SAMBUG

Architectural Requirements

VecFor
Software Developers

October, 2015

Abrie van Aardt	13178840
Werner Mostert	13019695
Kale-ab Tessera	13048423
Keagan Thompson	13023782
Michelle Swanepoel	13066294

Contents

1 Architectural Requirements

1.1 Quality Requirements

1.1.1 Maintainability

As a standard requirement for all projects, SAMBUG needs to be maintainable. This is especially due to the fact that the project has very high expansion potential. SUBTROP has shown interest in further maintaining the project after completion. Since development on the project is split among 5 members, we have to ensure that the code is as readable as possible and this includes using good documentation of code in the form of comments and Javadoc descriptions.

1.1.2 Scalability

Since the user base for the system will be comparatively small, scalability is not a great issue. The load on the system will not under any foreseeable circumstances be extremely high.

1.1.3 Performance Requirements

Performance is a very important requirement for the system. The goal of the system is to provide an efficient service rendered. Users of the system will be subject to high pressure environments where time is an issue. Using phone memory effectively is also a must as the phones used by the users might not be latest models, and therefore may struggle with some of the more graphical parts of the app.

1.1.4 Reliability and Availability

Since users will be relying heavily on this system to make important business decisions the system must be available to be used at all times. As such, a reliable system is of paramount importance. The storing of data and persistence of said data to the server must be completely reliable since this is the essential part of the project goal.

1.1.5 Security

As specified by the client, for the sake of reducing usage complexity of the system, the security may be relaxed in this case due to the fact that the users are generally not comfortable with using technological products. However, security measures should be put in place to help the user navigate through the app in order to complete the data entry process.

1.1.6 Auditability

There is no specific auditability requirement for the system since there will be no business value in tracking user activity, but for some traceability the user details of someone who edits the database is stored.

1.1.7 Testability

Every service offered by the system should be testable via:
1. Unit tests
2. Integration tests

1.1.8 Usability

Usability is the main quality requirement for the system. In order for the system to have any actual business value it should be easy to use and effective. As far as possible, the user should be able to use the app without needing textual instructions.

1.1.9 Deployability

The system must be deployable to any cloud hosted, Windows based, virtual machine or locally hosted server.

1.2 Architecture responsibilities

The architectural responsibilities of the system include providing infrastructure for the following:

- A web access channel
- A mobile access channel
- Hosting for business logic and services offered by the system
- Relational database persistence
- Sending emails for recovering of accounts

2 Architectural Design

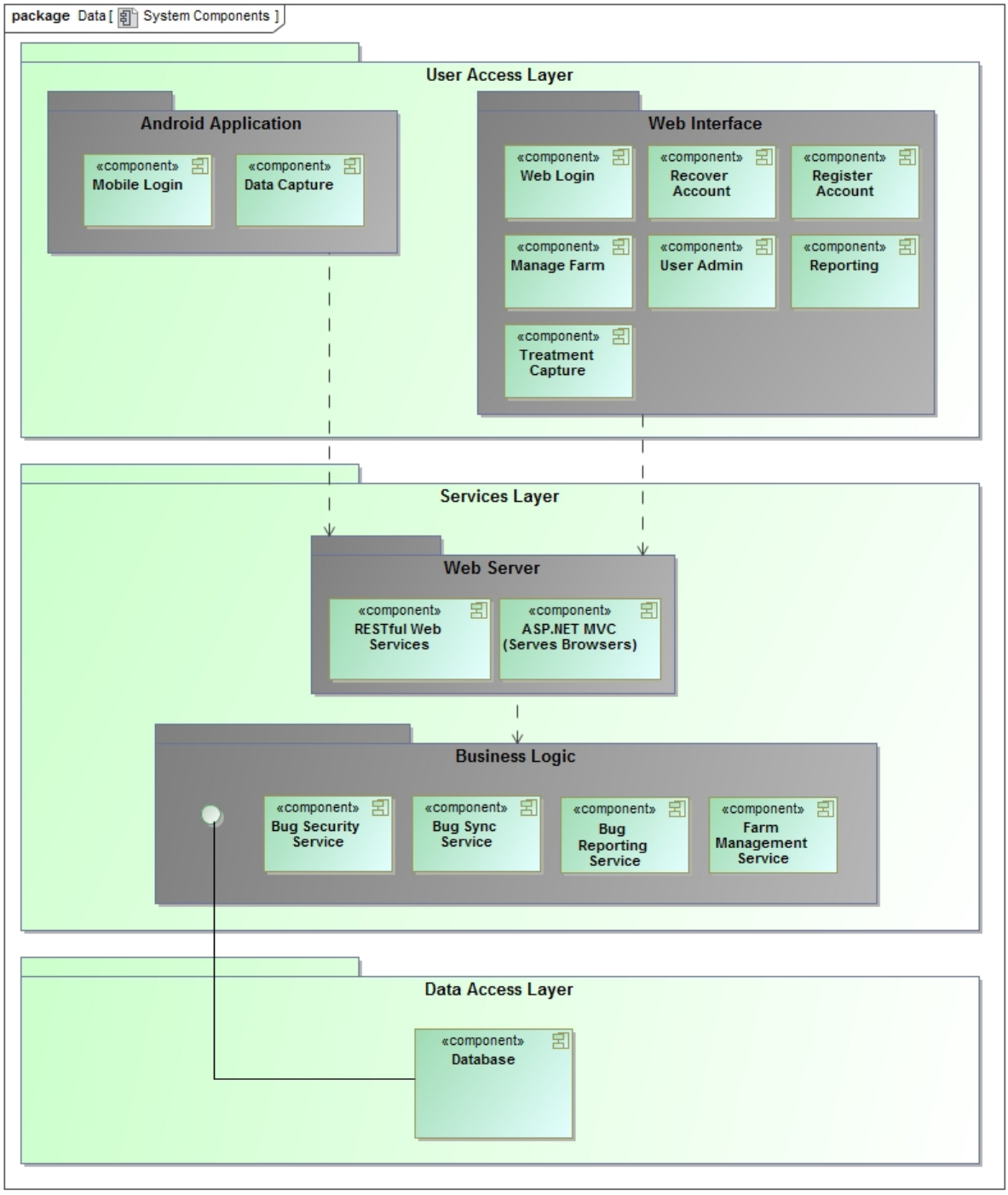
2.1 Overview

A layered architecture will be used to provide for better decoupling of the various access channels and architectural responsibilities. Within these layers the Model-View-Controller pattern will be applied to further separate concerns among what can be viewed as three distinct parts of a Web application. The following provides an overview of the architecture:

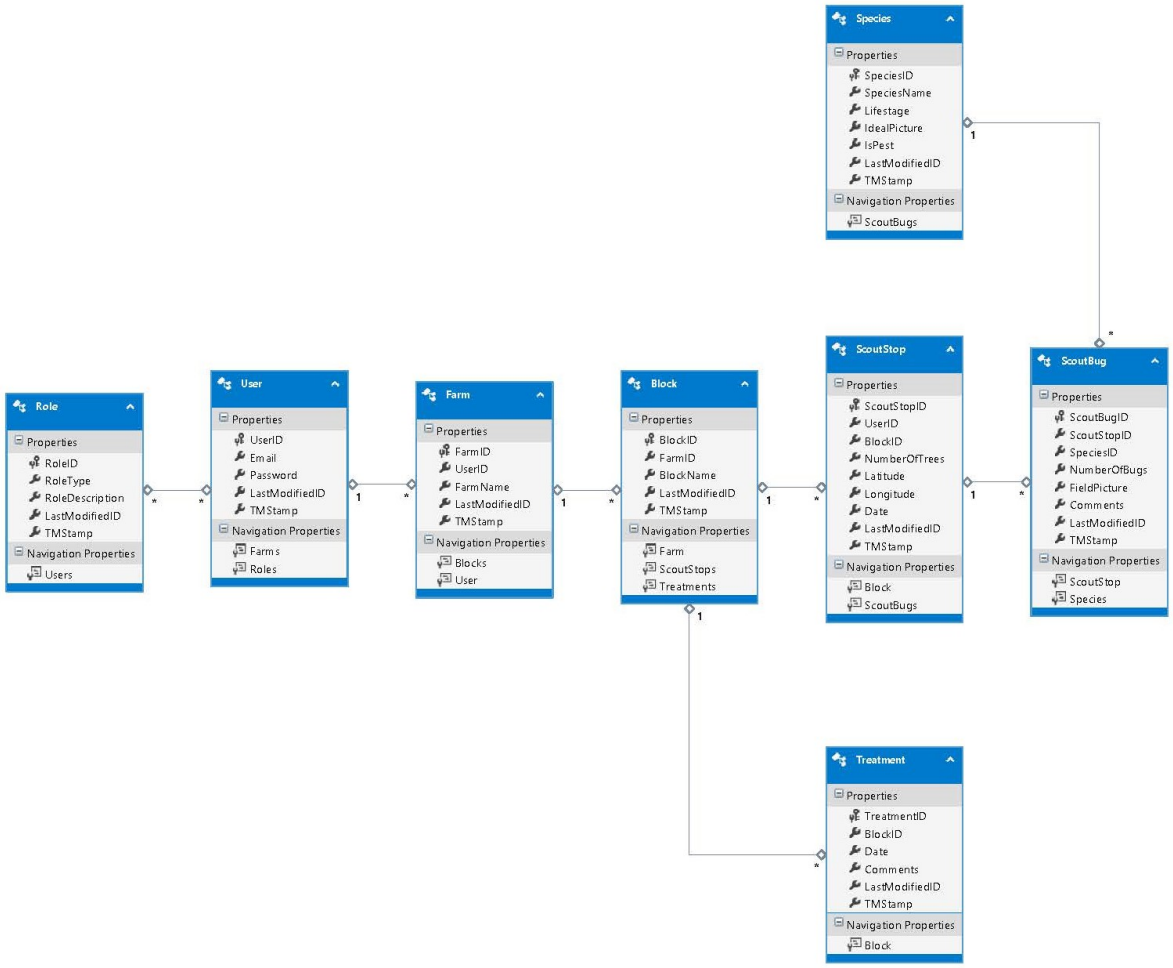
- Access Layer
  - View
    In many ways the Android application serves as an access channel to the Web server. However, it is not categorised as a "View" in the MVC sense. Our MVC pattern is only used across the Access and Services layers. Therefore, this only encompasses HTML views that are rendered using the Razor rendering engine.
  - View Model
    Model definitions are used both in the View and in the Controller, since these define the primary format of data transfer between those two components. It is important to remember that since MVC is used solely for the frontend, these models are only view models, and not data or domain models.
- Services Layer
  - View Model
  - Web and RESTful Controllers
    The MVC Web Controllers communicate with Browser-based clients whilst the RESTful controllers use JSON to communicate with other clients.
  - Business Logic
    Business Logic is extracted from both sets of controllers to allow for common services and for thinner controllers.

- Data Access Layer
  - Domain Model
  - Data Abstraction (Repository)
    This interface defines a contract between the application and the database, allowing us to swap our current database implementation for any other implementation that realises the contract. Query methods map entities to domain models which are returned to the Services layer for further processing. This further decouples our domain from our data.
  - MSSQL Implementation
    This is our current Database implementation. Entity Framework is used as an ORM. Entities are defined here and are used to query the database with LINQ.

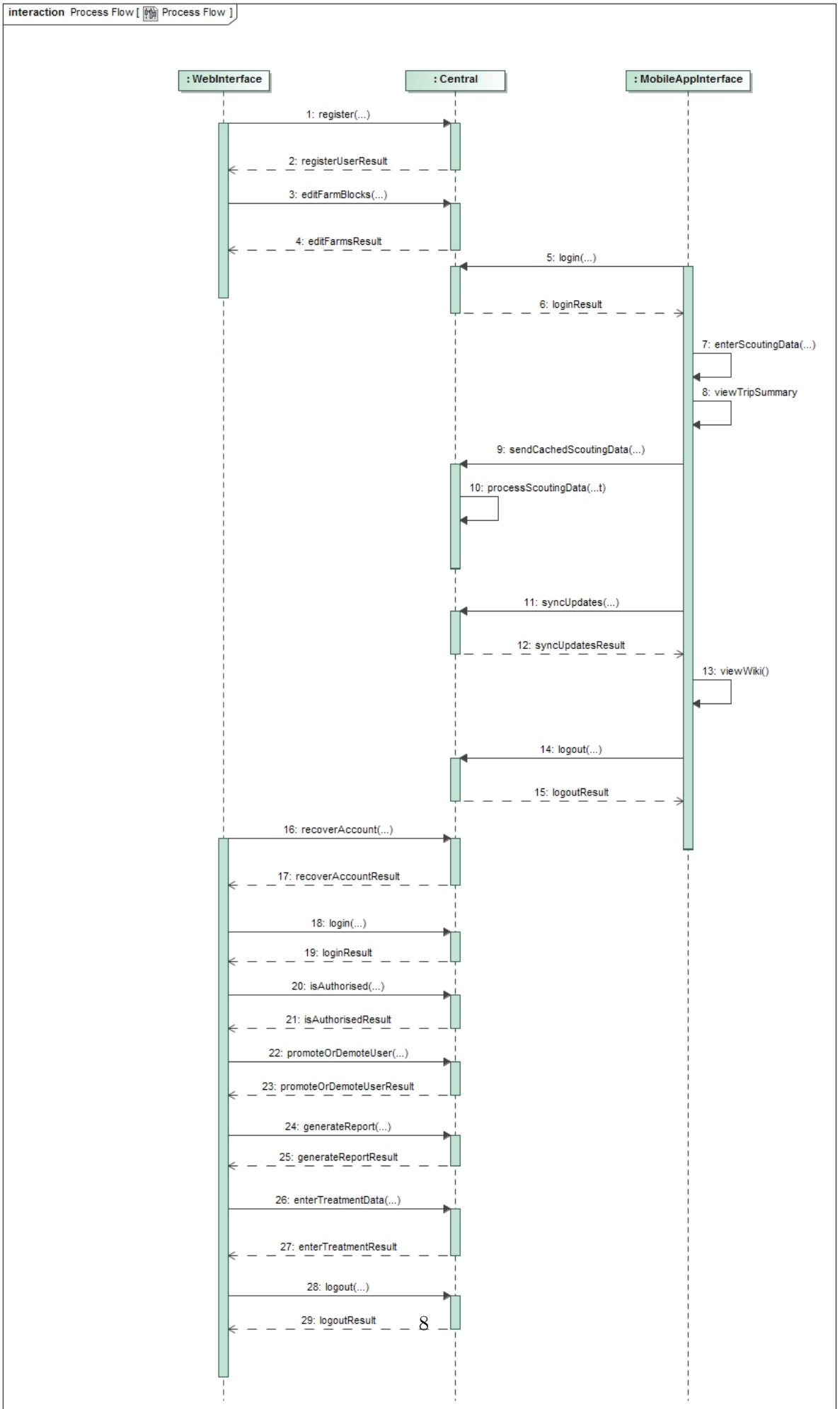
2.2 Infrastructure Layout



2.3 Database Design



2.4 Process Flow



2.5 Technologies

2.5.1 Mobile Application

2.5.1.1 Android Studio

We chose to use this development suite because of the following:

- It is the industry standard for Android app development.
- The IDE provides lots of code analysis with the tool named Lint. It also has a very good dynamic layout preview to design the interfaces for specific screen variations.
- Furthermore, we decided to develop the app natively instead of using a cross-platform alternative like Titanium because the client specified that only an Android app would be required and therefore it seems pointless to deal with hassles that may arise in cross-platform development.
- A large focus for our group, specifically in the mobile app, is a high quality user interface. This particular aspect is something we stand to lose if we develop for cross-platform since we would only be able to use common controls between Android and iOS, for example.

For more information regarding Native vs. Cross-platform development, click here.

2.5.1.2 Mockito

There are many Mocking Frameworks available for Android App development. Mockito stands out from the others in the following respects:

- Mockito is simple to use, implement and is a lot more readable, especially when compared to the likes of RhinoMock, JMockit and EasyMock. These other frameworks are are old, unpleasant and difficult to find documentation for.
- Mockito not only allows interfaces to be mocked, but also allows abstract and concrete classes to be mocked. Furthermore, this can be done without any extensions (e.g. EasyMock requires class extentions.)
- Since it is a single Jar file, it is easy to setup.
- Mockito supports spies (to create partial mocks), in order to gather indirect output from a test, meaning functions can be tested and verified without giving the test input.

For more information regarding Mockito vs. EasyMock, click here.
For more information regarding the pros and cons of Mockito, click here.
For more information regarding Mockito vs. PowerMock/JMockit, click here.

2.5.1.3 SQLite

SQLite is a relational database management system which is contained in a C programming library. It was selected because of the following:

- It is available for free.
- It is Portable, because the whole database is located on a single file making easy to transfer.
- It is easy to query data, because data is stored in a simple structured way.

- SQLite has higher performance than many other relational db management systems and uses a low amount memory.
- It is easier to manage since there is no file parsing and generated code that needs to be debugged.
- The content can easily be viewed using various third-party tools.

For more information regarding a comparison of SQLite and other relational db management systems, click here.
For more information regarding why SQLite is used in Android App Development, click here.

2.5.2 Web Backend

2.5.2.1 .NET WebAPI

The Microsoft .NET framework was chosen as a backend framework for hosting RESTful services. The following factors contributed to this decision:

- The .NET framework is a tried and tested industry standard technology which adds value to the Reliability component of the quality requirements.
- The .NET framework and packages used within the project became open source as announced by Microsoft at the Build 214 conference thus freeing the client from licensing fees. See the .NET Foundation for further details on open source Microsoft Packages.
- The main alternative to the .NET framework was the Java EE reference framework. Since Performance and Scalability is a very important quality requirement .NET was deemed superior due to case studies consulted:
  - Testing .NET vs Java EE with basic web pages
  - How to pick between Java and .NET
  - .NET vs Java Discussion

  It is worth noting that Java EE and .NET are very closely matched in many aspects. The difference between the two frameworks are nearly negligible in terms of performance.
- The .NET framework provides seamless integration with the other technologies utilised
- The .NET framework provides for quick and easy set-up for REST services and web front ends using MVC5

2.5.2.2 AutoMapper

AutoMapper is used to do automated maping from DTO's (Domain Transfer Objects) to Domain (or Data) objects, and vice versa. This is simply used to promote code reusability and readability.

2.5.2.3 .NET MVC 5

For the same reasons mentioned for the choice of .NET WebAPI. MVC 5 was chosen as a pattern at a lower level of granularity due to the built in support for it in the .NET framework and it can be viewed in most cases as a industry standard for web applications.

2.5.2.4 MSSQL Express

Express is the free version of the Microsoft RDBMS which caters for all the data needs of the system. This technology can easily be swopped out for other free database technologies such as MySQL due to the technology neutral architecural design.

2.5.2.5 AutoFac

AutoFac is used as a mocking framework for unit testing as well as for dependency injection in order to fascilitate more effective unit testing and pluggability. Using this dependency injection framework code maintainability is also improved.

2.5.3 Web Frontend

2.5.3.1 DataTables

DataTables is a plug-in for the popular jQuery Javascript library. It enables powerful DOM manipulation and data-binding specifically for data-driven web pages. Since the SAMBUG Website mainly features a dashboard that enables growers to peer into their bug collection and treatment history, tables and charts will be at the heart of the Web front-end. This library is specifically for dynamic table creation from multiple data sources.

2.5.3.2 Chartist-js

Chartist is a JavaScript charting library providing flexible and responsive charts for our Reporting Module. Chartist prefers convention over configuration and although it's compact and focused at its core, its functionality may be extended via plug-ins. This libary draws its chart elements in the Scalable Vector Graphics format, allowing crisp images, independent of device specifications.

2.5.3.3 AngularJS

Angular is the next generation JavaScript client framework for mobile, desktop and web. It provides two-way databinding (as does Aurelia.io) and extensible HTML as well as other great features such as client side dependency injection . Angular is the way forward for client side scripting frameworks. The other major competitor in this area is AureliaIO. See Aurelia vs AngularJS Round One: FIGHT!

2.5.4 Builds and Deployment

2.5.4.1 AppHarbor

This is a hosted .NET Platform as a Service. AppHarbor attaches to our version control system (Git) and monitors commits to the master branch. It automatically builds the latest source, runs unit and integration tests and finally deploys when the tests have all succeeded. AppHarbor also hosts our MSSQL development database.

2.5.4.2 Gradle

This build system is the standard for Android Studios and therefore it allows us to easily get apps built and running. Testing and debugging apps is made really simple using Gradle. It also allows us to perform incremental building.

2.5.4.3 MSBuild

MSBuild is the standard build script language for the .NET development environment. Since it functions seamlessly there is no need to change this technology.

2.5.5 Testing

2.5.5.1 JUnit

Junit was selected as the Unit Testing framework because:

- It is simpler to use and has better online documentation than competitors such as TestNG.
- Junit is the most popular testing framework in Industry.
- It is easy to integrate Junit as part of your build process.
- Since it is the most popular testing framework, new features are constantly being added.

For more information on why JUnit was chosen compared to other Testing Frameworks, click here and here.

2.5.5.2 MSTest

MSBuild is the standard testing pacakge for the .NET development environment. Since it functions seamlessly there is no need to change this technology.

2.5.6 Artifical Intelligence and Computer Vision

2.5.6.1 OpenCV

OpenCV is a open source library which focuses on computer vision applications and artifical intelligence. It is believed to be the most popular and reliable open source library in the field. The OpenCV library was used for preprocessing of images for the neural network which is also supplied via OpenCV. EmguCV was used as .NET bindings for the OpenCV library.

2.5.7 Additional Technologies

2.5.7.1 Fiddler 4

Fiddler is a .NET orientated debugging program similar to WireShark which is used to monitor HTTP requests and set up custom requests to any specified server in almost any specific form (GET, POST, PUT etc.)

2.5.7.2 Git and GitHub

Git is the version control system of choice which is utilised for the system. The public repository from whch the code base is hosted is named GitHub and may be accessed at `https://github.com/WMostert1/Vector-Software-Developers`.

2.5.7.3 Material Design

Material Design is a design language that was developed by Google used to make the experience of viewing a mobile or web application more real and alive. We have decided with this technology as we believe it can make our system only better, especially when Material Design

and implementations thereof has had time to grow. We believe it can make our system visually more appealing and also make our system more accessible.

Angular Material will be used on the web interface. Angular material is an implementation of Material Design.

2.5.7.4 Misc. Javascript Frameworks and Libraries

Various, mostly JavaScript, libraries are utilised for the reporting services and user interface.