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

Nppc plants. fuck.

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 [20] 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 [22] 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
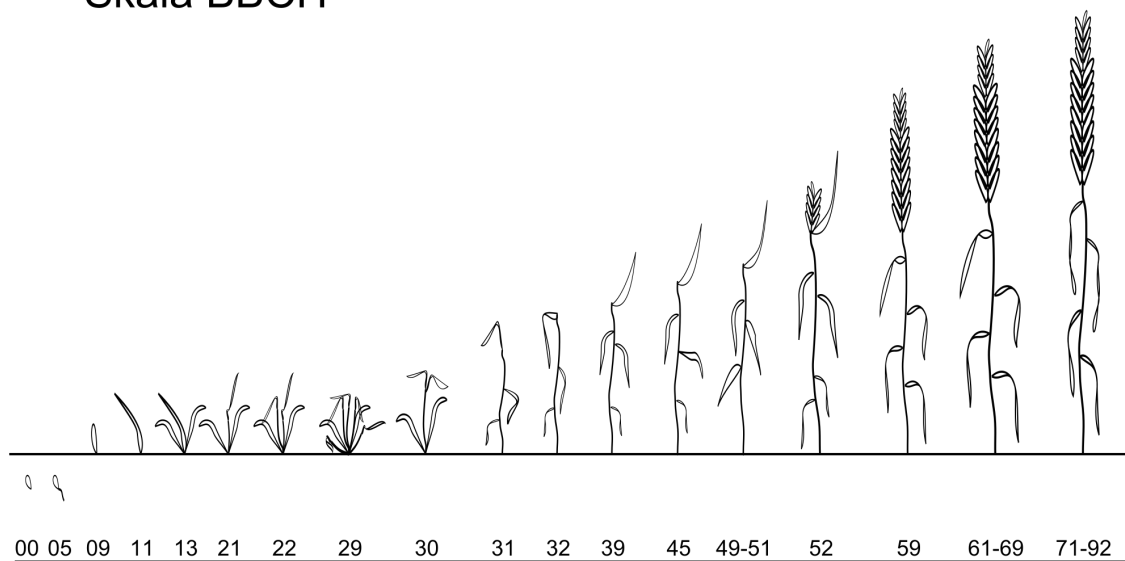
Figure 1.1: Example of 'atlas' style visualisation of wheat plant development stages. Source : Wikimedia Commons [13]

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.

1.1.3   Change of project topic

Challenges of interdisciplinary collaboration, managing expectations on both sides data data data

1.1.4   Existing solutions

1.2   Analysis

1.2.1   Requirements

- **Web based system**

- **Integrate with NPPC data repository**

- **Browse plants and plant images**

- **Add meta data to plants and plant images**

- **Import experiment data**

- **Display graphs of data in system**

- **Administrator panel or page to manage the system**

1.2.2 User Roles

For this system there were two identified user roles
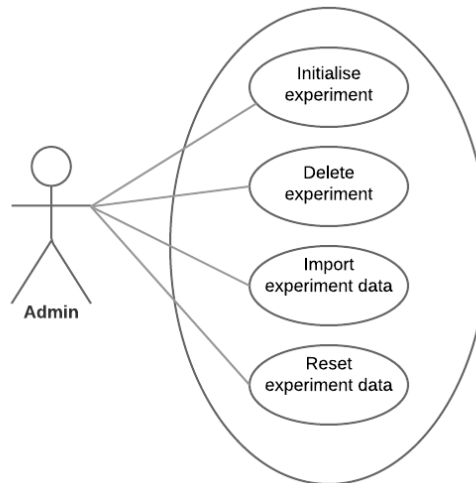
Figure 1.2: Use cases for admin user
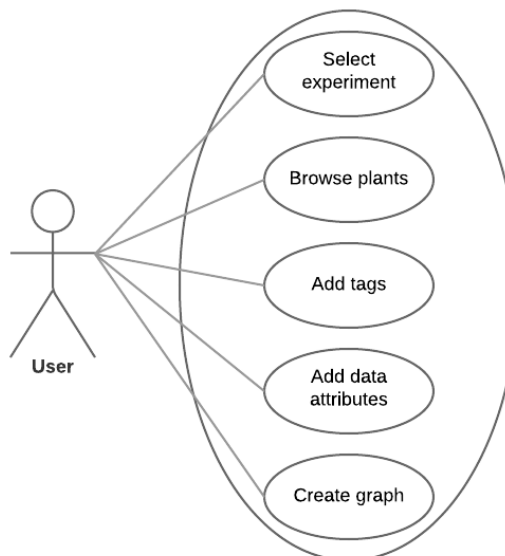
Figure 1.3: Use cases for generic user

Zegami, the site @ access.plant-phenoimics

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 to 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 [9] 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 [11], 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 1.4 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.
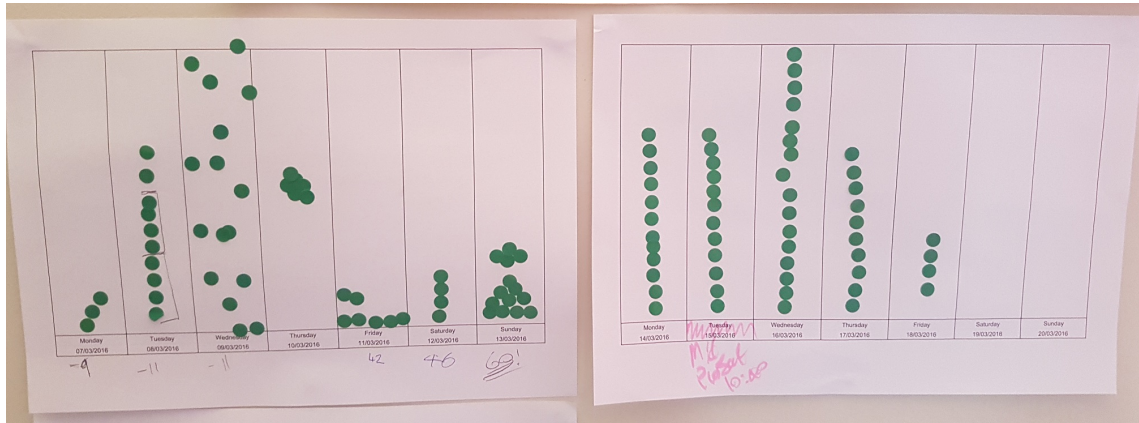
Figure 1.4: Tracking pomodoros

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, seperation 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.
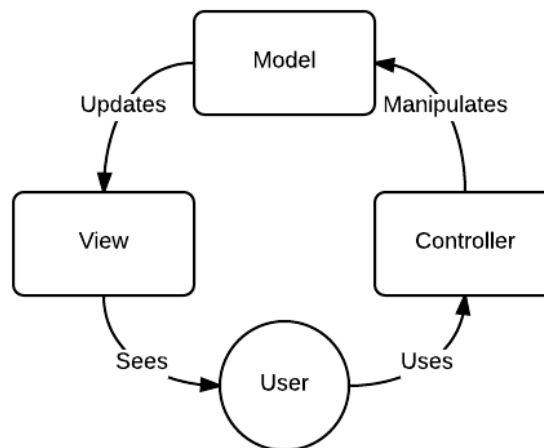
Figure 2.1: Model View Controller pattern overview

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.
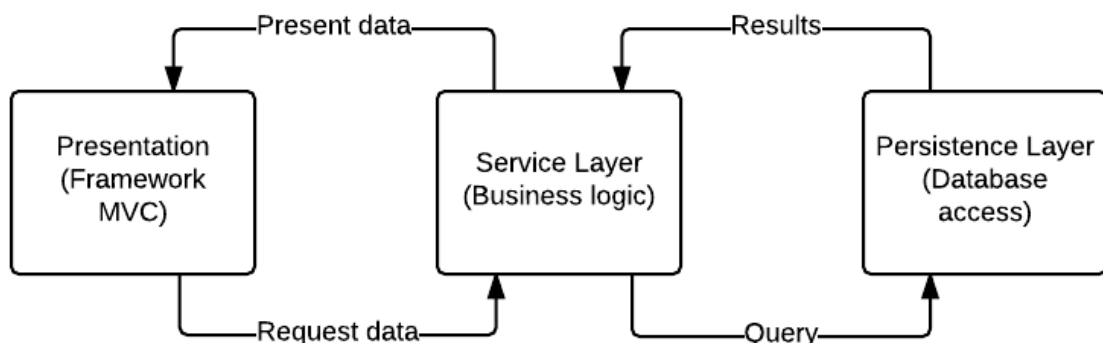
Figure 2.2: 3-tier architecture overview

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 with 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.
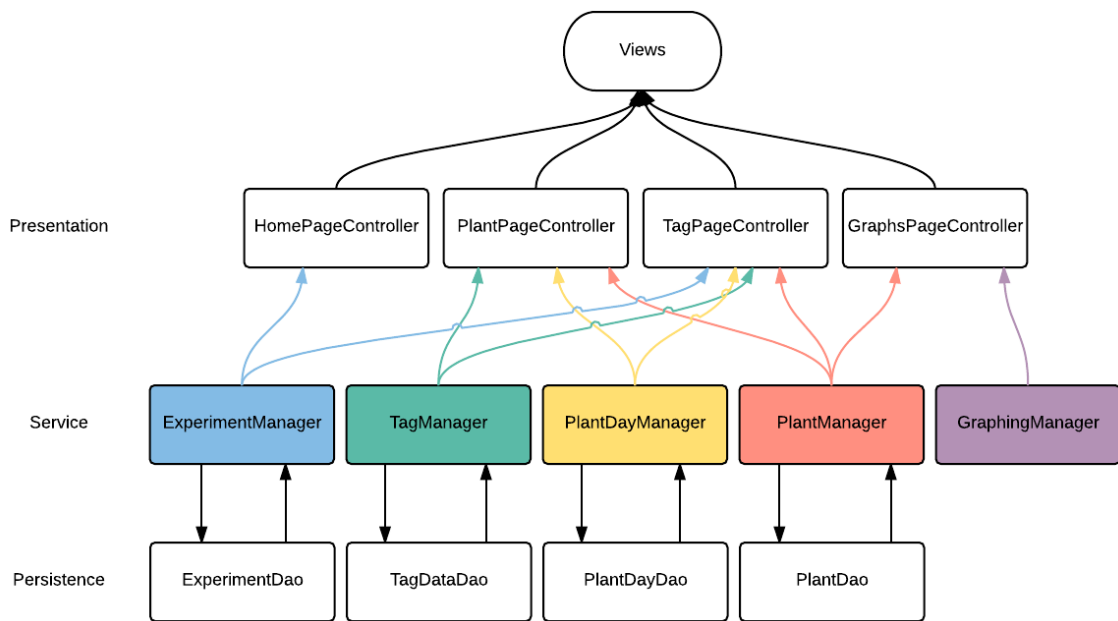

Figure 2.3: Architecture and dependency wiring

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 [6] 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.
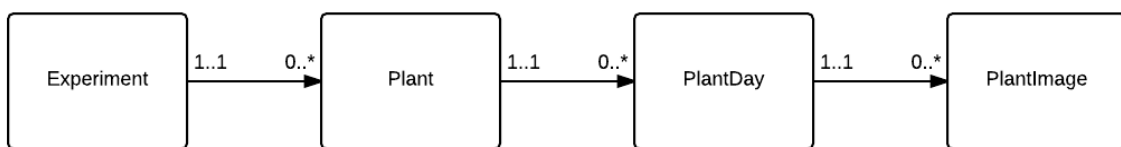
Figure 2.4: A simplified class diagram showing the relationship between primary domain entities

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.
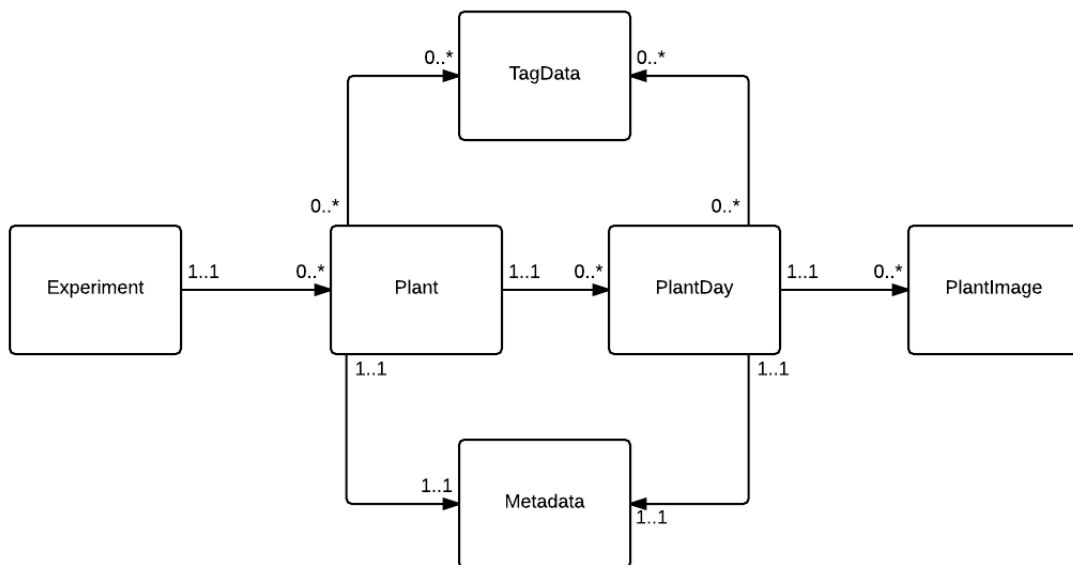

Figure 2.5: A simplified class diagram showing the domain model

2.4 Database

For this project the database structure is entirely derived by the Hibernate [3] 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.
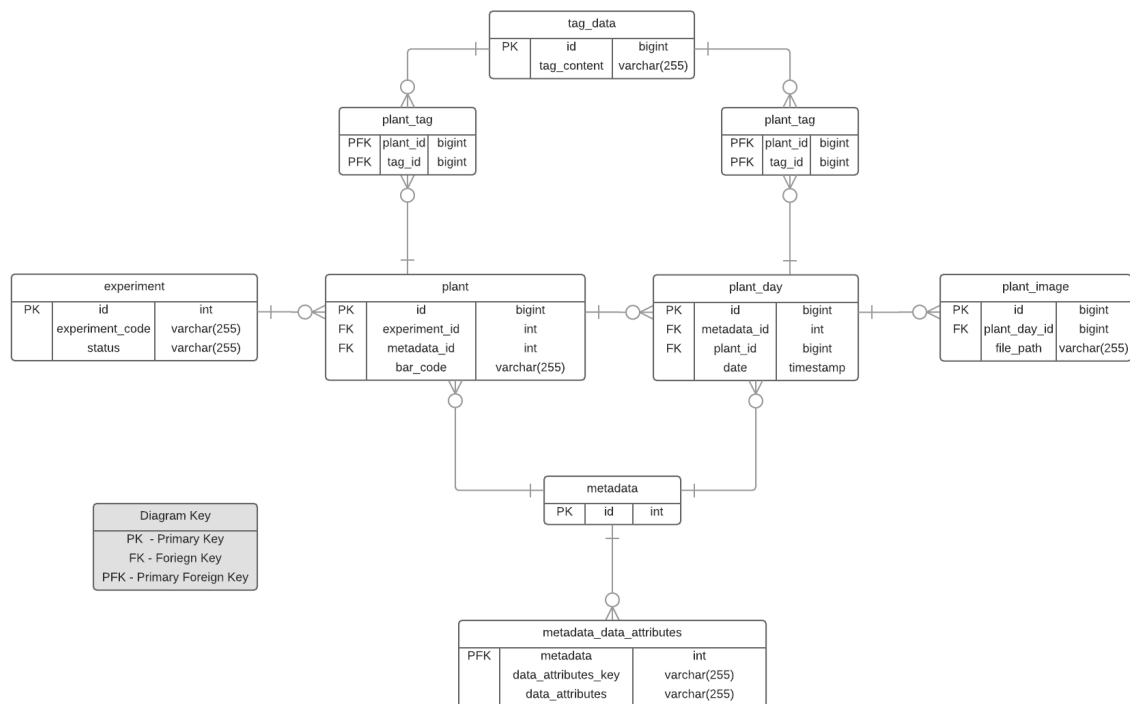

Figure 2.6: An entity relationship diagram for the database schema

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.
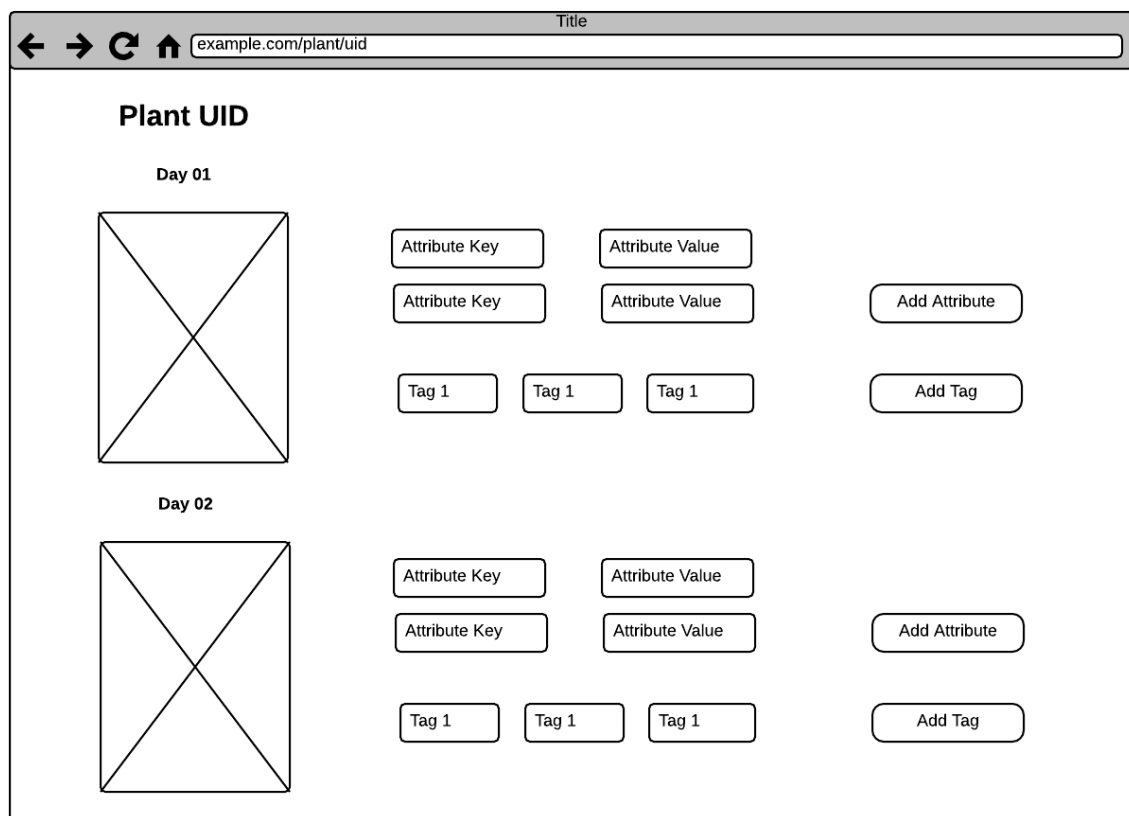
Figure 2.8: An early wireframe design for the Plant Details page

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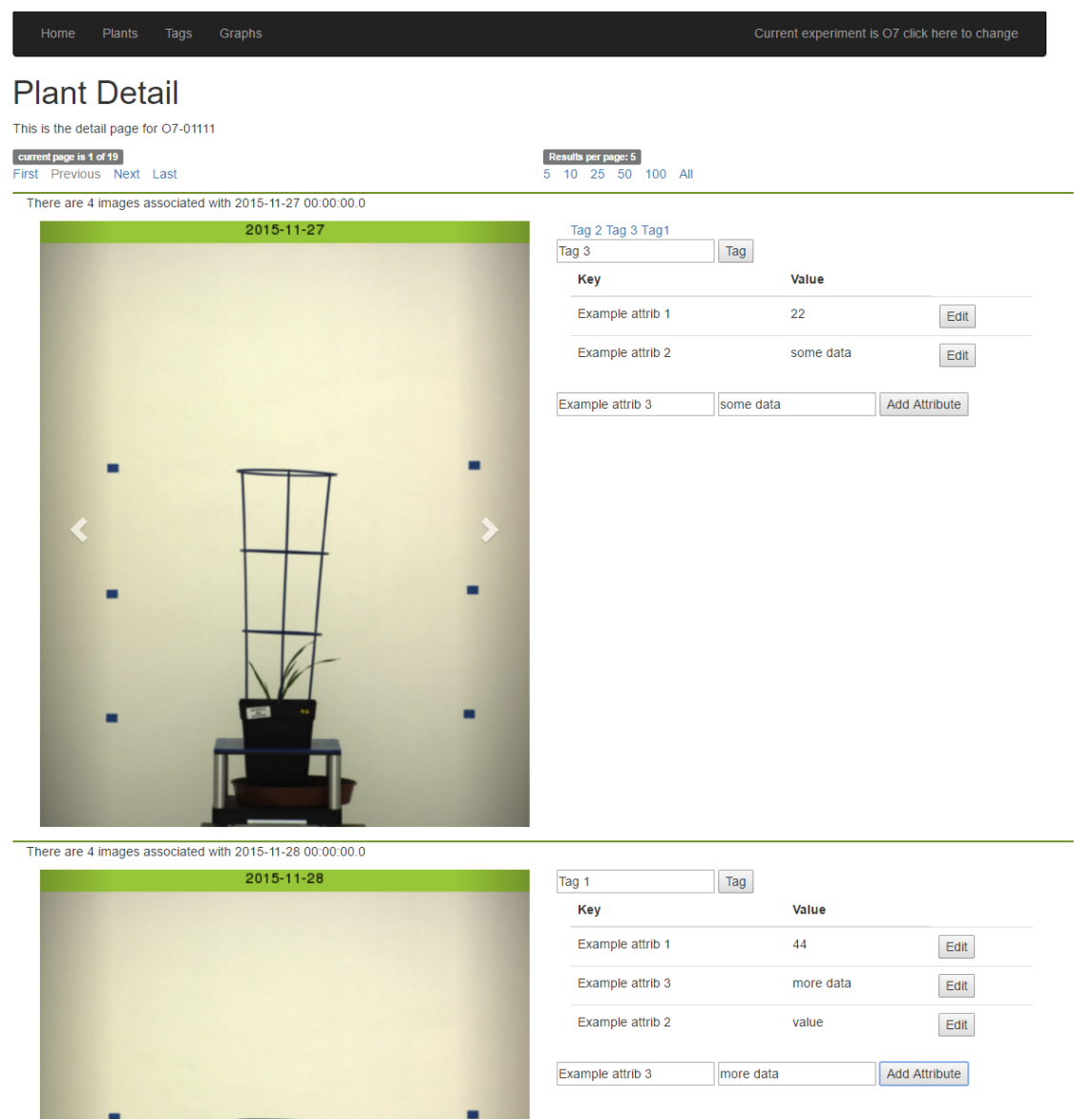
Figure 2.9: A screen shot showing final design of Plant Details page

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 [4] 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.
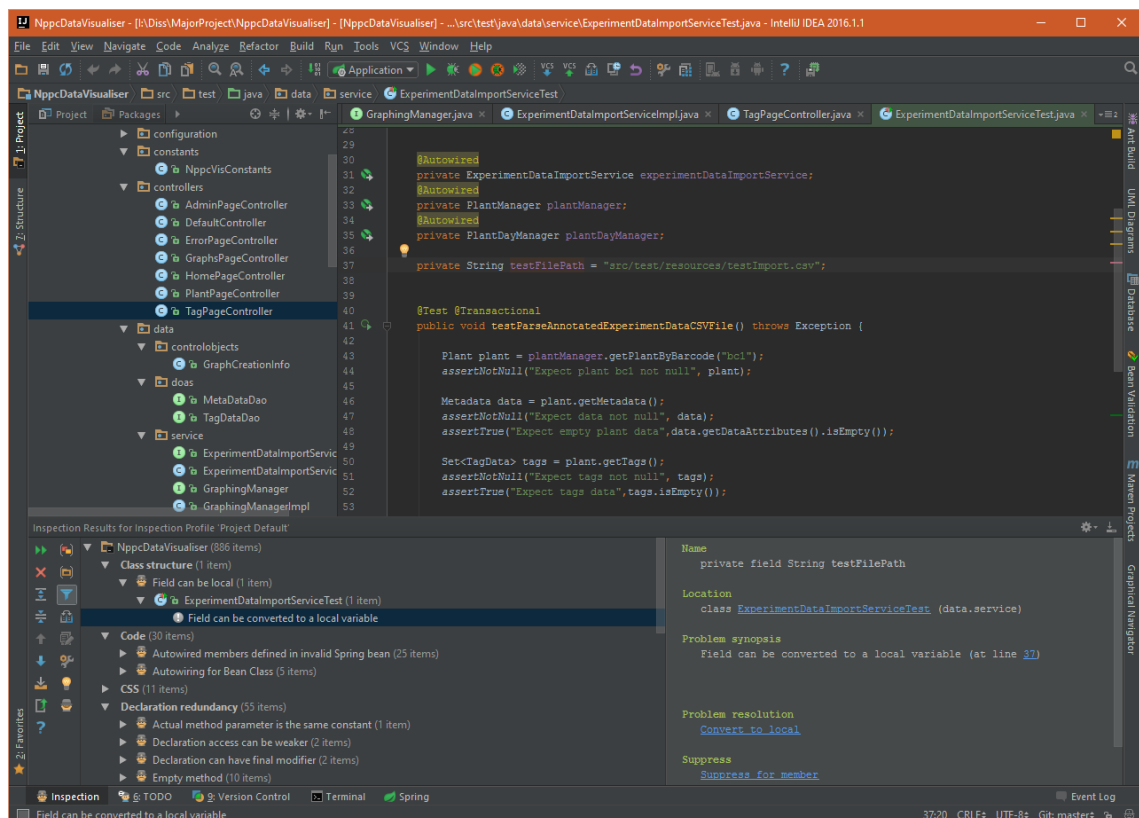

Figure 2.10: The IntelliJ IDE showing result of static analysis

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 [2], 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 [14], 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.
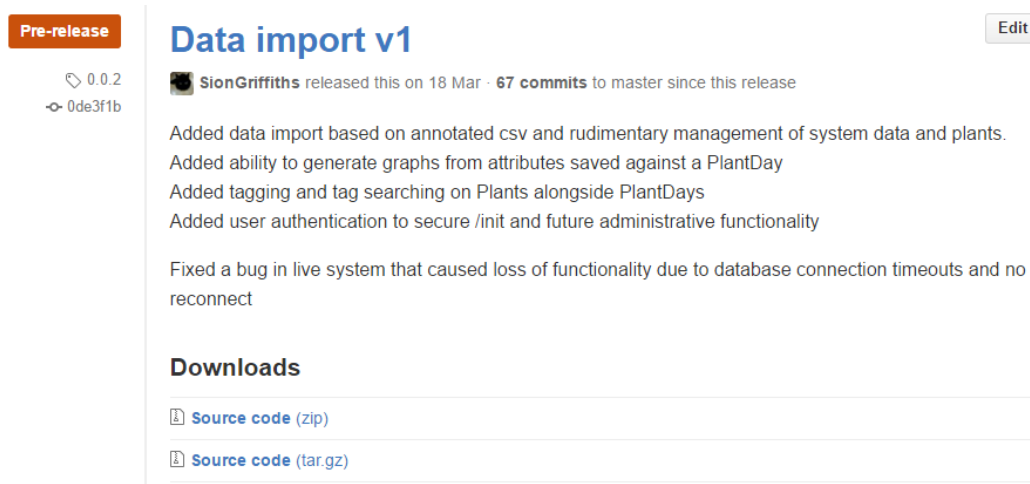

Figure 2.11: A tagged release of the project within Github

2.7.3 Jira

Jira [9] 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.
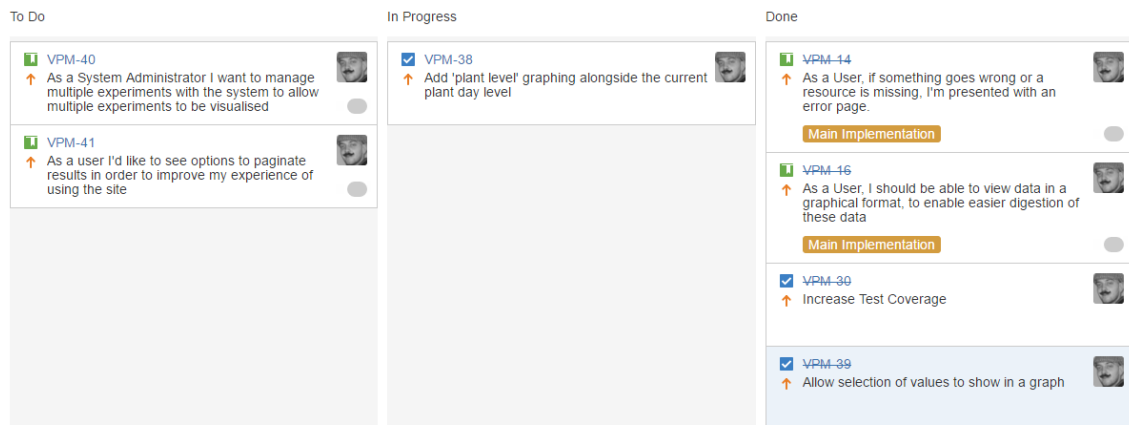
Figure 2.12: Current sprint screen in Jira

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 usgent 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 [12] 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 [8] 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 [7] 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 [5] 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]
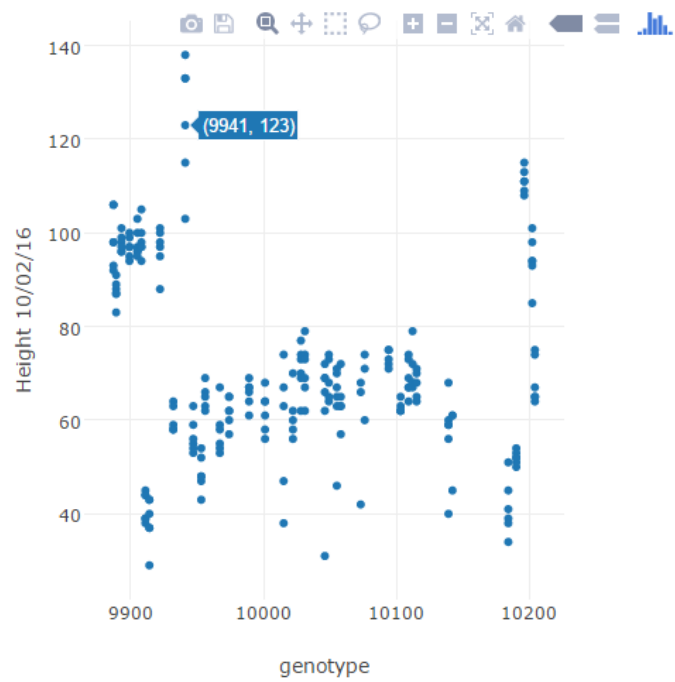
Figure 2.17: Project graph page featuring a Plotly.js generated graph

Chapter 3

Implementation

3.1   Integration with NPPC data repository

Fairly straight forward, recursive methods slow, iterative and tightly coupled fast The structure of
the data repository changed soon after implementation was begun

3.2   Graphing System

Wrapping things into nice JSON format for plotly, fun with join queries.

3.3   Domain model implementation and ORM

Talk about looping references and stack overflows? Laze/eager fetch?

3.4   Data Import

Use of the arrays to hold column values "hashing" the values into the right place using the numbers
in the list

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 environemnt, 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.

The provided hosting environment was provisioned and set up by Dr Colin Sauze, the data manager at the NPPC. The hosting environemnt 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. Sine 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 likely would 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

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 TTD 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.

```
1     @Test
2     public void resetTagsForExperiment()  {
3         Long id = 10L;
4         Plant plant = plantManager.getPlantByID(id);
5         Experiment experiment = plant.getExperiment();
6
7         assertEquals("Expected number of tags to be 2", 2 ,
              plant.getTags().size());
8
9         plantManager.resetTagsForExperiment(experiment);
10        assertEquals("Expected number of tags to be 0", 0 ,
              plant.getTags().size());
11    }
```

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 was 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.

```
1  @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 [18]. 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 4.3 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.
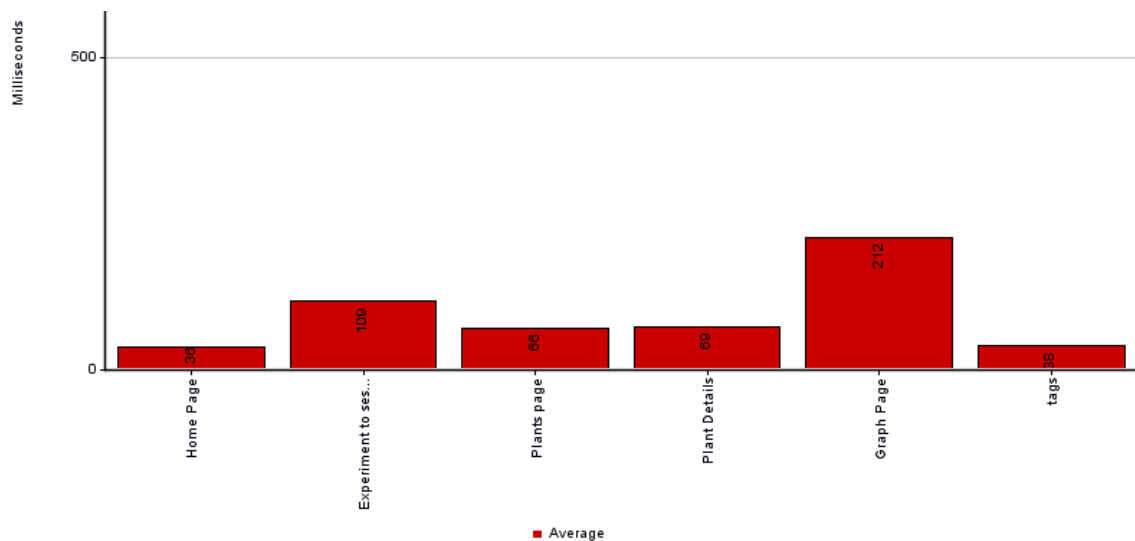
Figure 4.2: Visulisation of Jmeter test result of a fully initialised experiment
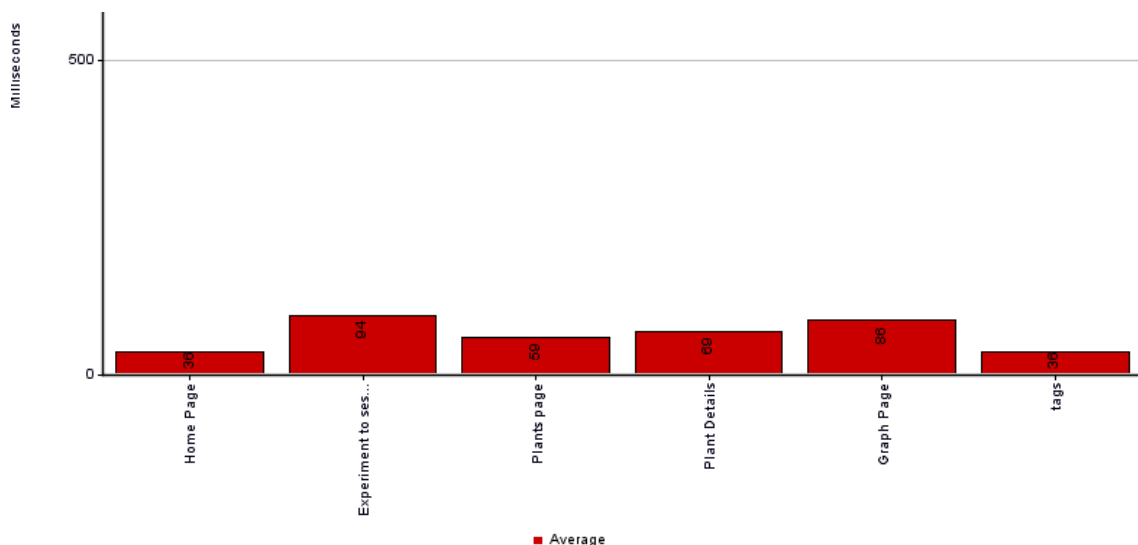
Figure 4.3: Visulisation of Jmeter test result of a partially initialised experiment

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 is 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 is 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 complete 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

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 was available.

Chapter 5

Evaluation

Examiners expect to find in your dissertation a section addressing such questions as:

5.1 Requirements

5.2 Implemented System

5.2.1 Strengths

5.2.2 Weaknesses

5.2.3 Future Work and Improvements

5.3 Process

5.4 Student Performance

Appendix A

Third-Party Code and Libraries

plotly jquery bootstrap spring?

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

Code Examples

3.1   Random Number Generator

The Bayes Durham Shuffle ensures that the psuedo random numbers used in the simulation are further shuffled, ensuring minimal correlation between subsequent random outputs [**?**].

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

    Git version control system page

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

    Hibernate object relational mapping system

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

    The IntelliJ Java IDE

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

    Plotly.js, an opensource javascript charting library

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

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

    Travic continnuous integration documentation

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

    Docker continuous integration system overview

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

    Homepage for the Jira issue tracking system

[10] 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

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

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

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

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

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

    An online Git repository hosting service

[15] 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.

[16] 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

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

    National Plant Phenomics Centre

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

    Article discussing tollerable wait times for web page loads

[19] 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

[20] 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

[21] 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

[22] 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