

SAMBUG

Master Specification

COS 301



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

October 2015

Contents

1	Background	2
2	Vision	2
3	Scope	2
4	Functional Requirements and Application Design	3
4.1	Domain Model	3
4.2	BugScouting	3
4.2.1	Module Scope	4
4.2.2	Use Cases	4
4.3	BugIntelligence	8
4.3.1	Module Scope	8
4.3.2	Use Cases	8
4.4	BugSecurity	11
4.4.1	Module Scope	11
4.4.2	Use Cases	11
4.5	BugReporting	15
4.5.1	Module Scope	15
4.5.2	Use Cases	15
5	Architectural Requirements	18
5.1	Quality Requirements	18
5.1.1	Maintainability	18
5.1.2	Scalability	18
5.1.3	Performance Requirements	18
5.1.4	Reliability and Availability	18
5.1.5	Security	19
5.1.6	Auditability	19
5.1.7	Testability	19
5.1.8	Usability	19
5.1.9	Deployability	19
5.2	Architecture responsibilities	19
6	Architectural Design	19
6.1	Overview	19
6.2	Infrastructure Layout	21
6.3	Database Design	22
6.4	Process Flow	24
6.5	Technologies	25
6.5.1	Mobile Application	25
6.5.2	Web Backend	26
6.5.3	Web Frontend	27
6.5.4	Builds and Deployment	27
6.5.5	Testing	28
6.5.6	Artifical Intelligence and Computer Vision	28
6.5.7	Additional Technologies	28

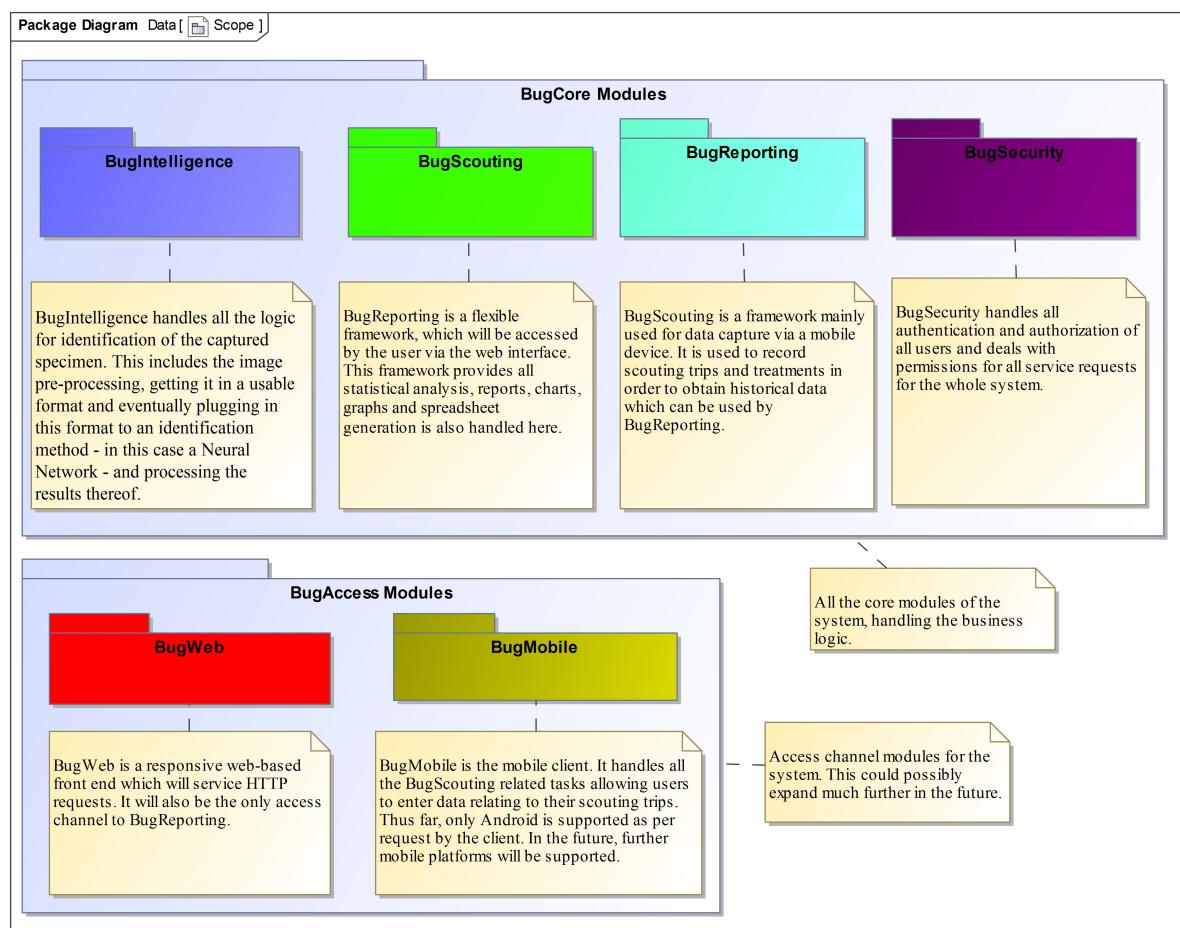
1 Background

South Africa is currently the largest producer of macadamia nuts in the world. One of the main production and quality limiting factors is the incidence of stink bug damage. Accurate scheduling of chemical sprays rely on accurate scout data and threshold levels of the insect pests in an orchard. However, scouting for these pests has a major shortfall, namely the accurate identification of pests, despite efforts to train growers and scouts by various means. Area wide control of pests and diseases is a concept that has been considered, but with the lack of scout data from across and within growing regions it is impossible to make such recommendations.

2 Vision

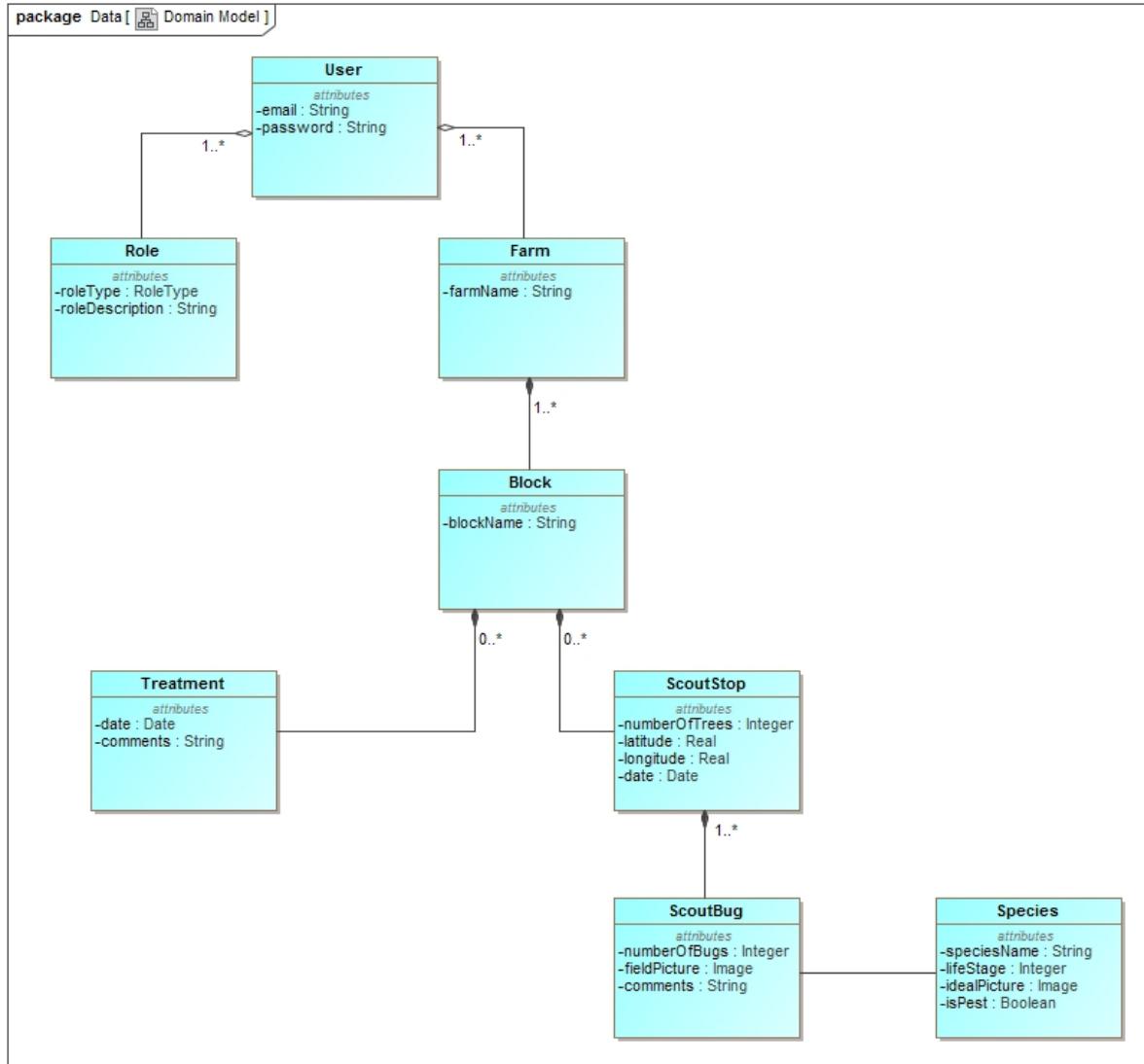
An innovative approach to handling the management and acquisition of scout data is to develop a smartphone application that is able to identify specific stink bug species by making use of the built-in camera of the smartphone. This application makes use of the smartphones built-in GPS to perform geotagging and uploading information to a central database. In addition to the data management, one of the primary goals is to supply data mining methods on historical data as obtained via the smartphone application. A visual representation of said data is therefore made available in a dynamic and easy way for the every day farmer and researcher.

3 Scope



4 Functional Requirements and Application Design

4.1 Domain Model



4.2 BugScouting

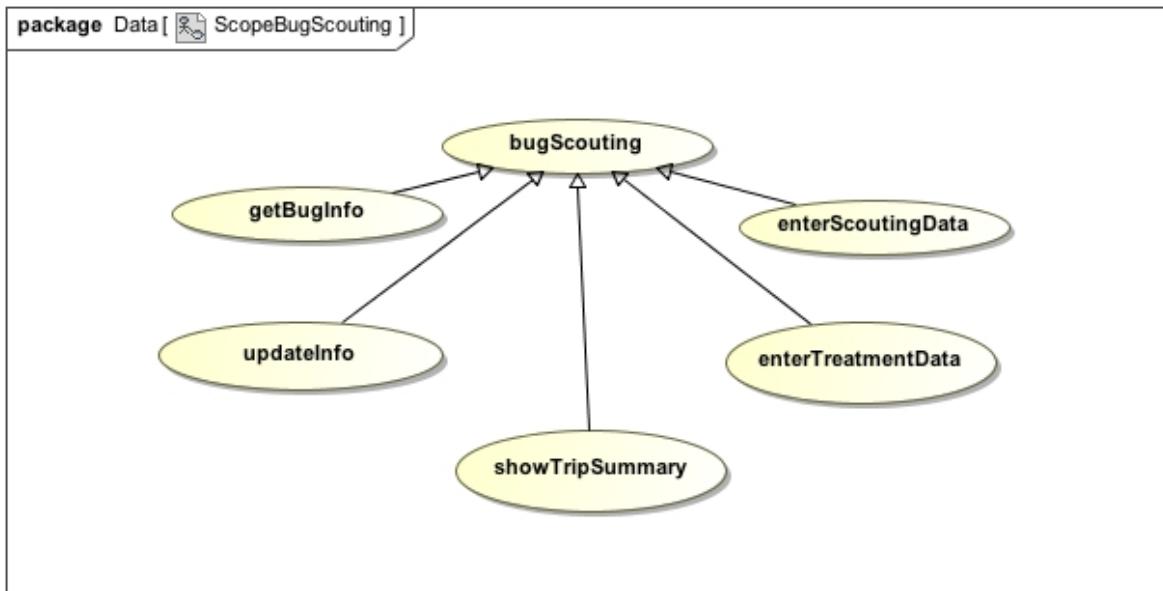
The BugScouting module has the following functionality:

1. It provides the functionality for a user to be able to enter data related to a scouting trip - entering data such as the number of trees scouted, the number of bugs per tree and the different kind of bugs specified. After this data has been entered, a summary should be displayed for the user.
2. It provides the ability for a user to record the details related to spraying, such as the date of spraying and the specific block that was sprayed.

In many ways this module may be regarded as the "core" functionality of the system. This encompasses the main purpose of the system.

4.2.1 Module Scope

The scope of the *BugScouting* module is shown below:



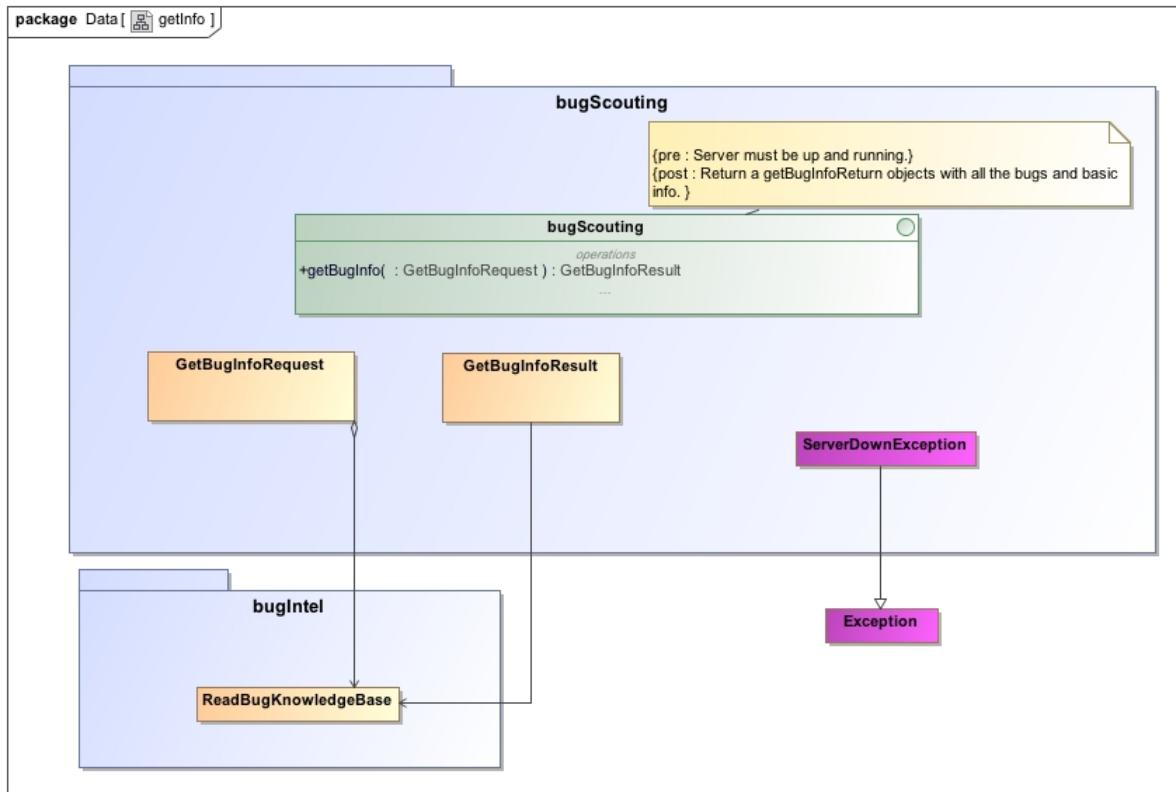
4.2.2 Use Cases

The concrete use cases for the *BugScouting* module follow:

4.2.2.1 getBugInfo [Priority - Medium]

This use case uses the BugIntelligence module to retrieve information on a specific specimen which will be used to display to the user a "wiki" like piece which has the purpose of informing the user.

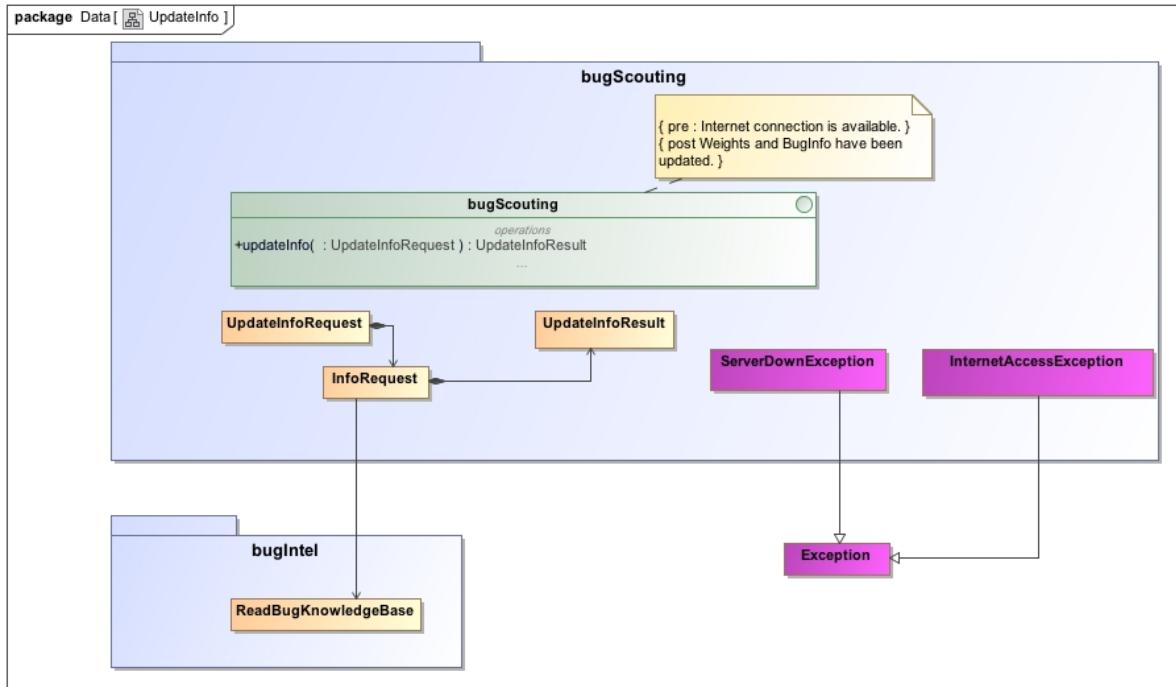
The Service Contract for the *getBugInfo* use case is shown below:



4.2.2.2 updateInfo [Priority - Medium]

The objective of this use case is to provide functionality to retrieve updated bug information and identification method updates.

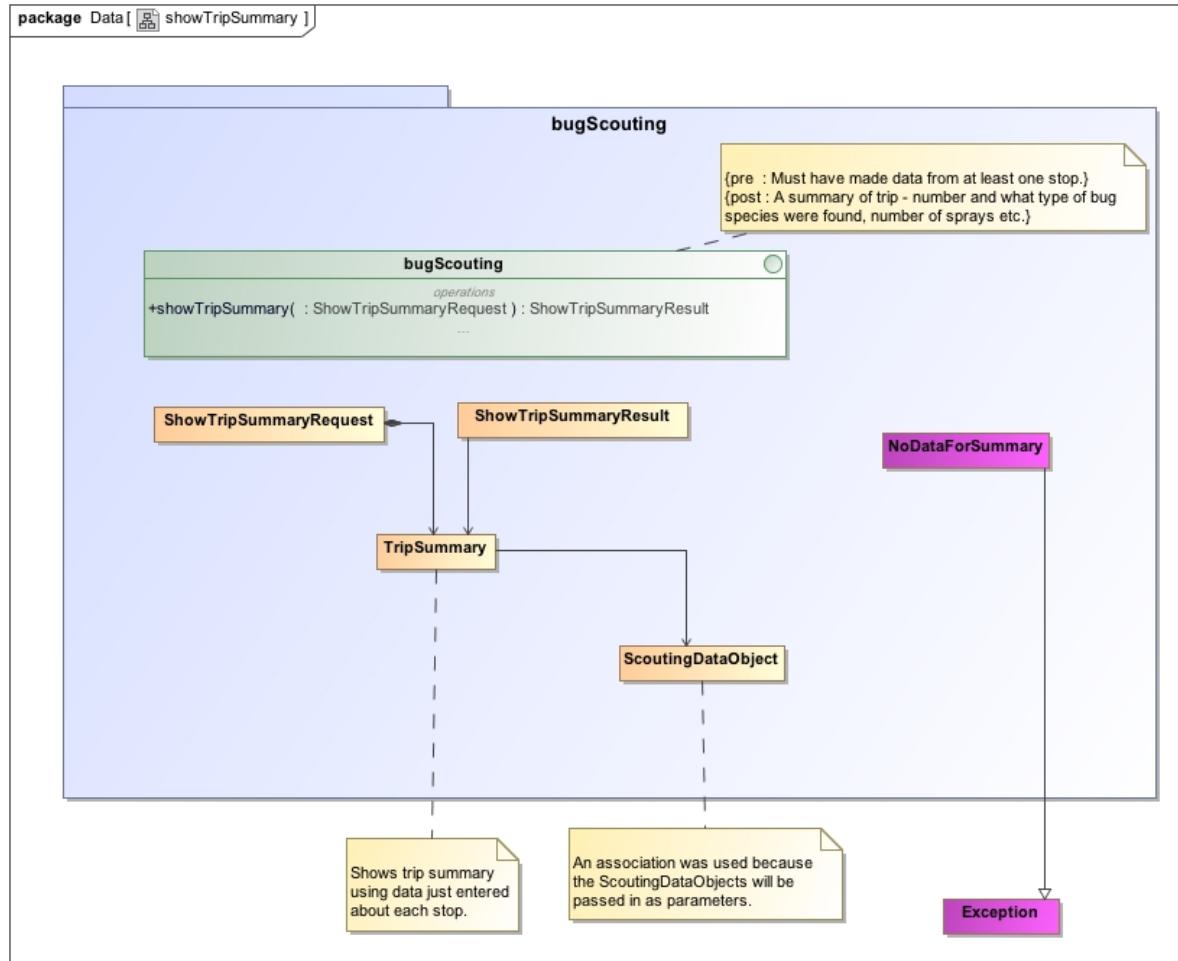
The Service Contract for the *getBugInfo* use case is shown below:



4.2.2.3 showTripSummary [Priority - High]

After completing a full scouting trip, this use case should provide a summary of the entire scouting trip. One scouting trip may consist of many scouting stops and each scout stop may include the finding of various bugs. The averages for the scouting trip data is used in the summary.

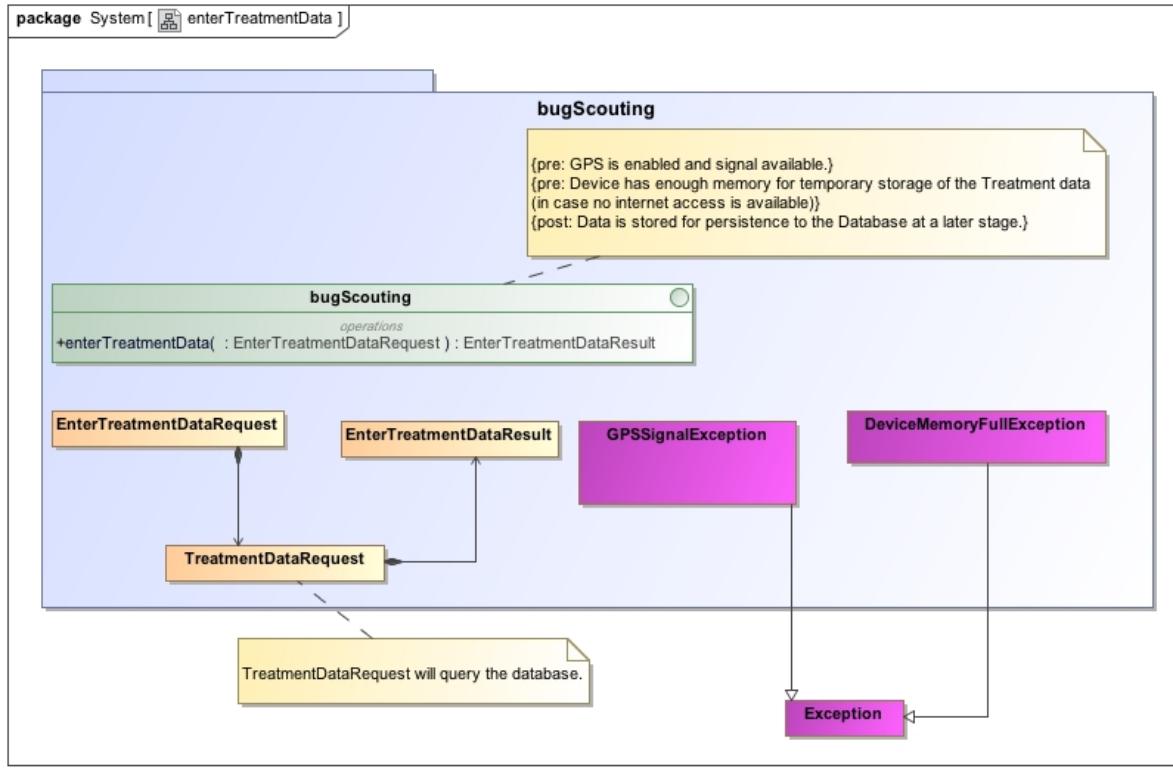
The Service Contract for the *showTripSummary* use case is shown below:



4.2.2.4 enterTreatmentData [Priority - Critical]

This use case is for entering data related to spraying chemicals (pesticide). This allows for the type of chemical, the date and the block or orchard number to be recorded.

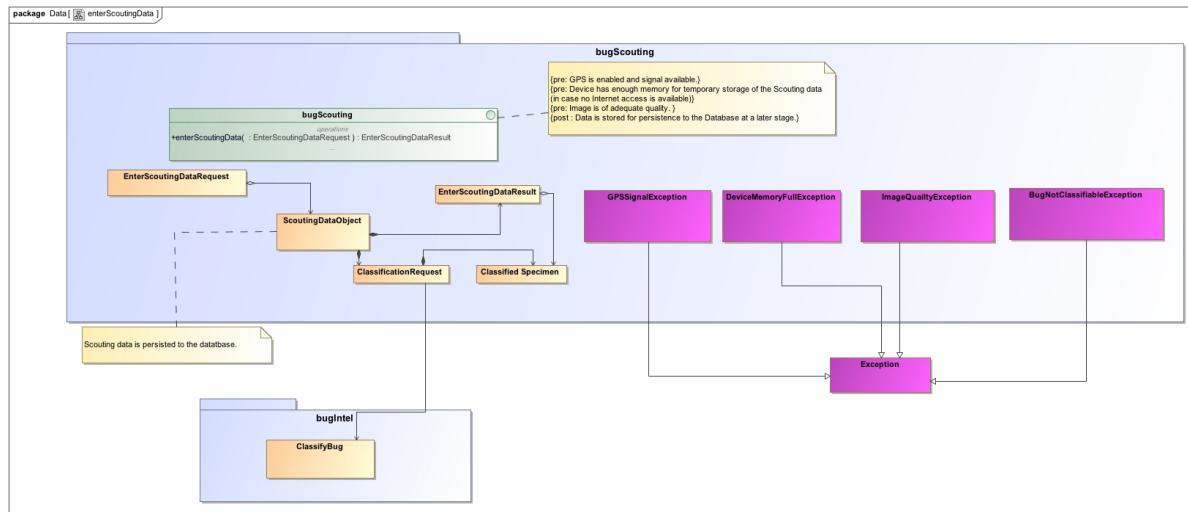
The Service Contract for the *enterTreatmentData* use case is shown below:



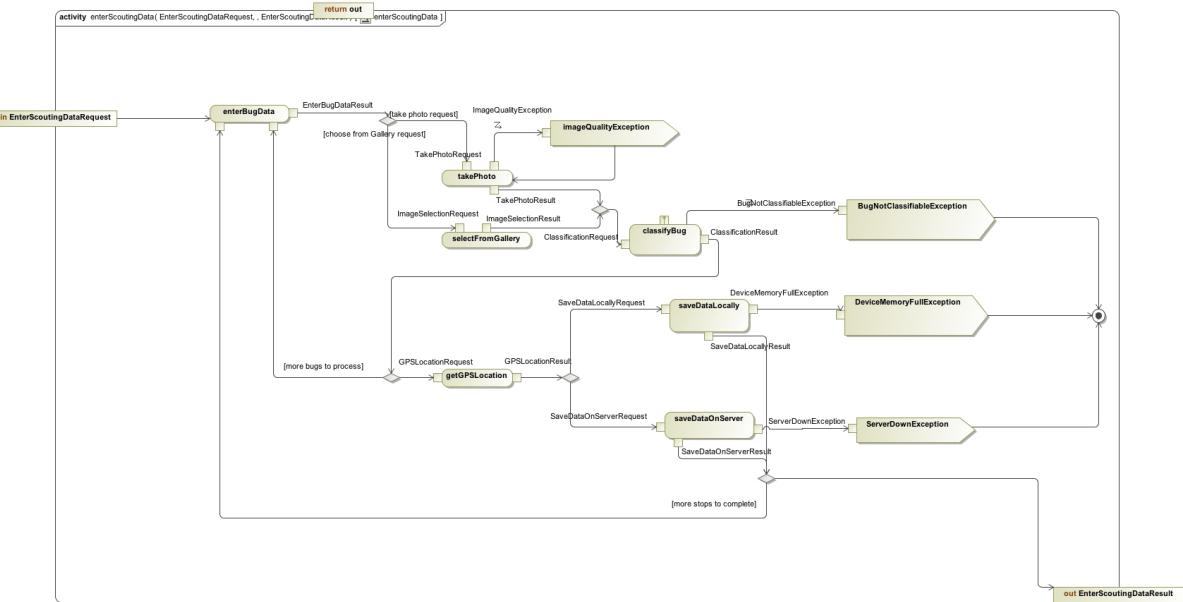
4.2.2.5 enterScoutingData [Priority - Critical]

This is the core use case of this module. This use case allows entering data related to a scouting stop. Data captured should include the number of trees observed, the number of bugs encountered and the block number where the scouting took place. After entering the data, the specimen should be identified (classified), either manually or automatically, before successfully submitting it for persistence.

The Service Contract for the *enterScoutingData* use case is shown below:



The Process Specification for the *enterScoutingData* use case is shown below:



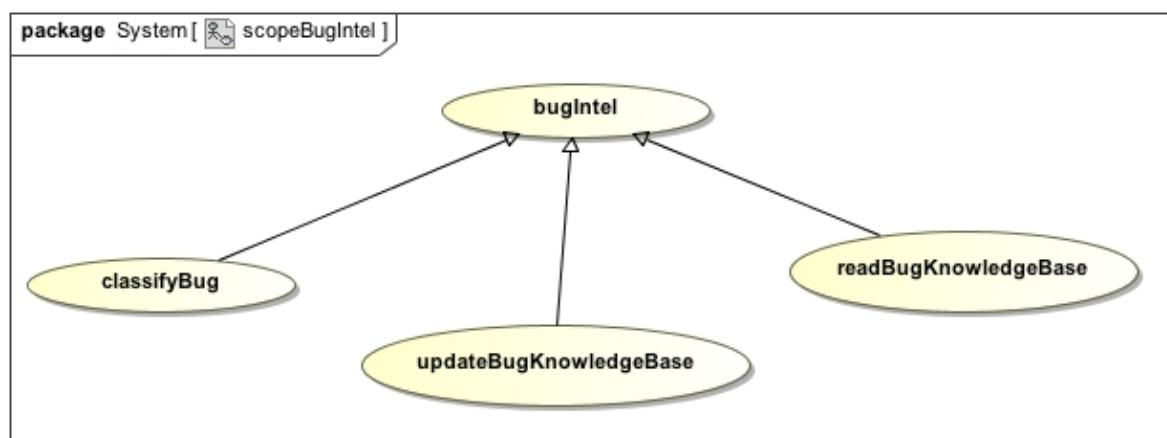
4.3 BugIntelligence

The BugIntelligence module has the following functionality:

1. It provides a pluggable method to classify a specimen according to species and life stage.
2. It provides an interface which can be used to obtain information related to any specific specimen which is identifiable by the system
3. It provides CRUD(Create Read Update Delete) functionality for bug information for specimens identifiable by the identification method

4.3.1 Module Scope

The scope of the *BugIntelligence* module is shown below:



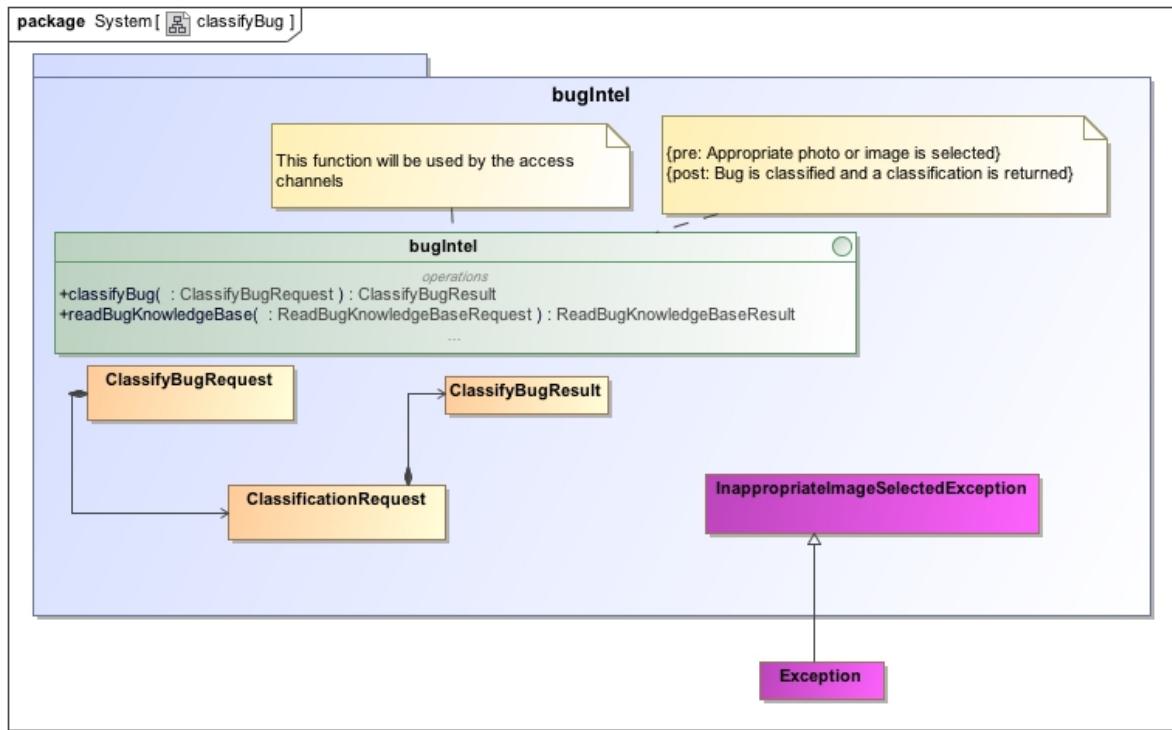
4.3.2 Use Cases

The concrete use cases for the *BugIntelligence* module follow:

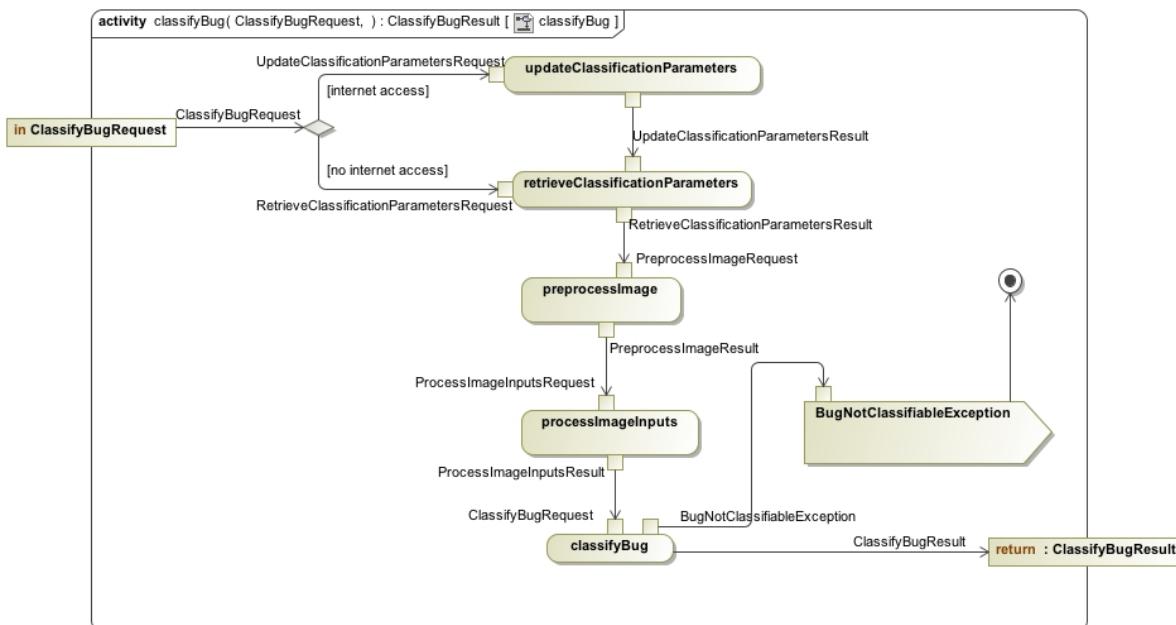
4.3.2.1 classifyBug [Priority - Critical]

This use case supplies a method to classify the bug according to life stage and species. The classification method used is pluggable.

The Service Contract for the *classifyBug* use case is shown below:

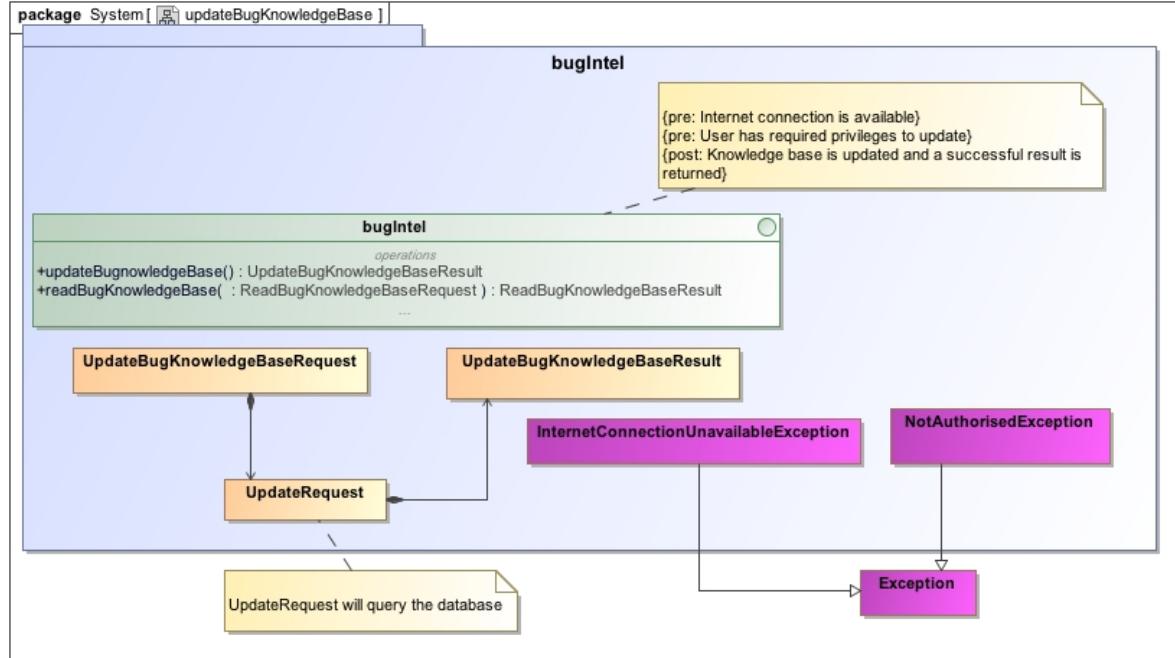


The Process Specification for the *classifyBug* use case is shown below:



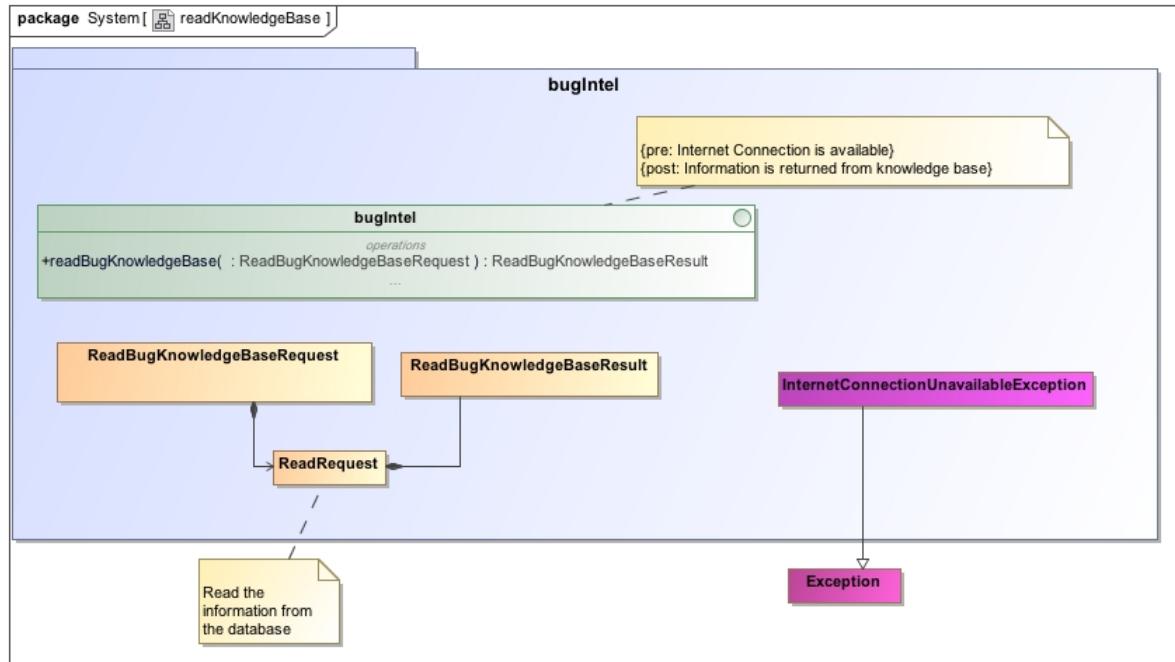
4.3.2.2 updateBugKnowledgeBase [Priority - Low]

This use case provides the functionality to edit the information used for classification. As in the example of a neural network being used as a classification method, the training examples may be edited. The Service Contract for the *updateBugKnowledgeBase* use case is shown below:



4.3.2.3 readBugKnowledgeBase [Priority - Critical]

This use case provides the functionality to get the information used for classification. As in the example of a neural network being used as a classification method, the training examples may be retrieved. The Service Contract for the *readBugKnowledgeBase* use case is shown below:



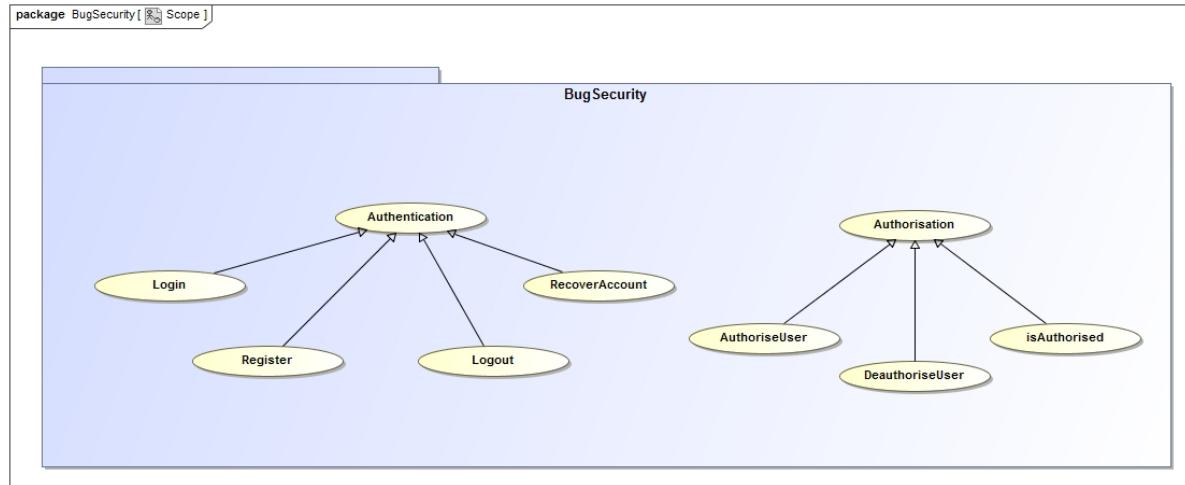
4.4 BugSecurity

The BugSecurity module allows the following functionality:

1. It provides functionality related to user accounts and user roles in order to login, register and recover your account within the capacity of a user role.
2. It provides functionality to determine whether a specific service request should be allowed.

4.4.1 Module Scope

The scope of the *BugSecurity* module is shown below:



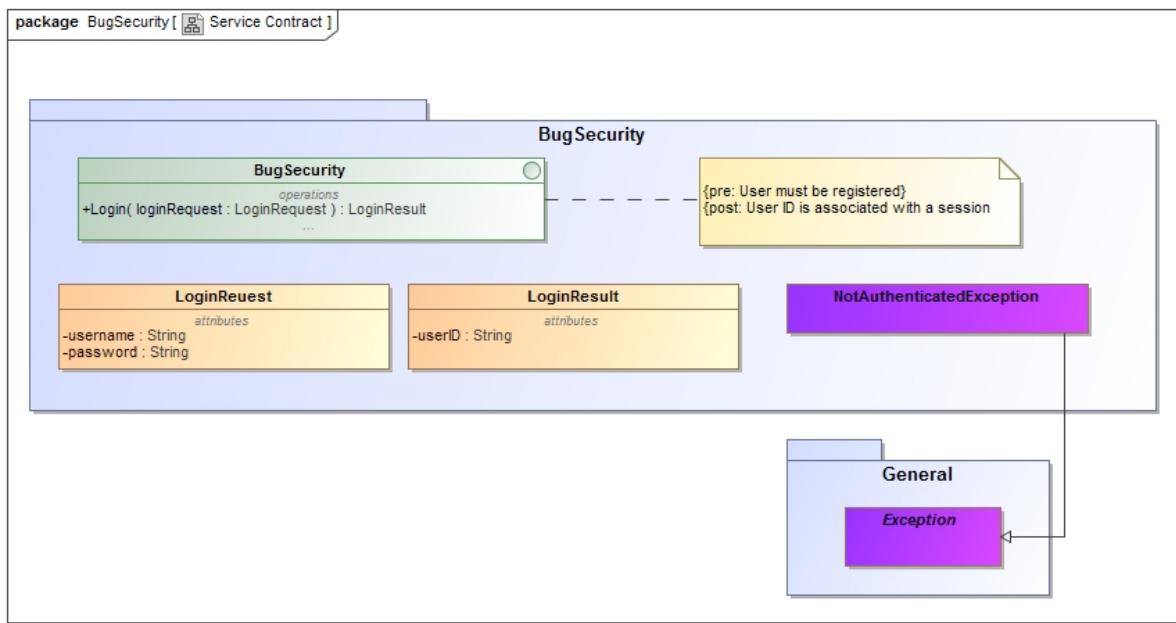
4.4.2 Use Cases

The concrete use cases for the *BugSecurity* module follow:

4.4.2.1 login [Priority - Critical]

This use case allows one to login with username and password credentials which will be validated and thus allowing the user to be authenticated.

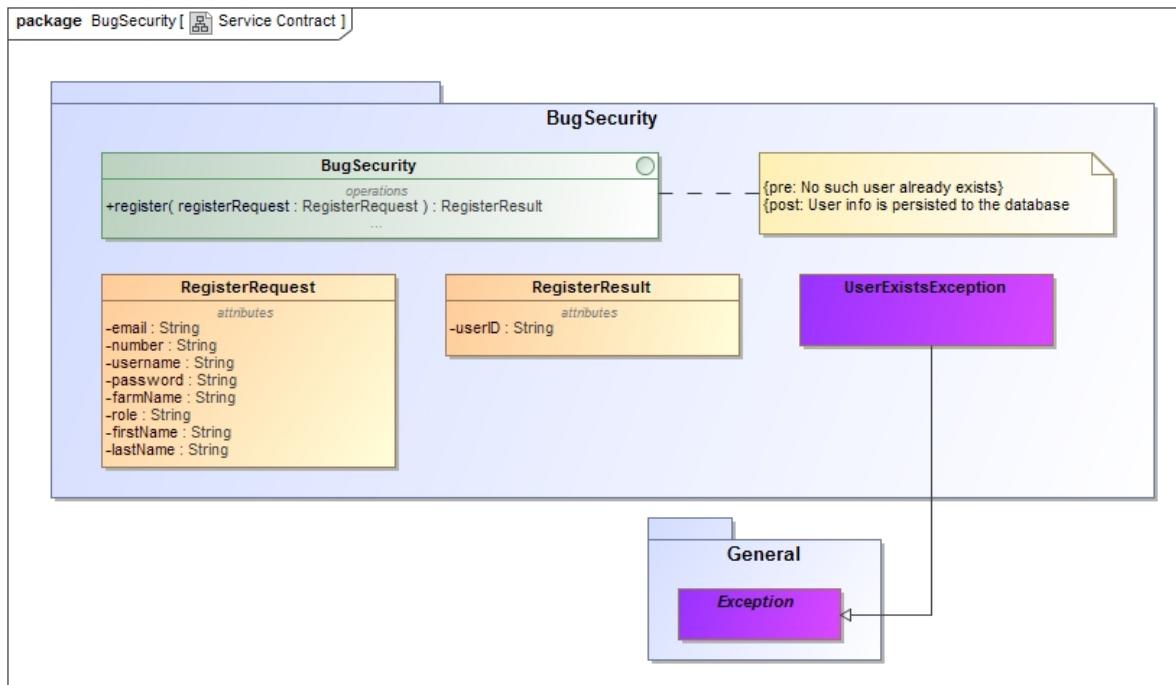
The Service Contract for the *login* use case is shown below:



4.4.2.2 register [Priority - Critical]

This use case caters for registration as a new user. A user account is created which is associated with a unique user name. Currently, there are no password format requirements.

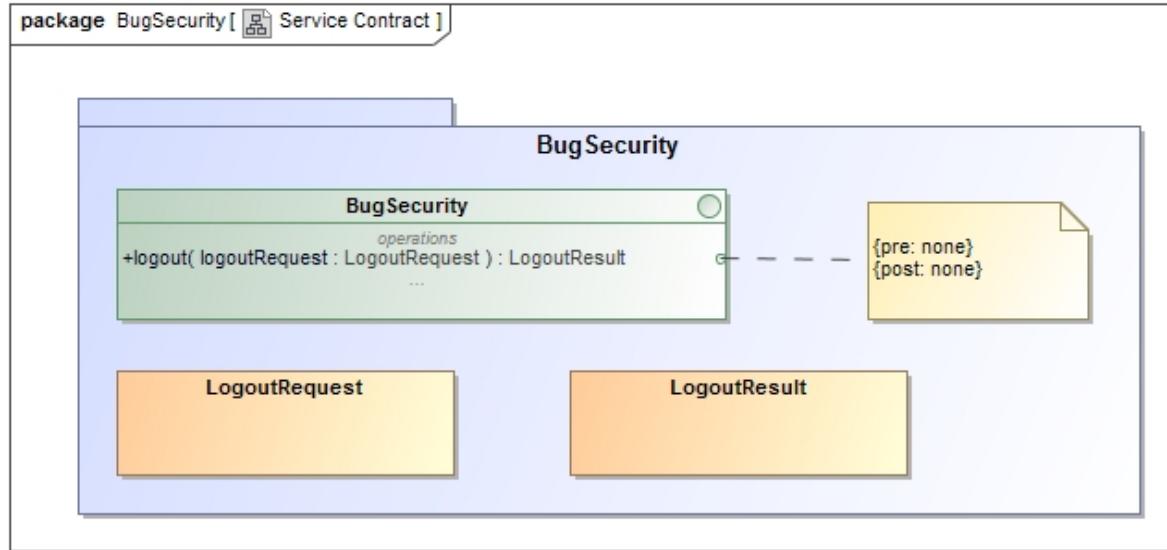
The Service Contract for the *register* use case is shown below:



4.4.2.3 logout [Priority - Medium]

This use case provides functionality to log out of the system.

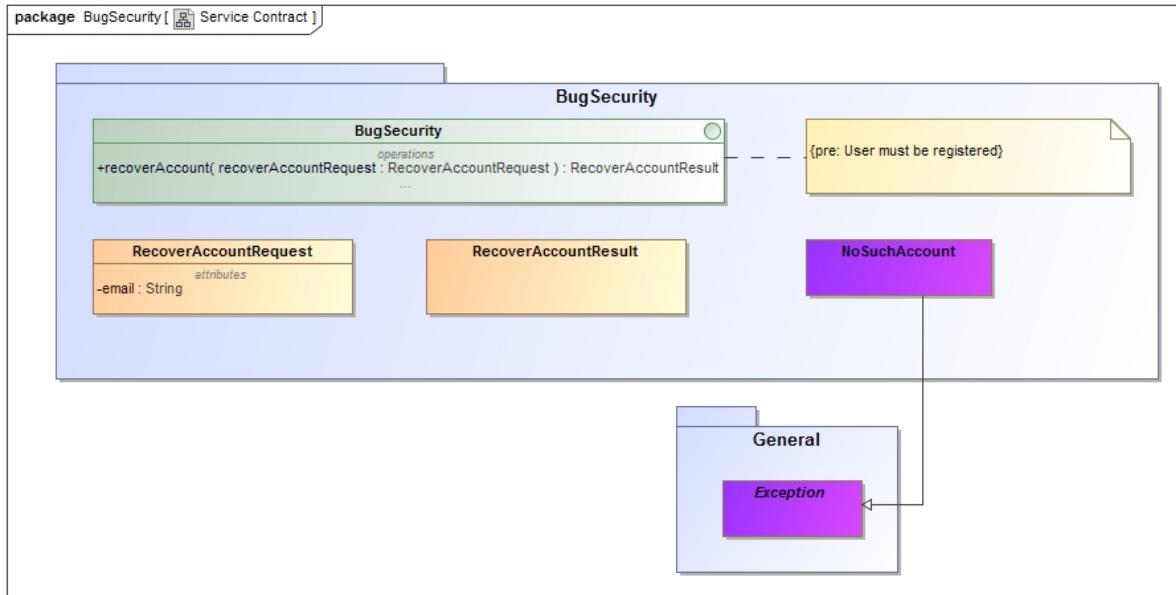
The Service Contract for the *logout* use case is shown below:



4.4.2.4 recoverAccount [Priority - Low]

In the case where a user forgets or has lost his/her credentials to access the system, this use case provides functionality to recover or to generate a new password. User verification is done via the email address associated with the account.

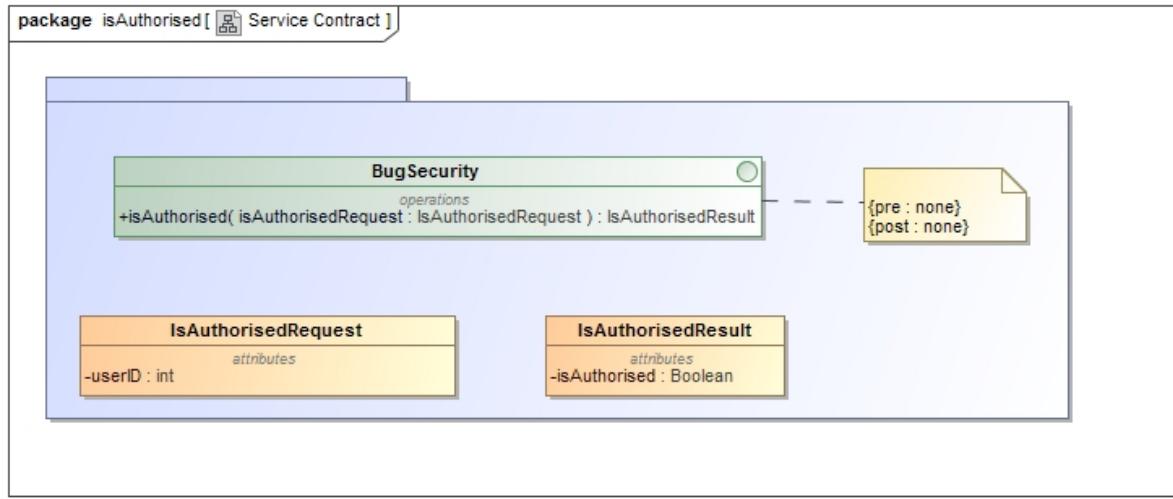
The Service Contract for the `recoverAccount` use case is shown below:



4.4.2.5 isAuthorised [Priority - High]

Whenever a service which should not be accessibility to entire plethora of users is requested, authorization is required. This use case makes provision for this.

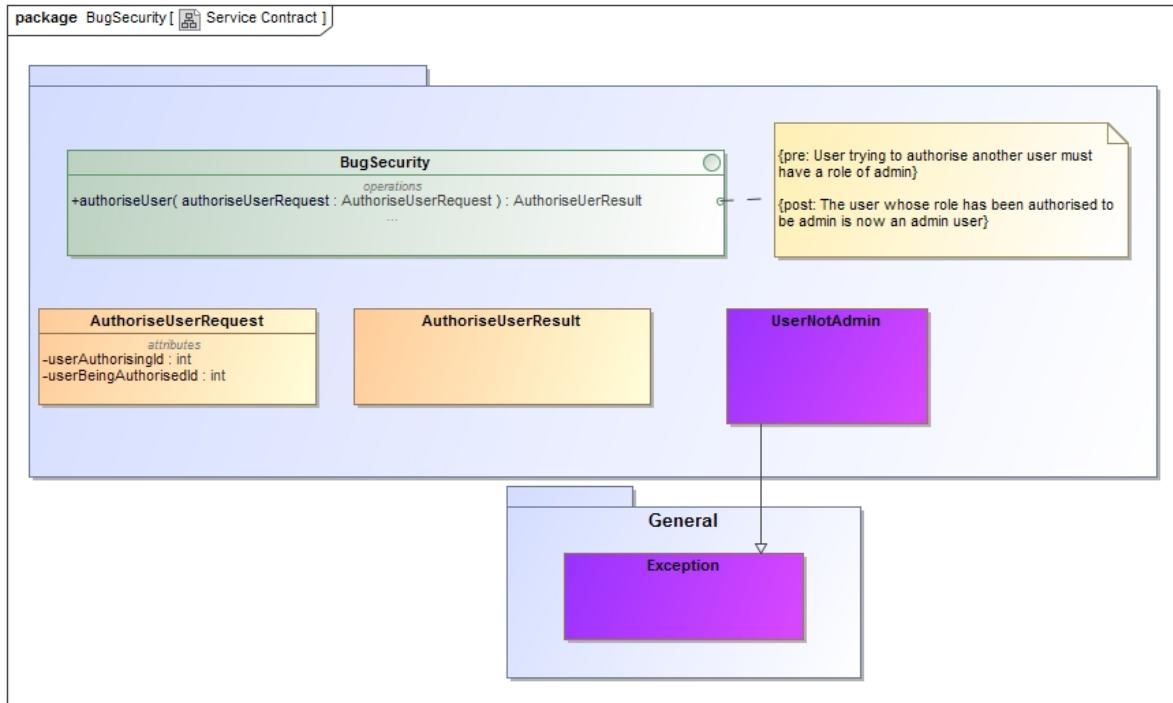
The Service Contract for the `isAuthorised` use case is shown below:



4.4.2.6 authoriseUser [Priority - Medium]

This use case allows for an authorization restriction for a service to be added/

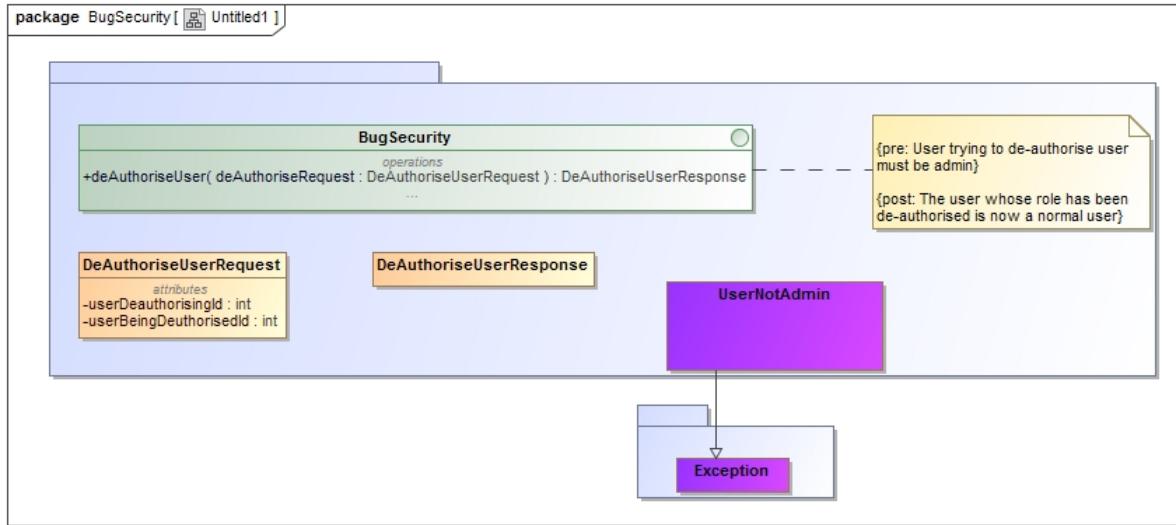
The Service Contract for the *authoriseUser* use case is shown below:



4.4.2.7 deauthoriseUser [Priority - Medium]

This use case allows for an authorization restriction for a service to be removed/

The Service Contract for the *deauthoriseUser* use case is shown below:



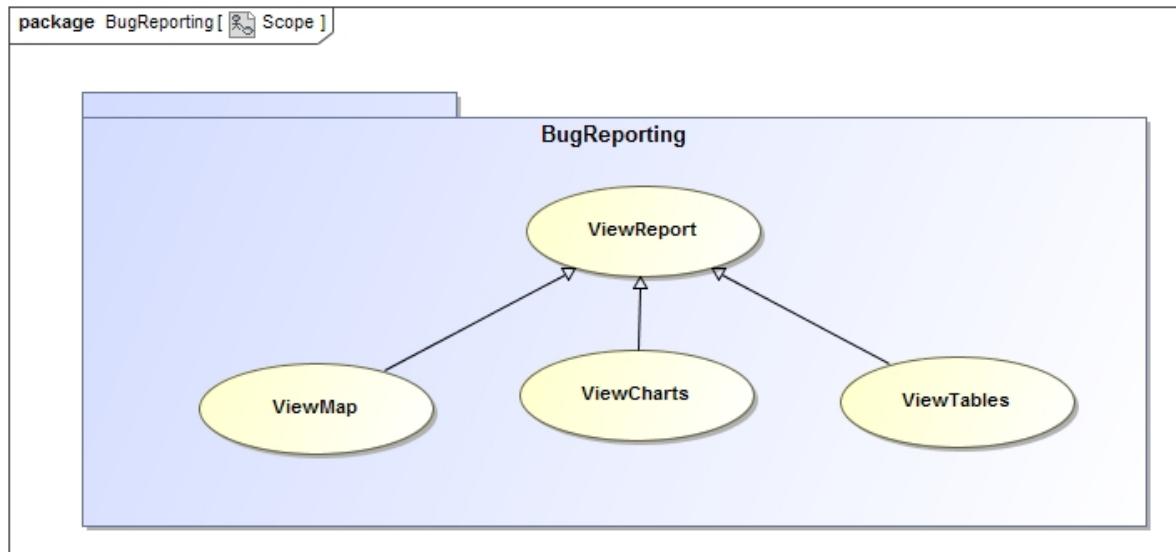
4.5 BugReporting

The BugReporting module allows the following functionality:

1. It provides functionality to generate historical data in a usable, tabular - as used by Microsoft Excel and similar software, format.
2. It provides functionality to generate a visual representation of historical data in the form of a dynamically chosen set of graphs.
3. It provides functionality to generate a heat map of the farm with regards to the population of stink bugs identified.

4.5.1 Module Scope

The scope of the *BugReporting* module is shown below:



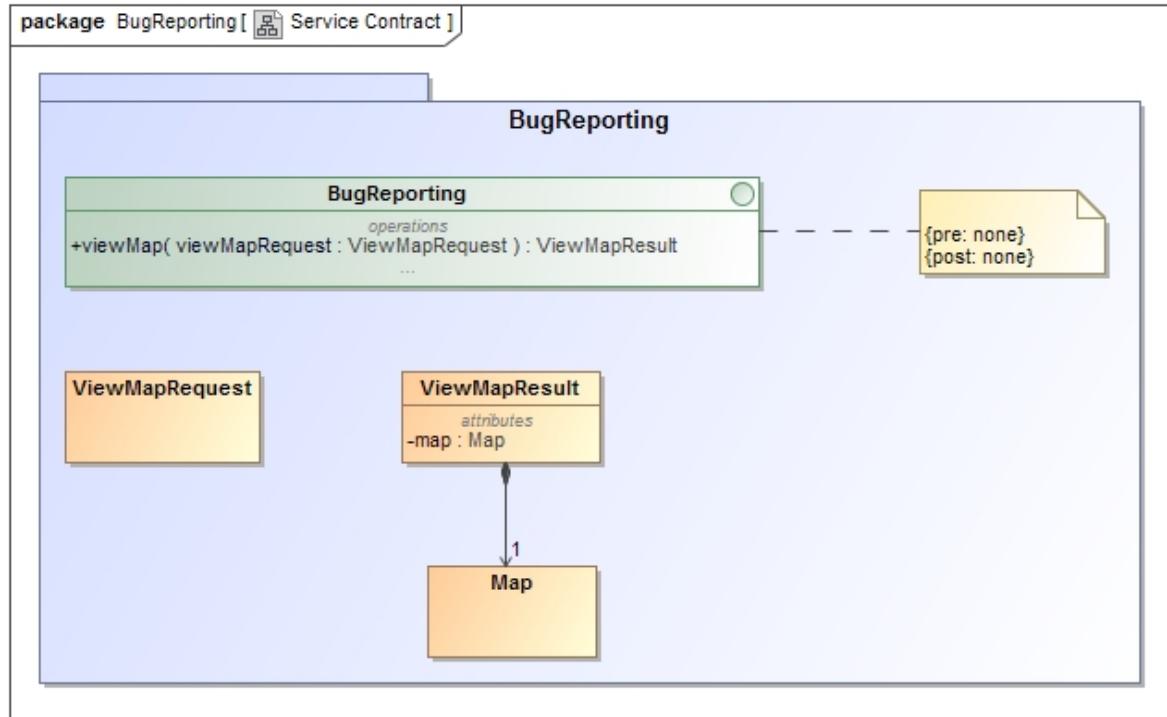
4.5.2 Use Cases

The concrete use cases for the *BugReporting* module follow:

4.5.2.1 viewMap [Priority - Medium]

This use case provides the functionality to view a heat map, where the total number of pests found will be shown. Areas with more pests will show a darker area on the map. One will this be able to conclude which areas are experiencing a problem with pests. Multiple constraints and filters can be applied to show only a subset of the data available.

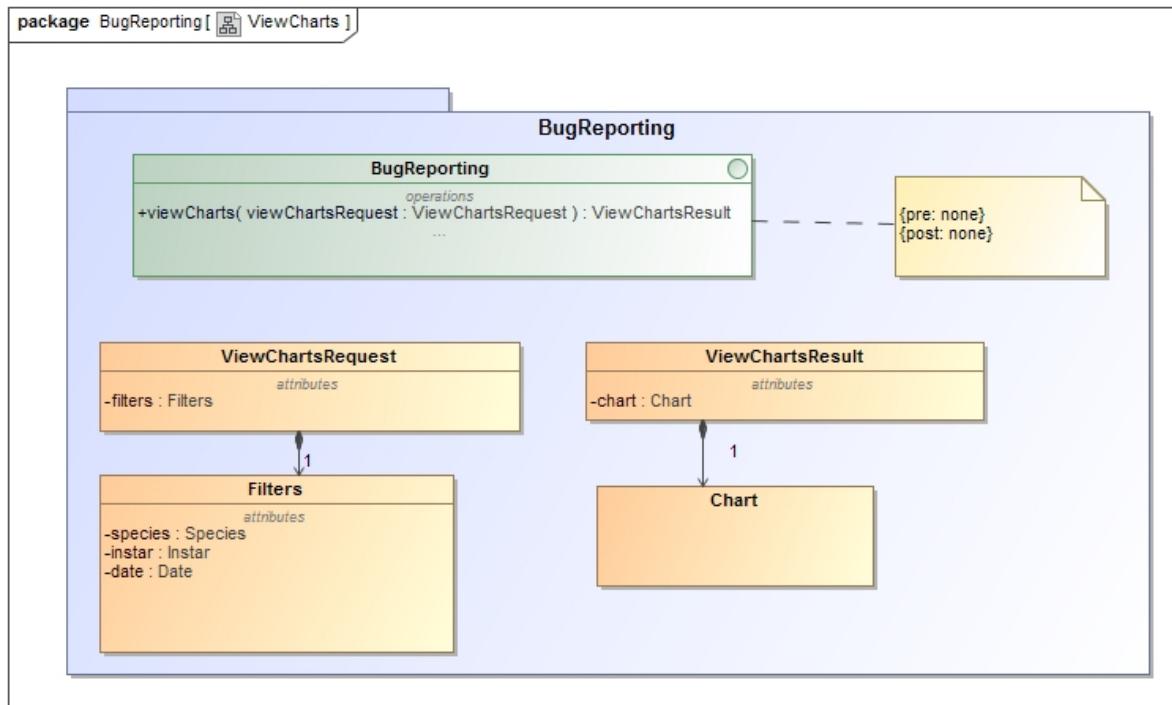
The Service Contract for the *viewMap* use case is shown below:



4.5.2.2 viewCharts [Priority - Medium]

This use case provides the functionality to view data that was collected through the mobile application, specifically scouting data. The data will be displayed in the form of dynamic charts, where one can choose multiple filters and constraints to apply to the data. One will be able to see how spraying affects the pest count.

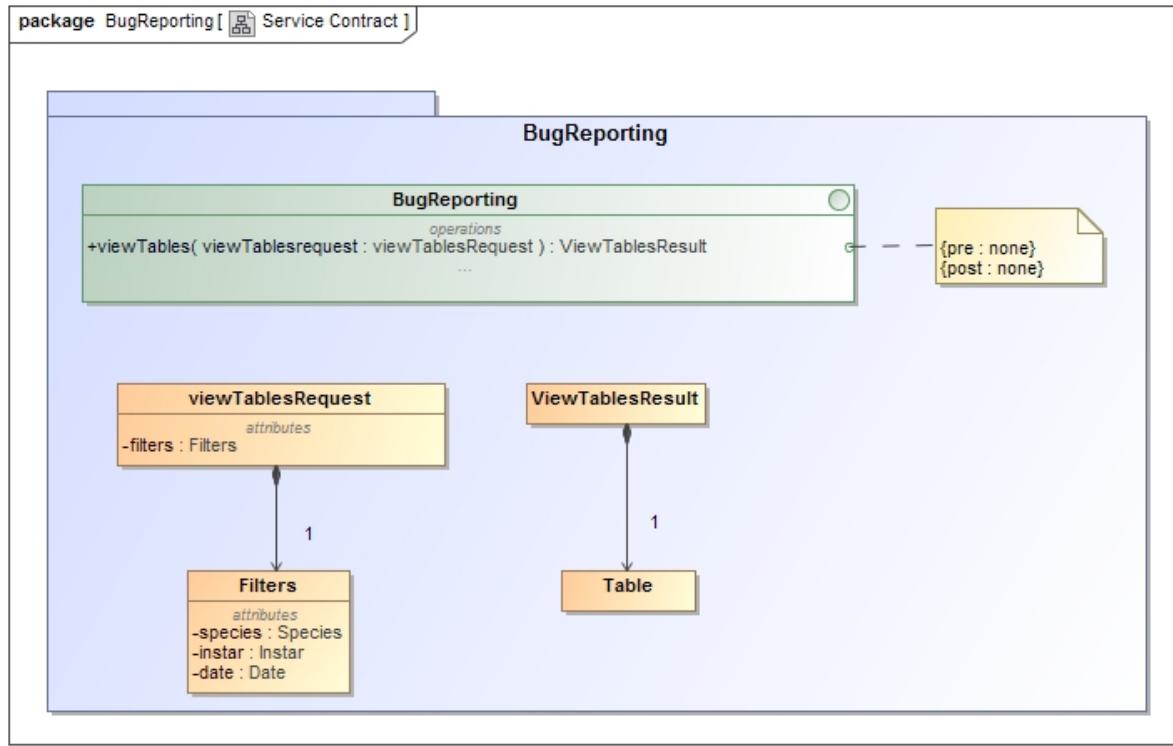
The Service Contract for the *viewCharts* use case is shown below:



4.5.2.3 viewTables [Priority - Medium]

This use case allows the user to see data that has been collected through the mobile application, specifically scouting data. The data will be displayed in the form of a table and will have the functionality to download this as an Excel spreadsheet, PDF document or to print the table. One will be able to also view spraying data. Multiple filters and constraints can be applied to the data to only view a subset of the data available.

The Service Contract for the *viewTables* use case is shown below:



5 Architectural Requirements

5.1 Quality Requirements

5.1.1 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.

5.1.2 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.

5.1.3 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.

5.1.4 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.

5.1.5 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.

5.1.6 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.

5.1.7 Testability

Every service offered by the system should be testable via:

1. Unit tests
2. Integration tests

5.1.8 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.

5.1.9 Deployability

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

5.2 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

6 Architectural Design

6.1 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

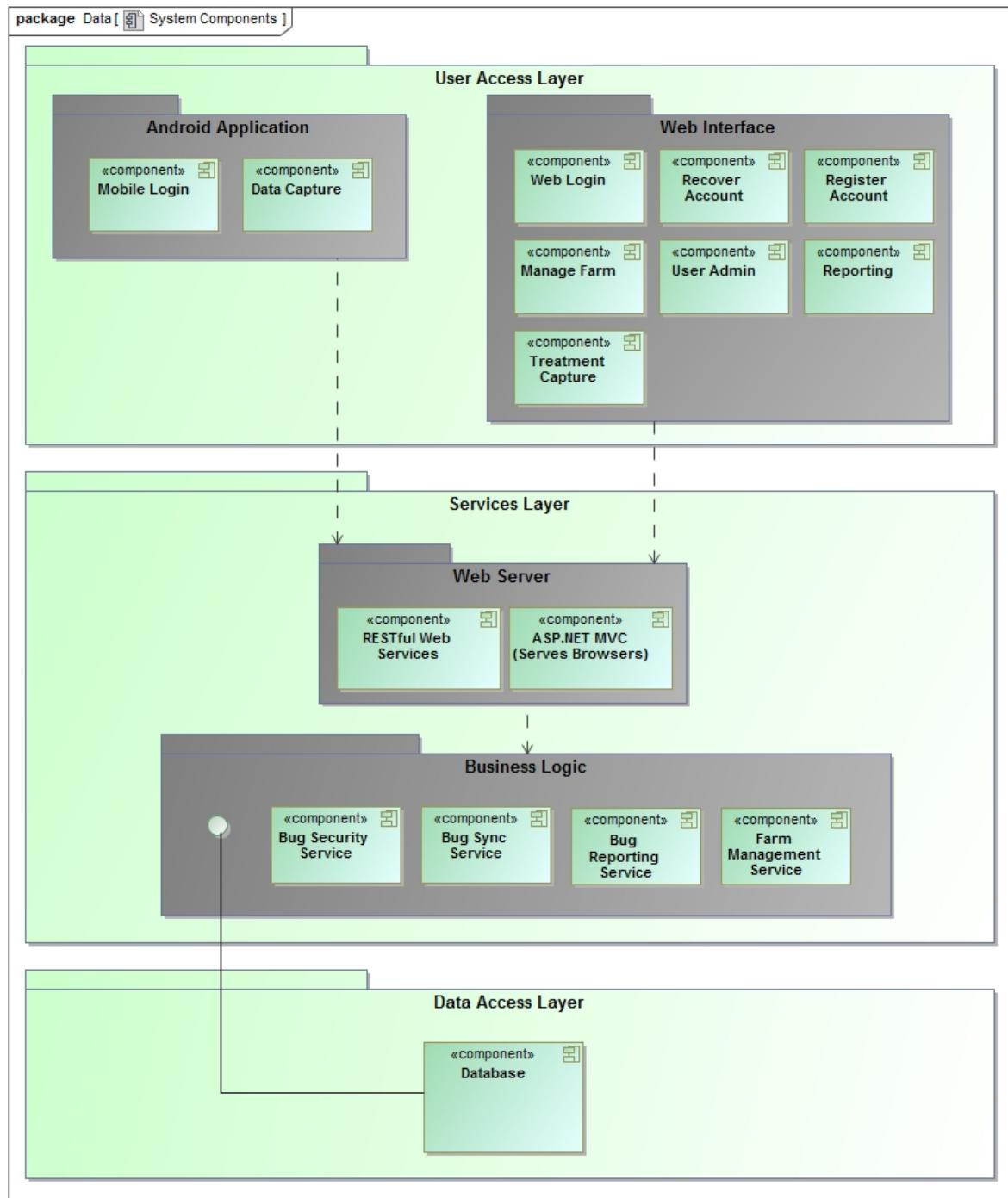
- Data Abstraction (Repository)

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.

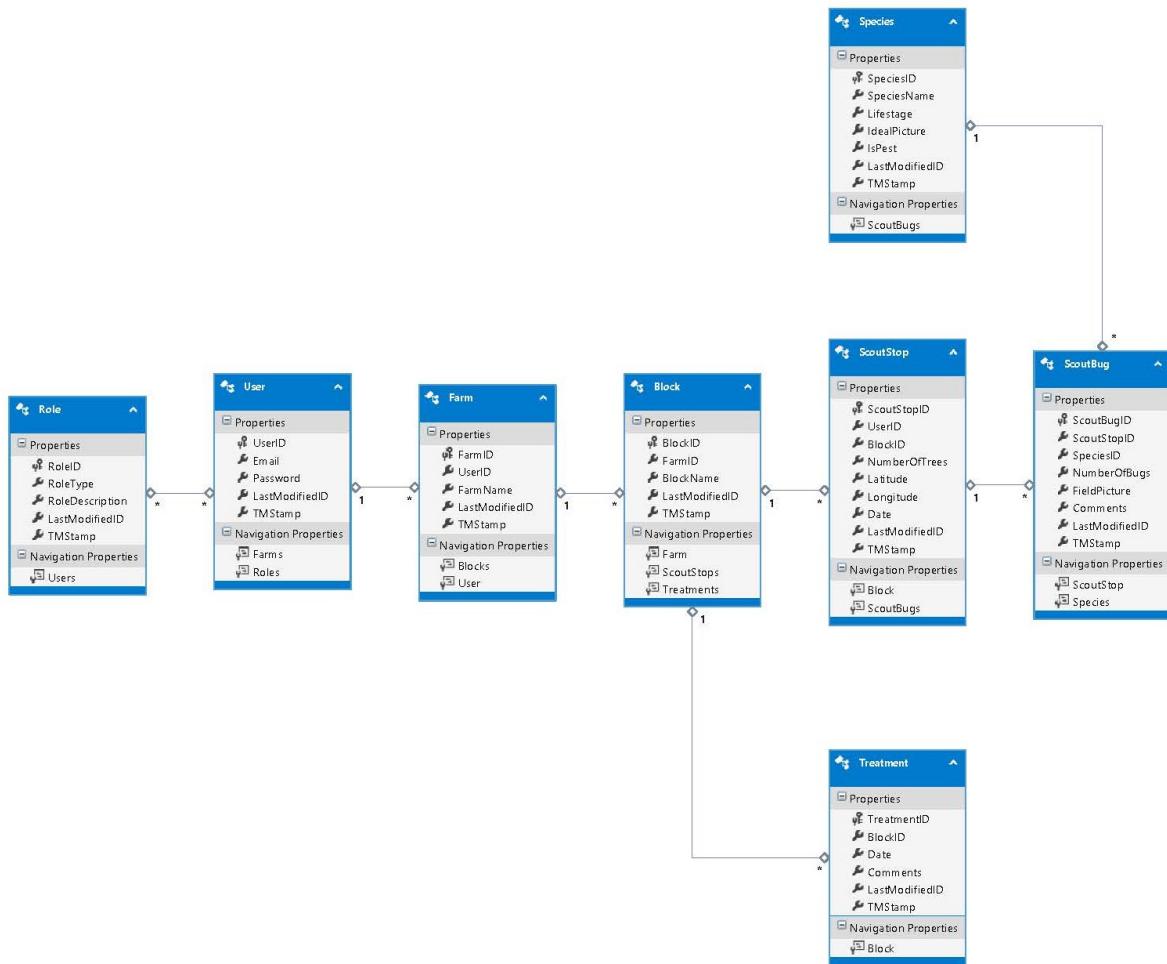
- MSSQL Implementation

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.

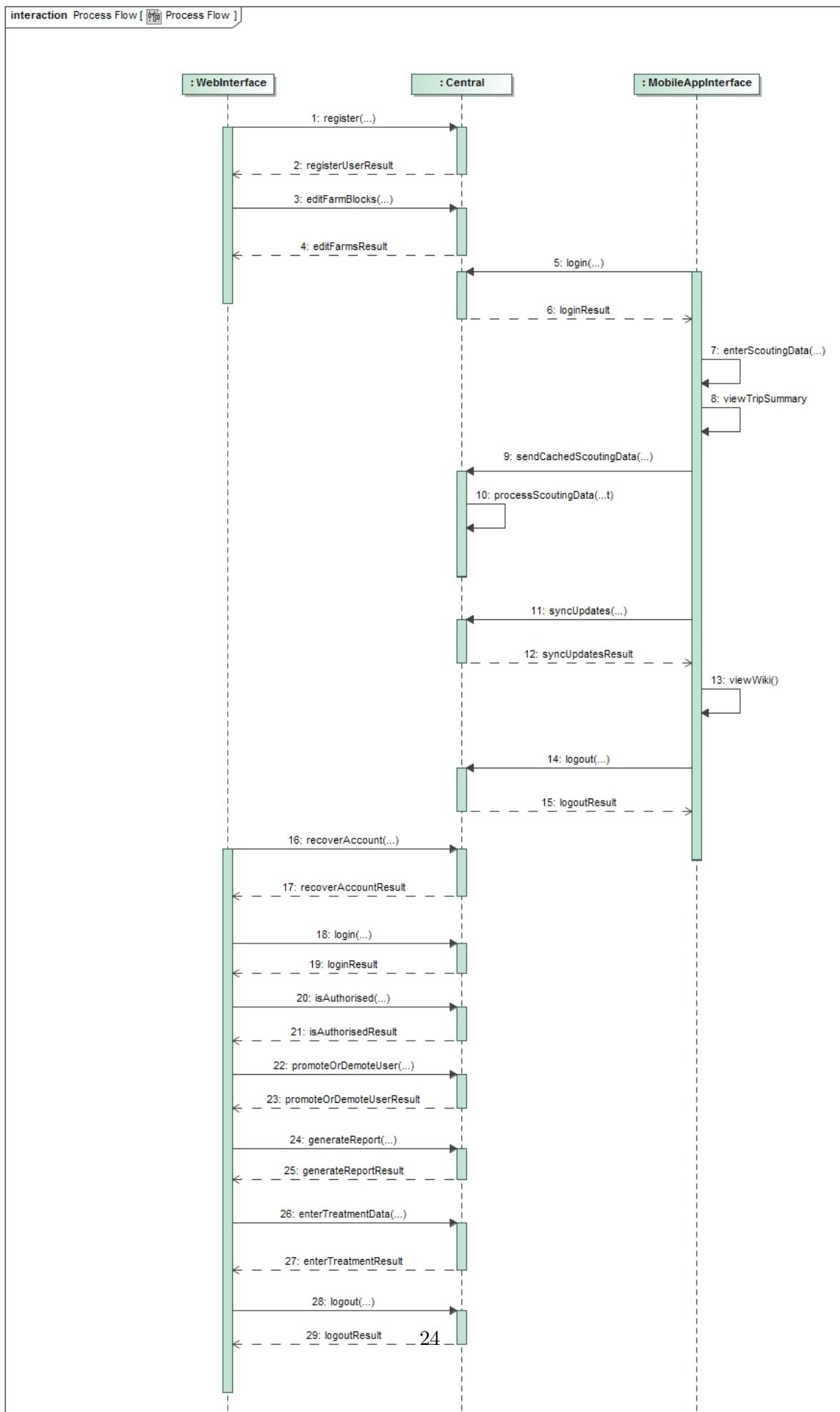
6.2 Infrastructure Layout



6.3 Database Design



6.4 Process Flow



6.5 Technologies

6.5.1 Mobile Application

6.5.1.1 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](#).

6.5.1.2 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 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](#).

6.5.1.3 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](#).

6.5.2 Web Backend

6.5.2.1 .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

6.5.2.2 AutoMapper

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

6.5.2.3 .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.

6.5.2.4 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 swapped out for other free database technologies such as MySQL due to the technology neutral architectural design.

6.5.2.5 AutoFac

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

6.5.3 Web Frontend

6.5.3.1 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.

6.5.3.2 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 library draws its chart elements in the Scalable Vector Graphics format, allowing crisp images, independent of device specifications.

6.5.3.3 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!

6.5.4 Builds and Deployment

6.5.4.1 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.

6.5.4.2 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.

6.5.4.3 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.

6.5.5 Testing

6.5.5.1 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](#).

6.5.5.2 MSTest

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

6.5.6 Artifical Intelligence and Computer Vision

6.5.6.1 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.

6.5.7 Additional Technologies

6.5.7.1 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.)

6.5.7.2 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>.

6.5.7.3 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.

6.5.7.4 Misc. Javascript Frameworks and Libraries

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