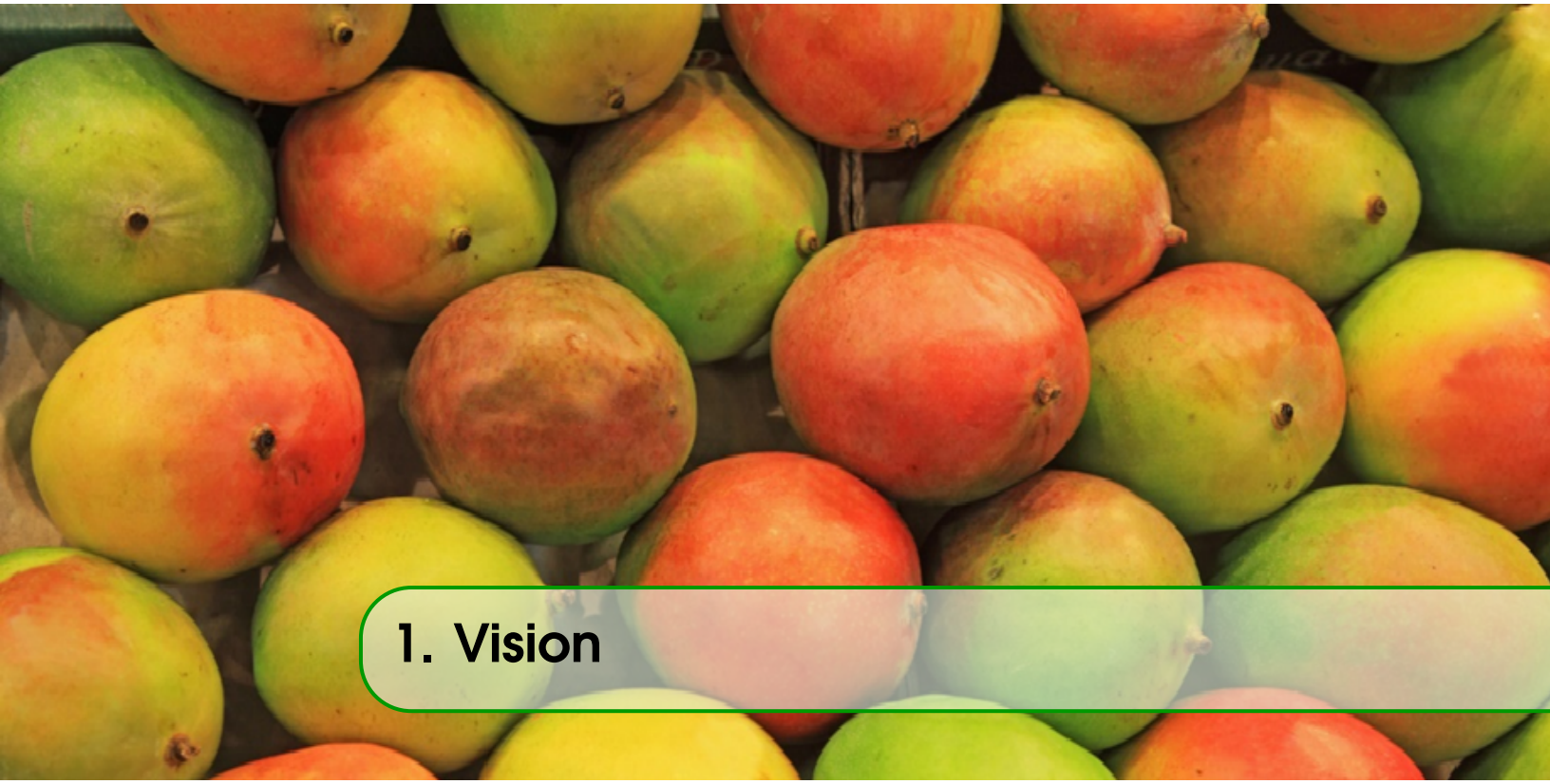
Harvest

Software Architectural Requirements and Design

HTTP_418

Christiaan Saaiman, 12059138
Michael Loosen, 14017254
Elizabeth Bode, 14310156
LC Meyers, 14024633

Contents



1. Vision

The client, Subtrop, has requested that we implement a system to keep track of the crop yields of farms registered with them. The scope of the system entails managing the administration involved in the tracking of yields and the performance of the farm workers. Currently, this is done by manual tracking on a paper-based system. The system is required to keep track of all yield data and the associated farms metadata.

The current paper-based system consists of various farmers, their farms, the foreman they have appointed, their workers that work under the foreman and the different Subtrop crop type associations. Farms are divided into orchard blocks with each block having one crop type associated with it. Workers are paid according to their performance and, thus, this data has to be recorded. These two factors provide the potential for statistical data reports, which could aid farmers in strategic decision-making.

The most fundamental requirements that the client would like the system to fulfil are:

- Keeping track of yield data in order to see which orchards are successful
- Keeping track of worker performance in order to ensure that they are working to their full potential

2. Background

- Limitations of current system

  The current system being used by the client involves manually tracking worker performance with the use of a clipboard, with a list of all the workers names, and a pen. The farmers have to manually record every yield that each worker collects in order to track their performance to determine their wages. The current system is inefficient due to the tedious nature of it. This tediousness reduces the foremans productivity as time is wasted unnecessarily searching for workers in the list and keeping track of all the lists chronologically. As the foreman is in charge of the workers, the workers have to wait until the yield they collected has been recorded which in turn also reduces their productivity.

- High theft issue

  Farmers in the subtropical growers industry are continuously burdened by theft of their products by their own employees. This results in decreased revenue, increased costs, reduced productivity and a decrease in employer-employee trust, which creates the potential for a poor working environment. All of which necessitates the development of a new system that seeks to reduce the probability of crime by implementing monitoring on the foremans devices.

- Need for statistical data

  The potential for generating statistical data that could aid farmers in future decisions from the manual data they collect is immense. The current paper-based storage of data prevents the generation of statistical reports due to the immense workload and time it requires. There is a huge need for reports indicating the highest producing orchard blocks, appropriate allocation of orchard block conditions to crop types and expected revenue. All of which could assist the farmers in future strategic planning.

3. Architecture Requirements

3.1 Architectural Scope

We at HTTP_418 have taken a philosophy of quality work. Therefore we have decided to create an application that is centralised around a few core ideas. This ideology will drive us to create an application for Subtrop as best as we can.

We will work in a persistence framework because we recognise our clients need to have accurate, reliable data on hand when necessary. Our framework relies on databases, with integration for any database, meaning that if we wish to switch from a SQL to a NoSQL database, we will be able to do so. Alongside this, our design will incorporate security with regards to accessing the data stored. We will also be making use of techniques and technologies that will allow us to persistently add and remove data with concurrent accesses to the database.

Our system will also feature a reporting aspect that will be characterised as central, independant, consistent accurate as well as timely reliable. The independence is a key feature such that a report can still be generated in the event that the rest of the system is experiencing problems.

Our system will be accessible on the client-side via a desktop web client as well as an application on Android Mobile. The web client will have greater functionality than the Android application as it will serve as an admin/control interface where the user is able to manage the system, update details and view statistics. The Android Application will merely provide basic functionality for data input and throughput.

The system will also require real-time updates. As the gatherers hand in their pickings and the foreman enters them on the application, the farmer must be able to view/track the changes from his web client.

Our system is also be required to produce heat maps. These are graphical representations of the yield that a certain area delivers. They are created using the data stored in our database.

Our system will be used in excess of 900 farmers and a much greater number of foreman, meaning that there will be a great emphasis on scalability, performance and reliability.

Lastly we would like to create the system in such a way that a user is able to add and remove modules

without it necessary to shut the system down.

3.2 Access Channel Requirements

The access channels can be divided into two sections, namely Human and System.

3.2.1 Human Access Channels

The user will be able to access the system through two channels, namely a web client interface and an Android application interface.
The web client interface will be strictly reserved for the farmers with which they can interact with the system in a superuser sense. The farmer will be able to assign workers to each foreman and in real time track the progress of his harvest, each workers statistics, the foremans location as well as view statistics on his harvests and view a heat map.
The Android application will have limited functionality. It will be used by the foremans while they are out with the harvesters to keep track of the amount of bags that each worker brings in. It will be made available on the Google PlayStore and will run any device running Android 4.0 or higher.

3.2.2 System Access Channels

Our framework that we chose, namely SailsJS makes RESTful architecture easy and simple. Sails uses the RESTful API to already REST wrap all our objects. Along with that, SailsJS features Waterline, an intuitive interface for any database we would like to use, from MongoDB to PostgreSQL.

3.3 Quality Requirements

There are many quality requirements for an application such as ours. Here we will look at them and in short discuss them.

3.3.1 Performance

Description

Application performance is measured as amount of work done within a certain time period. Our job as developers is to get as much done as quickly possible without sacrificing integrity, security and Reliability.

Justification

The more work can be accomplished by the usage of this application, the more productive farmers can be in other areas and the better they can use the data to determine where to apply fertiliser etc.

Requirements

- Network requests must be benchmarked, monitored and aggregate statistics about network requests must be made available.
- Function calls must be timed and benchmarked and this data should be logged.
- Network responses should be cached on server side to lighten the load on database as well as decreasing the round trip time of request - response.

3.3.2 Reliability

Description

The system we are developing must be reliable in its data persistence and accessibility. System crashes and data loss is not acceptable in any circumstance.

Justification

The preservation of data is extremely important for every farmer seeing as this data will be converted into valuable information used to create reports and heat maps.

Requirements

- To enable offline activity and access of data from the Android app, data synced must be made differentiable with timestamps.
- The system must resolve synchronization conflicts using timestamps.
- Hot swapping of system modules should not affect system service reliability.

3.3.3 Scalability

Description

Scalability is the applications ability to handle above normal workloads. Our client has specified the estimate of number of users for our application, however we will cater for for a larger number not to only be safe, but to also cover the possibility of rapid expansion.

Justification

Our application requires scalability due to the large number of farmers that will be using this application. An estimate of in excess of 900 farmers, each with multiple foreman will make use of it.

Requirements

- The application must be able to cater for more than 900 farmers each with multiple foreman each.
- The application must be able to handle multiple concurrent sessions

3.3.4 Security

Description

Security in terms of software applications is referred to authentication, authorization, data security accounting. Authentication is the ability of our application to identify a user using the application. Authorization is the level of authority that an authenticated user has to execute commands. Data security has to do with our applications data only being accessible through secure channels and accounting refers to the fact that a users actions and effects on the system can be traced and accounted to them.

Justification

Security is one of the most important features in todays age, the Information Age. Information is power. We are concerned about the users data integrity, availability, confidentiality and accountability. Our reason for our worry about data integrity is because the user will be using the data he gathers to determine how to proceed further with his/her business. Real time data analysis and viewing will be a key aspect of our application, so we need to make sure that the integrity of that data is to be trusted. We want to be sure that the user that uses this application does not have access to other users details and statistics and along with this we want accountability for the users who do use this application and bring changes to the database.

Requirements

- User access rights will be checked against the database and depending on the level of clearance, a different user interface will be populated.
- The Android applications user authentication details should not be stored on the Android device itself, but on the server so that the service of the application can be denied if the need arises.
- Passwords and unique salts should be used for each user.
- If a user attempts to access his user account and fails to enter the correct password x amount of times, the account should be locked and require a superuser to unlock it.
- Access to the database should require authentication

3.3.5 Flexibility

Description

Flexibility is the applications ability to be changed dynamically either via a plugin for extension or otherwise.

Justification

Flexibility is necessary for any application to be successful, otherwise it would mean that the application was hardcoded to only work in a certain manner on a certain machine. It is also necessary that when changes need to be made, that as little as the as possible is influenced in the system and needs to be changed. A flexible system is also easier maintain and repair.

Requirements

- The system should allow unregistered users to use existing third party web service login credentials to register or login to the service.
- The system should be decoupled from the database technology it uses and allow the client to select and change the database it uses in future.
- Authentication mechanisms used should be decoupled from the system, allowing them to be interchangeable.
- Modules should be decoupled from one another, allowing the system to be extensible without a break in service which is achieved by integrating new modules and swapping out existing ones.

3.3.6 Maintainability

Description

The system is to be designed in such a way that it is easily updated, modified or extended by the client in the future. In order to achieve these requirements, design patterns and best practices such as

coding style guides are normally used to ensure uniformity and modularity across the system.

Justification

Many systems require regular changes, not because they were poorly designed or implemented, but because of changes in external factors.

Requirements

- Code should be documented in the applicable language documentation framework.
- A coding style should be agreed upon between all the developers so that the same style is used throughout and the same terminology is applied.
- The application should be clearly separated into clear and concise modules to divide concerns and allow for easier maintenance.

3.3.7 Auditability/Monitorability

Description

The system is to be designed to be verbose and transparent in its workings, and to ensure maximum data security, to allow role players to have insights into how the system is used and how it may be improved.
These requirements are achieved by making the maximum amount of relevant data available to authorized users, logging performance critical information, and by enforcing strict constraints on the data that is stored.

Justification

This is an important process in Software Engineering, where all the informations must be correct so requiring all the developers to see who made changes and when so that consistency must be kept in order to keep the database accurate and reliable.

Requirements

- Database data should be consistent and never deleted.
- User creation and edit times should be visible.
- Keep an immutable log of user actions.
- Logging of stack traces and crash analytics should be implemented.

3.3.8 Integrability

Description

The system should allow for future external integration with other platforms such as security authentication providers, other external research metadata databases etc.

Justification

The system necessitates integrability to allow for maximum usability, as the integrability of the system is directly related to how usable it is. To be usable, the system must allow for easy migration, not just from previous systems, but also to future systems and future data storage mediums.

Requirements

- The system should allow technology neutral importing and exporting of data.
- The back-end system should integrate with a desktop web client and Android mobile app clients.
- The system should be able to integrate with different back-end authentication services.

3.3.9 Cost

Description

The cost of the system entails any initial expenses as well as any ongoing expenses which the client may incur at some point. Such expenses arise from software licenses, external computing resources required as well as future maintenance of the system in terms of time.

Justification

The expenses made and software licenses cost must be taken into account in order to set the price of the software.

Requirements

- System should be cheap to operate, maintain and extend. If the quality and maturity of technologies available allow it, technologies used must be freely available/usable.
- As far as possible, open source compatible, mature technologies should be used, to ensure system stability as far as possible.

3.3.10 Usability

Description

Usability refers to ease with which humans, and to a lesser extent, servers, interact with the system in question. Usability can be measured in various ways such as using quantifiable scientific measures or more subjective measures with a key question point being if the API follows conventions and so on.

Justification

It is important that the new system is usable as it is a user-centric system. Ensuring that the system is usable will ensure that users capture accurate and correct information into the system which will for better performance and farmer strategic planning.

Requirements

- Each view in the desktop and mobile clients should be related to a single topic only.
- The web client should render properly and be fully functional in modern web browsers.
- Mobile devices running Android 4.0 and upwards should be fully supported.
- Material design UI guidelines prescribed by Google must be used, to ensure that the clients feel modern and familiar.
- Android app and web client should support the full back-end API specifications.
- The user must be allowed to elect whether they would like their data to be available offline.
- Mobile client users should be able choose between downloading data over Wi-Fi or 3G.

3.4 Integration Requirements

Our web client will be usable on most modern day PCs with support for the big browsers. With the usage of SailsJS we are able to automatically incorporate RESTful APIs with our web client as well as seamless database integration.
Our Android application will focus on acquiring data and sending it once a internet connection is achieved seeing as the foreman wont always have signal out in the orchids. We will persist the data entered locally on the phone only to upload the data later. Due to internet costs we will make sure we keep the data as small as possible and the bandwidth usage low as well. Phonegap and Ionic help

with these regards.

SailsJS provides us with the necessary framework to easily achieve this. It takes care of the REST wrapping automatically and integrates with our database. It uses NodeJS and is built on top of Express, thus we are using fully JavaScript libraries everywhere.

3.5 Architecture Constraints

The only constraints placed on us with regards to the architecture is that the client wants a web client that has a superuser functionality and an Android application. We were given free reign in our judgement on how we wish to accomplish this task. In our technologies section we discuss what we will be using and how it relates to our project.

4. Architectural Tactics or Strategies

4.1 Software Engineering Context : Architectural Tactics

Architectural tactics is concerned with achieving coherence between a design decision and a quality response/promise. Simply put, this refers to how we plan to keep our promise of the level of quality stated previously.

4.2 Performance Tactics

4.2.1 Three Groups of Performance Tactics

- Resource demand tactics
  Simply explained, these are tactics that regulate the allocation of resources
- Conflict resolution tactics
  These are tactics designed specifically to resolve conflicts between modules etc
- Multiple Resource Managements Tactics
  These tactics are resposible for the mangements of concurrent resource accesses and similar tasks

4.3 Reliability Tactics

Reliability of the system refers to amongst other things, the coherence of information stored within the system and presented to users(concurrent). This can be achieved through the use of a duplication scheme which can be sought after using Passive Redundancy, which simply means that the system is designed to store the same information multiple times in multiple independent locations. System reliability also refers to the systems automatic fault detection.

4.3.1 System Context

- Offline data storage
  The user is able to use the Android application even without signal and/or internet connection.

The data entered into the application gets persisted locally and is later then sent to the server once the phone regains connection.

4.4 Scalability Tactics

Scalability refers to the ability to handle increased workload without adding more resources to the system. One way to achieve scalability is to perform a regular SWOT analysis on the system:
Strengths : the kinds of growth the system is designed to handle
Weaknesses : the kinds of growth the system will have trouble handling
Opportunities : possible changes in the system which could be exploited to increase performance
-Threats : possible changes in the workload that the system will not be able to handle

4.5 Security Tactics

Security refers to the protection of information systems from unauthorized disruption that could potentially result in the corruption of information. It is achieved by enforcing access controls on individuals, according to a user hierarchy, to ensure only permitted users have access to the necessary information.

4.5.1 System Context

- Resistance to SQL injections

  SQL injections are dependent on data type awareness in order to be successful. To prevent this, the data being transferred will be encrypted to hide its data type so only the system can understand it. All authentication data will be stored in an encrypted form and never in plain-text.

- Enforce access rights according to user hierarchy

  The different users of the system, such as the farmer and the foremen, will have different access rights accordingly. The farmer will be a super user and will, thus, have access to all functionality that the system offers. The foremen will be general users and have restricted access to administration functionalities. This will simply be enforced by enabling and disabling certain functionalities according to user logins.

- Password encryption

  Upon registration of a user account, the users password will be stored in the database after it has been automatically hashed. When the user logs in, the hashed password will be retrieved and converted to plain-text to perform authentication. No passwords will be stored in the database in plain-text format.

- Enforce database authentication

  Access to database queries will be limited according to the aforementioned user hierarchy.

The farmer will have direct access to view and modify the database whereas the foremen will only be able to make additions to the database regarding worker performance.

4.6 Flexibility Tactics

Flexibility refers to the ability to respond to changes whether internally or externally and how the software reacts to the changes. It also describes how much changes need to be made in order to adapt to the changes.

4.6.1 System Context

- When the mobile app is adapted to run on other platforms the system must not be changed to much and easily plug into the new platform.
- When a different database is chosen the system must not notice that the database is changed.
- When farmers from different industries start to use the service, they only need to update the system to be aware of their specific specifications for yields and other data, with minimal effort.
- Changes to the internal structure of the system should be adapted for with minimal effort.

4.7 Maintainability Tactics

Maintainability refers to modifying the existing system to make improvements or fix bugs without jeopardizing the data integrity and functionality. The need for maintainability is usually based on the following factors: changes in the software environment, new user requirements, bug and error fixes and the requirement of preventative measures for future problems.

4.7.1 System Context

- Enforce code documentation

  JSDoc will be enforced as the documentation framework as its sole purpose is to generate documentation for JavaScript, the language which the whole system will be based on. This will ensure that other developers will always be able to understand what is going on in the code.

- Setup coding standards and conventions

  The CSS3 files should be sorted alphabetically to aid in navigation and understanding for other developers. Other good practices such as naming conventions for variables, functions and classes should be specified to ensure consistency and readability throughout.

- Enforce modularization of system components

  The system should separate concerns by dividing the system components into distinct and independent modules to improve maintainability. This will be enforced with the use of various frameworks such as Node.js, Express,js, Sails,js and Angular.js, which require separation of concerns as they are MVC-oriented.

4.8 Auditability Tactics

Refers to information about the performance of the system , and to provide a log of which user provided which input at which point in time.

4.8.1 System Context

- Any errors thrown by the system or fatal errors made by users should be logged for security and reliability analyses.
- Any attacks aimed at the system should be logged if possible
- Data in the database should be consistent and that will be achieved by constraints put in place and validation of user input.
- Data in the database should never be deleted, if the user deletes data, the data will be moved to another database so that data can be recovered.
- Regular data backups need to be made.

4.9 Integrability and Extensibility Tactics

Integrability refers to testing whether separately developed components work together correctly. Extensibility evaluation is focused on how the addition of new features can take place without losing existing features.

4.9.1 System Context

- JavaScript will be used on the server-side and the client-side of the architecture in order to seamlessly integrate the flow of data.
- The back-end of the system should integrate with a Web client and the mobile applications clients respectively.
  - This will be achieved using Sails.js as this integrates well with Node.js and can, therefore, integrate with AngularJS, which is what both the Web interface and mobile interfaces will be built in.

4.10 Cost Tactics

Costs refer to all the funding required in order for the system to function fully. These costs include development costs, operational costs and maintenance costs.

4.10.1 System Context

- Ensure costs are kept to a minimal

  The chosen technologies are open-source to ensure the costs associated to the operation, maintainability and extension of the system are kept to a minimum. However, if the need arises that the system requires the incorporation of a functionality that open-source software cant offer, then the software will be chosen according to the best value for money. Thus, development costs will be kept to a minimum. The only concerns regarding operational costs are regarding the deployment of the application onto a public exchange interface and the server. The client requires the application to be present in the Google PlayStore and the Apple AppStore, which will infer yearly costs. If the client currently has a functional server that

could handle the workload of the application then the server costs wont be an issue. Otherwise, a server will need to be bought and setup, which will result in various costs. The potential need to invest in an additional server might also occur, but this is dependent on the popularity and scalability of the system. For system development, any free open source IDE or text editor and browser can be utilized by the developers to reduce maintenance costs.

4.10.2 Usability Tactics

Usability refers to the usefulness of the components that have been developed.

4.10.3 System Context

- The web client should be fully functional in popular Web browsers.
  - The Web client must function correctly in all of the popular Web browsers in order to increase the usability, as not everyone uses the same browser.
  - The Web client should function in older versions of the popular browsers as well, as the user may not have the latest updates installed.
- Mobile devices running Android 4.0 and upwards and iOS 7 and upwards should be fully supported.
- The Android app and Web client should support the full back-end API specifications.

5. Reference Architectures and Frameworks

5.1 Backend System

5.1.1 Programming Languages

- Javascript:
  - **Description**
    Javascript is a dynamic scripting language that supports procedural or object orientated programming through use of prototypes. Javascript is the main language on which Node.js packages are written in.
  - **Performance**
    Because of the fact that Javascript is a lightweight scripting language it is fairly fast and mainly depends on the performance of the javascript interpreter or runtime environment.
  - **Flexibility**
    Javascript is a scripting language which means that javascript can run on anything that can interpret the javascript scripts.
  - **Usability**
    Javascript can be used at the client side along with web development or it can be used on node.js to create a web server.

5.1.2 Frameworks

- Node.js
  - **Description**
    Node.js is in fact not a framework but rather a runtime environment for Javascript that enables you to create web servers by only using Javascript. Node.js is built on Googles V8 Javascript Engine which is a very high performance Javascript interpreter built into Google Chrome and written in C++.
  - **Performance**
    Node.js uses an event driven non I/O blocking model that means that no blocking will occur on the server and waiting for real time updates from a lot of connected clients will not tax the server and reduce its performance. This means a lot for the real time nature of our project.
  - **Scalability**
    Because Node.js responds to events instead of waiting on requests like other platforms such as JavaEE typically does the server can easily be scaled to handle a huge amount of client requests. The fact that Node.js runs on an asynchronous model means that many concurrent requests will not result in long waiting times for a response.
  - **Security**
    Node.js has been built with security in mind and is still actively being improved on the security front.
  - **Flexibility**
    Node.js relies on packages to be added, this means that virtually anything can be achieved by including a package from the huge variety of packages available from **Node Package Manager**.
- Express.js
  - **Description**
    Express.js is a minimal web application framework for Node.js. It supports single page, multi page and hybrid page web and mobile applications and is the de facto framework to use for Node.js based web servers.
  - **Performance**
    Express.js is very minimal and therefore does not affect the performance of Node.js much.
- Sails.js
  - **Description**
    Sails.js is a MVC style framework that runs on top of Express.js. It is built to enable rapid development of server side web applications and works very well for applications that needs real time functionality.
  - **Flexibility**
    Sails.js does not depend on any type of front end. It also supports any database and can support many different databases in the same project. Any database that is not supported can still be used through adapters.
  - **Maintainability**
    Sails.js is designed to be coded once and just work. Therefore it is coded once and forgotten very little maintenance is needed.
  - **Usability**

Sails.js is built from the start to be very easy to use and set up. It can also create RESTful apis for you project

5.1.3 Database System

- Neo4j
  - **Description**
    Neo4j is a popular graph database that is an ACID compliant transactional database. The database management system uses Cypher Query Language or in short CQL
  - **Performance**
    Because of the graph nature of Neo4j there is no need to iteratively traverse a linear list of indexes to find an entry, this means read and write performance of Neo4j is very fast.
  - **Scalability**
    The so called index-free adjacency property of a graph means that the time it takes to get the relationships between entries stays constant as the database size increases as opposed to relational databases where the time takes longer as the database enlarges.
  - **Usability**
    According to the developers of Neo4j it is very easy to learn and use.

5.1.4 Operating System

- Linux

5.1.5 Dependency Management and Build Tools

- NPM
  - **Description**
    Node package manager is not only used to install new packages into the local Node repository, but it also has the added feature of managing dependencies on packages within your project.

5.2 Web Interface

5.2.1 Programming Languages

- Javascript
- HTML5
  - **Description**
    HTML5 is the 5th iteration of Hyper Text Transfer Protocol. HTML is a markup language used mainly in websites to markup text into something more interesting that plain text.
  - **Usability**
    HTML5 is mainly used for web pages but can also be used to design an interface in hybrid mobile apps.
- CSS3
  - **Description**
    CSS3 is the third iteration of Cascading Style Sheet. CSS itself is also a markup language. CSS is used in conjunction with HTML to mark up web pages.
  - **Usability**
    CSS is used to stylise HTML text to make it more visually appealing. It can be used to

color text, define spacing define image sizing and much more.

5.2.2 Frameworks

- Sails.js
- AngularJs
  - **Description**
    AngularJS is a Javascript framework that extends HTML5. It extends HTML DOM and makes it more responsive.
  - **Flexibility**
    AngularJS most suited for application development where single page applications are mostly used. But AngularJS also works well to make normal web pages responsive. Angular is usability test ready and is linked together by Dependency Injection.
  - **Usability**
    AngularJS makes it very easy to implement MVC by asking the programmer to split the project into the components and then Angular will handle the rest.
- Bootstrap
  - **Description**
    Bootstrap is a mobile-first front-end framework for websites and webapps that mainly aids in layout design of web projects.
  - **Flexibility**
    Bootstrap makes websites responsive to screen sizes and automatically switches to a more mobile friendly layout when screen sizes reach a certain size. This makes it possible to code the website once for different screen sizes.
  - **Usability**
    Bootstrap can be used for normal full sized websites or for sites intended for mobile devices or even for websites meant to be viewed on both screens. Bootstrap also works brilliantly for managing the layout in hybrid apps.

5.2.3 Libraries

- Google Maps API
  - **Description**
    There was no other place to put this but under frameworks, even though it is not a framework but rather an Application Programming Interface. Google Maps API is a service from Google that enables developers to embed google maps into their websites and perform operations on the embedded map.

5.2.4 Database System

- Neo4j
- Cookies
  - **Description**
    Cookies are a temporary method of local storage mainly used to remember user logins or user preferences on websites
- HTML5 Local Storage
  - **Description**
    HTML5 Local Storage is a way for websites to store data on the client in a sandbox so

that the stored data is only accessible by the browser that controls the sandbox.
  - **Security**
    The sandbox makes sure no malicious data will make it through to any important places on the computer
  - **Usability**
    There exists some local databases that take advantage of HTML5 Local Storage to make a type of offline database that can be used to store data that needs to be synced with a web database at a later stage.

5.2.5 Operating System

- Device Operating System Independent
- Web Browser
  - Google Chrome
  - Mozilla Firefox
  - Microsoft Internet Explorer 8,9 & 10
  - Microsoft Edge
  - Apple Safari

5.2.6 Dependency Management and Build Tools

- Unit.js
- Dagger

5.3 Android Client

5.3.1 Programming Languages

- Javascript
- HTML5
- CSS

5.3.2 Frameworks

- Sails.js
- AngularJS
- Bootstrap
- Phonegap
  - **Description**
    Phonegap is a framework that lets developers take any ordinary web development skills and make mobile apps with that knowledge.
  - **Usability**
    Developers code the app in any website and the framework will convert the website into an app, it does this by running the website in some sort of implicit web browser.
  - **Flexibility**
    The framework has support for plugins that makes more native operations possible for the hybrid app, such as being able to use the camera to take photos or use that accelerometer to take certain measurements.
- Ionic

- **Description**
  Ionic is a hybrid mobile app development framework that includes HTML, CSS and Javascript to create native looking hybrid mobile apps.
- **Flexibility**
  Ionic is built on top of AngularJS and Phonegap to create powerful yet native hybrid apps. It is native focused which means that Ionic follows the native platform designs to build apps that look like its been natively developed for the platform on which the hybrid app runs.
- **Performance**
  Ionic is optimized to be fast so that your hybrid app is fast and not slow as a website.
- **Usability**
  According to the Ionic website it is easy to learn if you already know web development. On top of that you also learn AngularJS as you learn Ionic.

5.3.3 Libraries

- Google Maps API

5.3.4 Database System

- HTML5 Local Storage
- Neo4j

5.3.5 Operating System

- Android 4.0 (Ice cream sandwich) and up.
- iOS 7 and up

5.3.6 Dependency Management and Build Tools

- Unit.js
- Dagger

6. Access and Integration Channels

6.1 Access Channels

The farmers will be able to access the system through both a Web interface, specifically designed for the farmer, and a Mobile interface with underlying web-services. The foremen will only have access to the Mobile interface. The Web interface will be accessible by navigating to the URL of the web page within the preferred web browser of the farmer. The Mobile interface will be accessible by downloading the application from the Google Play Store or the Apple AppStore and installing it on their device. Both interfaces will make use of cookies and session IDs in order to keep the user logged in to the system to facilitate ease of use.

6.2 Integration Channels

The entire system will be implemented using a client-server architecture due to the dynamic message-passing nature of the application. However, the frameworks that we will be using to implement the required functionality are based on the Model-View-Controller architecture. Thus, it will be the effective integration of these frameworks MVC-based functionality working collaboratively as a client-server architecture. The entire system will be coded using JavaScript as it is a dynamic, versatile language that has a lot of support. This also ensures that integration between the back-end and front-end system wont cause any serious complications. Libraries and frameworks will also be used to adapt JavaScript to be able to run on a mobile environment.

Node.js will handle the server-side communication for the system as it handles real-time data transfer quickly and efficiently regardless of scalability. Express.js will be the primary web server framework to integrate with Node.js in order to handle the message-passing. Sails.js will be a fundamental framework implemented on both the front-end and back-end of the system as it offers a multitude of functionalities and primarily integrates with Node.js. The Angular.js framework will be used collaboratively with HTML5 and JavaScript to provide the required functionality. CSS3 and the Bootstrap framework will be used to handle the styling of the front-end. AJAX will potentially be used to handle the passing of any data that cannot be handled by the other abovementioned frameworks. The Google Maps JavaScript API Heatmap Layer will be incorporated into the system in order to generate the heatmap functionality that the system requires. PhoneGap and Ionic will be used together to adapt the Web interface code to run on an Android and iOS platform.

Neo4j is the chosen database for the system as it offers the advantages of both a NoSQL database and a SQL database. It effectively handles scalability without sacrificing performance, whilst also having the ability to store relationships between data entries. Sails.js also provides a plugin that adapts our implementation to effectively integrate with Neo4j.

7. Technologies

7.1 Backend System

7.1.1 Programming Languages

- JavaScript

7.1.2 Runtime Environment

- Node.js

7.1.3 Frameworks

- Express.js
- Sails.js

7.1.4 Database System

- Neo4j

7.1.5 Server

- Cloud-based

7.1.6 Dependency Management and Test Tools

- Unit.js
- Dagger

7.2 Web Interface

7.2.1 Programming Languages

- HTML5
- CSS3
- JavaScript

7.2.2 Frameworks

- Sails.js
- Angular.js
- Google Maps JavaScript API Heatmap Layer
- Bootstrap
- AJAX

7.2.3 Database System

- Neo4j
- HTML5 Local Storage
- Cookies

7.2.4 Browser Compatibility

- Google Chrome
- Mozilla Firefox
- Microsoft Internet Explorer 8, 9 and 10
- Microsoft Edge
- Apple Safari

7.2.5 Dependency Management and Test Tools

- Unit.js
- Dagger

7.3 Android and iOS Client

7.3.1 Programming Languages

- JavaScript

7.3.2 Frameworks

- PhoneGap
- Ionic
- Sails.js
- Angular.js
- Google Maps JavaScript API Heatmap Layer
- Bootstrap
- AJAX

7.3.3 Database System

- Neo4j

7.3.4 Operating System

- Android Version 4 (Ice Cream Sandwich) to 6 (Marshmallow)
- iOS 7 to 9

7.3.5 Dependency Management and Test Tools

- Unit.js
- Dagger

8. Functional Requirements

8.1 Use Case Prioritization

8.1.1 Critical

- Login/Logout user
- Change Password
- Recover Password
- View/Edit/Create Farmer Web interface
- View/Edit/Create Farm Web interface
- View/Edit/Create Foreman (by farmer) Web interface
- View/Edit/Create Worker (by farmer) Web interface
- View/Create/Edit Orchard Details (crop dimensions, crop type, irrigation type, date planted, yields per hectare) Web interface
- View/Create/Edit Irrigation Type (by farmer) Web interface
- View/Create/Edit Crop Type Web interface
- View/Update Worker Performance (yields collected per worker)
- View/Create/Edit Yield Measurement Type (by farmer, eg. kg, bag, g, etc.) Web interface
- View/Create/Edit Cultivation Method Web interface
- Allocate/Deallocate Foreman to Orchard Blocks Web interface
- Assign/Reassign Cultivation Method to Specific Orchard Blocks Web interface
- Assign/Reassign Workers to Foreman (by farmer) Web interface

8.1.2 Important

- Import Census Data Web interface
- Generate Statistical Report of Worker Performance (time intervals) Web interface
- Generate Statistical Report Crop Yield per Orchard (potentially linked to heatmap generation) Web interface

8.1.3 Nice to have

- View Heat Map Web interface
- View/Create/Edit Foremans Shift (potentially linked to location tracking) Web interface
- Notify Farmer Regarding Foremans Locations (according to time intervals)
- Notify Farmer of Foremans Activity History Every Half an Hour
- Generate Revenue Report Regarding Seasonal Yields (to plan paying workers, operational costs, etc.) Web interface
- Generate Statistical Report Regarding Time Taken to Yield Specific Crops Web interface

8.2 Use Cases

8.2.1 Login User

- Description
  This use case will be used by the users of the Web interface, Android interface and the iOS interface to initiate login via the back-end service.
- Pre-Conditions
  1. The user has a registered account within the database.
  2. The users account is not locked.
- Post-Conditions
  1. The user will be logged in and have access to the necessary functionality.

8.2.2 Logout User

- Description
  This use case will be used by the users of the Web interface, Android interface and the iOS interface to log a user out of the system.
- Pre-Conditions
  1. The user is currently logged into the system.
- Post-Conditions
  1. The user will be logged out of the system and have no access to any functionality.

8.2.3 Change Password

- Description
  This use case will be used by the users of the Web interface, Android interface and the iOS interface to change their password.
- Pre-Conditions
  1. The user has a registered account within the database.
  2. The users account is not locked.
- Post-Conditions
  1. The users password is updated in the database.

8.2.4 Recover Password

- Description
  This use case will be used by the users of the Web interface, Android interface and the iOS interface to recover their forgotten password.
- Pre-Conditions

1. The user has a registered account within the database.
2. The users account is not locked.

- Post-Conditions
  1. The user will receive an email containing their password.

8.2.5 Create Farmer

- Description
  This use case will be initiated by the farmer to create his super user account for the system via the Web interface.
- Pre-Conditions
  1. The farmer doesnt already exist in the database.
- Post-Conditions
  1. The farmer will have a super user account registered in the database.

8.2.6 View Farmer

- Description
  This use case will be initiated by the farmer to view the current state of his super user account for the system via the Web interface.
- Pre-Conditions
  1. The farmer is currently logged into the system. (i.e. Super user logged in)
  2. The farmer already exists in the database.
- Post-Conditions
  1. The farmers account details will be displayed.

8.2.7 Edit Farmer

- Description
  This use case will be initiated by the farmer to edit his super user account for the system via the Web interface.
- Pre-Conditions
  1. The farmer is currently logged into the system. (i.e. Super user logged in)
  2. The farmer already exists in the database.
- Post-Conditions
  1. The farmers details are updated in the database.

8.2.8 Create Farm

- Description
  This use case will be initiated by the farmer to register his farm on the system via the Web interface.
- Pre-Conditions
  1. The farmer is currently logged into the system. (i.e. Super user logged in)
  2. The farm doesnt already exist in the database.
- Post-Conditions
  1. The farm is registered in the database.

8.2.9 View Farm

- Description
  This use case will be initiated by the farmer to view the current state of his farms details on the system via the Web interface.
- Pre-Conditions
  1. The farmer is currently logged into the system. (i.e. Super user logged in)
  2. The farm already exists in the database.
- Post-Conditions
  1. The farms account details will be displayed.

8.2.10 Edit Farm

- Description
  This use case will be initiated by the farmer to edit his farms details on the system via the Web interface.
- Pre-Conditions
  1. The farmer is currently logged into the system. (i.e. Super user logged in)
  2. The farm already exists in the database.
- Post-Conditions
  1. The farms details are updated in the database.

8.2.11 Create Foreman

- Description
  This use case will be initiated by the farmer to register his foremen individually on the system as general users via the Web interface.
- Pre-Conditions
  1. The farmer is currently logged into the system. (i.e. Super user logged in)
  2. The foreman doesnt already exist in the database.
- Post-Conditions
  1. The foreman will have a general user account registered in the database.
  2. Login details are generated for the foreman.

8.2.12 View Foreman

- Description
  This use case will be initiated by the farmer to view the current state of his foremans general user account for the system via the Web interface.
- Pre-Conditions
  1. The farmer is currently logged into the system. (i.e. Super user logged in)
  2. The foreman already exists in the database.
- Post-Conditions
  1. The foremans account details will be displayed.

8.2.13 Edit Foreman

- Description
  This use case will be initiated by the farmer to edit his foremans general user account for the system via the Web interface.

- Pre-Conditions
  1. The farmer is currently logged into the system. (i.e. Super user logged in)
  2. The foreman already exists in the database.
- Post-Conditions
  1. The foremans details are updated in the database.

8.2.14 Create Worker

- Description
  This use case will be initiated by the farmer to add his workers individually onto the system via the Web interface.
- Pre-Conditions
  1. The farmer is currently logged into the system. (i.e. Super user logged in)
  2. The worker doesnt already exist in the database.
- Post-Conditions
  1. The workers details are in the database.

8.2.15 View Worker

- Description
  This use case will be initiated by the farmer to view the current state of his workers details on the system via the Web interface.
- Pre-Conditions
  1. The farmer is currently logged into the system. (i.e. Super user logged in)
  2. The worker already exists in the database.
- Post-Conditions
  1. The workers account details will be displayed.

8.2.16 Edit Worker

- Description
  This use case will be initiated by the farmer to edit his workers details on the system via the Web interface.
- Pre-Conditions
  1. The farmer is currently logged into the system. (i.e. Super user logged in)
  2. The worker already exists in the database.
- Post-Conditions
  1. The workers details are updated in the database.

8.2.17 Demarcate Orchard Blocks using Coordinates

- Description
  This use case will be initiated by the farmer to demarcate the orchard blocks on his farm according to map coordinates via the Web interface.
- Pre-Conditions
  1. The farmer is currently logged into the system. (i.e. Super user logged in)
  2. The orchard block hasnt already been demarcated.
- Post-Conditions
  1. The orchard blocks demarcated area is stored in the database.

8.2.18 View Demarcated Orchard Blocks

- Description
  This use case will be initiated by the farmer to view the demarcated orchard blocks on his farm via the Web interface.
- Pre-Conditions
  1. The farmer is currently logged into the system. (i.e. Super user logged in)
  2. The orchard block has already been demarcated.
- Post-Conditions
  1. The demarcated orchard blocks details are displayed.

8.2.19 Edit Demarcated Orchard Blocks

- Description
  This use case will be initiated by the farmer to edit the demarcated orchard blocks on his farm via the Web interface.
- Pre-Conditions
  1. The farmer is currently logged into the system. (i.e. Super user logged in)
  2. The orchard block has already been demarcated.
- Post-Conditions
  1. The demarcated orchard blocks details are updated in the database.

8.2.20 Create Irrigation Type

- Description
  This use case will be initiated by the farmer to create an irrigation type used on his farm on the system via the Web interface.
- Pre-Conditions
  1. The farmer is currently logged into the system. (i.e. Super user logged in)
  2. The irrigation type doesnt already exist in the database.
- Post-Conditions
  1. The irrigation type is added to the database.

8.2.21 View Irrigation Type

- Description
  This use case will be initiated by the farmer to view the current state of an irrigation types details on the system via the Web interface.
- Pre-Conditions
  1. The farmer is currently logged into the system. (i.e. Super user logged in)
  2. The irrigation type already exists in the database.
- Post-Conditions
  1. The irrigation types details will be displayed.

8.2.22 Edit Irrigation Type

- Description
  This use case will be initiated by the farmer to edit an irrigation types details on the system via the Web interface.
- Pre-Conditions

1. The farmer is currently logged into the system. (i.e. Super user logged in)
2. The irrigation type already exists in the database.

- Post-Conditions
  1. The irrigation types details are updated in the database.

8.2.23 Create Crop Type

- Description
  This use case will be initiated by the farmer to create a crop type planted on his farm on the system via the Web interface.
- Pre-Conditions
  1. The farmer is currently logged into the system. (i.e. Super user logged in)
  2. The crop type doesnt already exist in the database.
- Post-Conditions
  1. The crop type is added to the database.

8.2.24 View Crop Type

- Description
  This use case will be initiated by the farmer to view the current state of a crop types details on the system via the Web interface.
- Pre-Conditions
  1. The farmer is currently logged into the system. (i.e. Super user logged in)
  2. The crop type already exists in the database.
- Post-Conditions
  1. The crop types details will be displayed.

8.2.25 Edit Crop Type

- Description
  This use case will be initiated by the farmer to edit a crop types details on the system via the Web interface.
- Pre-Conditions
  1. The farmer is currently logged into the system. (i.e. Super user logged in)
  2. The crop type already exists in the database.
- Post-Conditions
  1. The crop types details are updated in the database

8.2.26 View Worker Performance

- Description
  This use case will be initiated by the foreman to view the current state of a specific workers performance details on the system via the Android or iOS interface.
- Pre-Conditions
  1. The foreman is currently logged into the system.
  2. The worker already exists in the database.
- Post-Conditions
  1. The workers performance details will be displayed.

8.2.27 Update Worker Performance

- Description
  This use case will be initiated by the foreman to update a specific workers performance by adjusting the yields collected by the worker on the system via the Android or iOS interface.
- Pre-Conditions
  1. The foreman is currently logged into the system.
  2. The worker already exists in the database.
- Post-Conditions
  1. The workers performance details are updated in the database.

8.2.28 Create Yield Measurement Type

- Description
  This use case will be initiated by the farmer to create a yield measurement type used on his farm on the system via the Web interface.
- Pre-Conditions
  1. The farmer is currently logged into the system. (i.e. Super user logged in)
  2. The yield measurement type doesnt already exist in the database.
- Post-Conditions
  1. The yield measurement type is added to the database.

8.2.29 View Yield Measurement Type

- Description
  This use case will be initiated by the farmer to view the current state of a yield measurement types details on the system via the Web interface.
- Pre-Conditions
  1. The farmer is currently logged into the system. (i.e. Super user logged in)
  2. The yield measurement type already exists in the database.
- Post-Conditions
  1. The yield measurement types details will be displayed.

8.2.30 Edit Yield Measurement Type

- Description
  This use case will be initiated by the farmer to edit a yield measurement types details on the system via the Web interface.
- Pre-Conditions
  1. The farmer is currently logged into the system. (i.e. Super user logged in)
  2. The yield measurement type already exists in the database.
- Post-Conditions
  1. The yield measurement types details are updated in the database.

8.2.31 Create Cultivation Method

- Description
  This use case will be initiated by the farmer to create a cultivation method used on his farm on the system via the Web interface.
- Pre-Conditions

1. The farmer is currently logged into the system. (i.e. Super user logged in)
2. The cultivation method doesnt already exist in the database.

- Post-Conditions
  1. The cultivation method is added to the database.

8.2.32 View Cultivation Method

- Description
  This use case will be initiated by the farmer to view the current state of a cultivation methods details on the system via the Web interface.
- Pre-Conditions
  1. The farmer is currently logged into the system. (i.e. Super user logged in)
  2. The cultivation method already exists in the database.
- Post-Conditions
  1. The cultivation methods details will be displayed.

8.2.33 Edit Cultivation Method

- Description
  This use case will be initiated by the farmer to edit a cultivation methods details on the system via the Web interface.
- Pre-Conditions
  1. The farmer is currently logged into the system. (i.e. Super user logged in)
  2. The cultivation method already exists in the database.
- Post-Conditions
  1. The cultivation methods details are updated in the database.

8.2.34 Allocate Foreman to Orchard Block

- Description
  This use case will be initiated by the farmer to allocate a specific foreman to a specific orchard block on his farm on the system via the Web interface.
- Pre-Conditions
  1. The farmer is currently logged into the system. (i.e. Super user logged in)
  2. The foreman already exists in the database.
  3. The orchard block already exists in the database.
- Post-Conditions
  1. The foreman-orchard assignment is added to the database.

8.2.35 Deallocate Foreman from Orchard Block

- Description
  This use case will be initiated by the farmer to deallocate a specific foreman from a specific orchard block on his farm on the system via the Web interface.
- Pre-Conditions
  1. The farmer is currently logged into the system. (i.e. Super user logged in)
  2. The foreman-orchard assignment already exists in the database.
- Post-Conditions
  1. The foreman-orchard assignment is removed from the database.

8.2.36 Assign Cultivation Method to Orchard Block

- Description
  This use case will be initiated by the farmer to assign a specific cultivation method to a specific orchard block on his farm on the system via the Web interface.
- Pre-Conditions
  1. The farmer is currently logged into the system. (i.e. Super user logged in)
  2. The cultivation method already exists in the database.
  3. The orchard block already exists in the database.
- Post-Conditions
  1. The cultivation-orchard assignment is added to the database.

8.2.37 Reassign Cultivation Method to Orchard Block

- Description
  This use case will be initiated by the farmer to reassign a different cultivation method to a specific orchard block on his farm on the system via the Web interface.
- Pre-Conditions
  1. The farmer is currently logged into the system. (i.e. Super user logged in)
  2. The cultivation-orchard assignment already exists in the database.
- Post-Conditions
  1. The cultivation-orchard assignment is updated in the database.

8.2.38 Assign Workers to Foreman

- Description
  This use case will be initiated by the farmer to assign a specific worker to a specific foreman on the system via the Web interface.
- Pre-Conditions
  1. The farmer is currently logged into the system. (i.e. Super user logged in)
  2. The worker already exists in the database.
  3. The foreman already exists in the database.
- Post-Conditions
  1. The worker-foreman assignment is added to the database.

8.2.39 Reassign Workers to Foreman

- Description
  This use case will be initiated by the farmer to reassign a specific worker to a different foreman on the system via the Web interface.
- Pre-Conditions
  1. The farmer is currently logged into the system. (i.e. Super user logged in)
  2. The worker-foreman assignment already exists in the database.
- Post-Conditions
  1. The worker-foreman assignment is updated in the database.

8.2.40 Import Census Data

- Description
  This use case will be initiated by the farmer to import current census data onto the system via

the Web interface to reduce the amount of data needed to be input manually.

- Pre-Conditions
  1. The farmer is currently logged into the system. (i.e. Super user logged in)
  2. The census data is valid and in the correct format.
- Post-Conditions
  1. The census data is added and successfully integrated into the system.

8.2.41 Generate Statistical Report of Worker Performance (according to time intervals)

- Description
  This use case will be initiated by the farmer to generate a report showing the performance of his workers during certain time intervals for statistical purposes via the Web interface.
- Pre-Conditions
  1. The farmer is currently logged into the system. (i.e. Super user logged in)
  2. The data on the workers performance, required for the report, is present in the database.
- Post-Conditions
  1. The workers performance report has been generated in a usable format.

8.2.42 Generate Statistical Report of Crop Yield per Orchard

- Description
  This use case will be initiated by the farmer to generate a report showing the crop yield per orchard for statistical purposes via the Web interface.
- Pre-Conditions
  1. The farmer is currently logged into the system. (i.e. Super user logged in)
  2. The data on the crop yields for each orchard, required for the report, is present in the database.
- Post-Conditions
  1. The crop yield per orchard report has been generated in a usable format.

8.2.43 View Heat Map

- Description
  This use case will be initiated by the farmer to view the crop yields per orchard blocks according to a heat map via the Web interface.
- Pre-Conditions
  1. The farmer is currently logged into the system. (i.e. Super user logged in)
  2. The data on the crop yields for each orchard, required to generate the heat map, is present in the database.
- Post-Conditions
  1. The crop yield per orchard heat map is generated and displayed.

8.2.44 Create Foremans Shift

- Description
  This use case will be initiated by the farmer to allocate a foreman to a specific shift on the system via the Web interface.
- Pre-Conditions
  1. The farmer is currently logged into the system. (i.e. Super user logged in)

2. The foreman already exists in the database.
3. The foreman-shift assignment doesnt exist in the database.

- Post-Conditions
  1. The foreman-shift assignment is added to the database.

8.2.45 View Foremans Shift

- Description
  This use case will be initiated by the farmer to view the current state of a foremans shift details on the system via the Web interface.
- Pre-Conditions
  1. The farmer is currently logged into the system. (i.e. Super user logged in)
  2. The foreman-shift assignment already exists in the database.
- Post-Conditions
  1. The foreman-shift assignment details will be displayed.

8.2.46 Edit Foremans Shift

- Description
  This use case will be initiated by the farmer to edit a foremans shift details on the system via the Web interface.
- Pre-Conditions
  1. The farmer is currently logged into the system. (i.e. Super user logged in)
  2. The foreman-shift assignment already exists in the database.
- Post-Conditions
  1. The foreman-shift assignment details are updated in the database.

8.2.47 Notify Farmer Regarding Foremans Locations (according to time intervals)

- Description
  This use case will be initiated when a foreman leaves the demarcated area he has been allocated during his shift hours. When this occurs, a SMS or in-app notification will alert the farmer regarding this unusual occurrence via the Android or iOS interface.
- Pre-Conditions
  1. The farmer is currently logged into the system on his mobile device. (i.e. Super user logged in)
  2. The foreman is logged into the system on his mobile device.
  3. The data regarding the foremans shift, allocated orchard block and his current GPS location are available to initiate the notification.
- Post-Conditions
  1. The farmer receives an SMS or an in-app notification regarding the foremans current location.

8.2.48 Notify Farmer of Foremans Activity History Every Half an Hour

- Description
  This use case will be initiated every half an hour to notify the farmer on his moble device regarding the foremans activity history to prevent theft.
- Pre-Conditions

1. The farmer is currently logged into the system on his mobile device. (i.e. Super user logged in)
2. The foreman is logged into the system on his mobile device.
3. The foremans activity history is present in the database.

- Post-Conditions
  1. The farmer receives an SMS or an in-app notification every half an hour regarding the foremans activity history.

8.2.49 Generate Revenue Report Regarding Seasonal Yields

- Description
  This use case will be initiated by the farmer to generate a report showing the revenue generated according to seasonal yields per orchard block for statistical purposes via the Web interface.
- Pre-Conditions
  1. The farmer is currently logged into the system. (i.e. Super user logged in)
  2. The data on the crop yields for each orchard and the related revenue, required for the report, is present in the database.
- Post-Conditions
  1. The revenue per orchard report has been generated in a usable format.

8.2.50 Generate Statistical Report Regarding Time Taken to Yield Specific Crops

- Description
  This use case will be initiated by the farmer to generate a report showing the amount of time it takes to yield certain crops for statistical purposes via the Web interface.
- Pre-Conditions
  1. The farmer is currently logged into the system. (i.e. Super user logged in)
  2. The data regarding the time it takes to yield specific crops is present in the database.
- Post-Conditions
  1. The time taken to yield specific crops report has been generated in a usable format.

9. Open Issues

9.1 Database Issues

- We are not exactly sure how we are going to design our database and store data. We do not yet know if we need relationships or not
- We have not yet decided on a proper local database that takes advantage of HTML5 Local Storage.