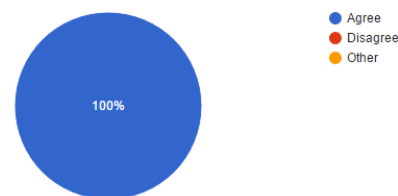The performance of the system is acceptable (5 responses)

Figure C.7: User testing results for question 6

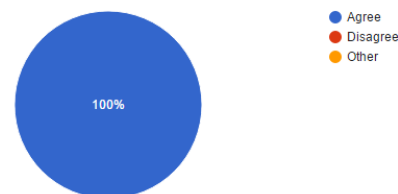There are no obvious bugs or crashes (5 responses)

Figure C.8: User testing results for question 7

Annotated Bibliography

[1] "Apache JMeter - Apache JMeter." [Online]. Available: http://jmeter.apache.org/

> An open-source Java based performance and load testing tool originally designed for web applications.

[2] "Australian Plant Phenomics Facility | Australian Plant Phenomics Facility." [Online]. Available: http://plantphenomics.org.au/

[3] "Elasticsearch." [Online]. Available: https://www.elastic.co/products/elasticsearch

[4] "Git." [Online]. Available: https://git-scm.com/

> Git version control system page

[5] "Hibernate ORM - Hibernate ORM." [Online]. Available: http://hibernate.org/orm/

> Hibernate object relational mapping system

[6] "IntelliJ IDEA the Java IDE." [Online]. Available: https://www.jetbrains.com/idea/

> The IntelliJ Java IDE

[7] "plotly." [Online]. Available: https://plot.ly/javascript/

> Plotly.js, an opensource javascript charting library

[8] "Spring Data Elasticsearch." [Online]. Available: http://docs.spring.io/spring-data/elasticsearch/docs/current/reference/html/

[9] "spring-projects/spring-boot." [Online]. Available: https://github.com/spring-projects/spring-boot

[10] "Travis CI User Documentation." [Online]. Available: https://docs.travis-ci.com/

> Travic continnuous integration documentation

[11] "Zegami." [Online]. Available: http://zegami.com/

[12] "What is Docker?" May 2015. [Online]. Available: https://www.docker.com/what-docker

> Docker continuous integration system overview

[13] Atlassian, "JIRA Software - Issue & Project Tracking for Software Teams." [Online]. Available: https://www.atlassian.com/software/jira

Homepage for the Jira issue tracking system

[14] R. Boyle, F. Corke, and C. Howarth, "Image-based estimation of oat panicle development using local texture patterns," *Functional Plant Biology*, vol. 42, no. 5, p. 433, 2015. [Online]. Available: http://www.publish.csiro.au/?paper=FP14056

Paper detailing a technique used to detect oat panicles via computer vision techniques. Development of panicles can be directly corrolated with certain growth stage (around GS55) in oats

[15] F. Cirillo, *The pomodoro technique*.   Simon and Schuster, 2014.

[16] "Codeship," Codeship Inc. [Online]. Available: https://codeship.com/

A continuous integration tool which can hook into other online resources such as GitHub

[17] W. Commons. (2012) Skala bbch. [Online]. Available: https://upload.wikimedia.org/wikipedia/commons/3/3e/BBCH_zbo%C5%BCa.svg

[18] "Build software better, together," GitHub, Inc. [Online]. Available: https://github.com

An online Git repository hosting service

[19] D. Kendal, C. E. Hauser, G. E. Garrard, S. Jellinek, K. M. Giljohann, and J. L. Moore, "Quantifying Plant Colour and Colour Difference as Perceived by Humans Using Digital Images," *PLoS ONE*, vol. 8, no. 8, p. e72296, Aug. 2013. [Online]. Available: http://dx.doi.org/10.1371/journal.pone.0072296

Paper detailing how a humans perception of colour in images of plants can affect judgements made about these images.

[20] M. N. Merzlyak, A. A. Gitelson, O. B. Chivkunova, and V. Y. Rakitin, "Non-destructive optical detection of pigment changes during leaf senescence and fruit ripening," *Physiologia Plantarum*, vol. 106, no. 1, pp. 135–141, May 1999. [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1034/j.1399-3054.1999.106119.x/abstract

Paper detailing senescence detection in plant images by analysis of colour

[21] "National Plant Phenomics Centre," National Plant Phenomics Centre. [Online]. Available: http://www.plant-phenomics.ac.uk/en

National Plant Phenomics Centre

[22] J. Nielsen, "Response times: the three important limits," 1994.

Article discussing tollerable wait times for web page loads

[23] J. R. Quinlan, "Induction of Decision Trees," *Mach Learn*, vol. 1, no. 1, pp. 81–106, Mar. 1986. [Online]. Available: http://link.springer.com/article/10.1023/A%3A1022643204877

Paper detailing the ID3 decision tree algorithm

[24] C. Spearman, "The Proof and Measurement of Association between Two Things," *The American Journal of Psychology*, vol. 15, no. 1, pp. 72–101, 1904. [Online]. Available: http://www.jstor.org/stable/1412159

[25] J. M. Tanner, R. H. Whitehouse, W. A. Marshall, M. J. R. Healty, and H. Goldstein, "Assessment of Skeleton Maturity and Maturity and Prediction of Adult Height (TW2 Method)," 1975. [Online]. Available: http://core.tdar.org/document/125299

> Paper detailing the atlas approach used in this instance to predict adult height in human's from skeletal features in children

[26] G. W. Williams, "Comparing the Joint Agreement of Several Raters with Another Rater," *Biometrics*, vol. 32, no. 3, pp. 619–627, 1976. [Online]. Available: http://www.jstor.org/stable/2529750

> A paper describing the comparison of expert opinion with that of other experts or a group of experts

[27] J. C. Zadoks, T. T. Chang, C. F. Konzak, and others, "A decimal code for the growth stages of cereals," *Weed res*, vol. 14, no. 6, pp. 415–421, 1974. [Online]. Available: http://old.ibpdev.net/sites/default/files/zadoks_scale_1974.pdf

> Paper detailing the decimal growth stages of cereal plants