Functional Requirements for the SAMBUG Project

COS301

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

Software Developers

May 2015

Contents

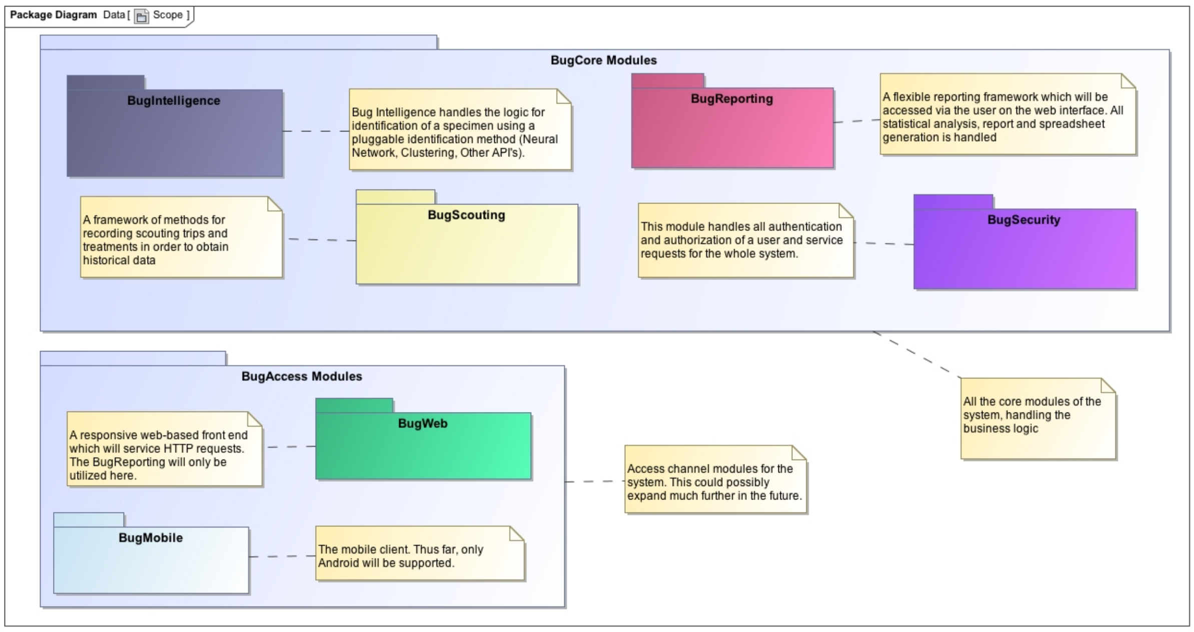
1 Background

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 timing of chemical sprays rely on accurate scout data and economic 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 hemipteran species by making use of the built-in camera of the smartphone. This application should ideally be able to make use of the smartphones built-in GPS to perform geotagging and uploading information to a central database.

3 Scope


Package Diagram Data [Scope]
BugCore Modules
BugIntelligence
Bug Intelligence handles the logic for identification of a specimen using a pluggable identification method (Neural Network, Clustering, Other API's).
BugReporting
A flexible reporting framework which will be accessed via the user on the web interface. All statistical analysis, report and spreadsheet generation is handled
A framework of methods for recording scouting trips and treatments in order to obtain historical data
BugScouting
This module handles all authentication and authorization of a user and service requests for the whole system.
BugSecurity
All the core modules of the system, handling the business logic
BugAccess Modules
A responsive web-based front end which will service HTTP requests. The BugReporting will only be utilized here.
BugWeb
Access channel modules for the system. This could possibly expand much further in the future.
BugMobile
The mobile client. Thus far, only Android will be supported.

4 Functional Requirements and Application Design

4.1 Domain Model

4.2 BugScouting

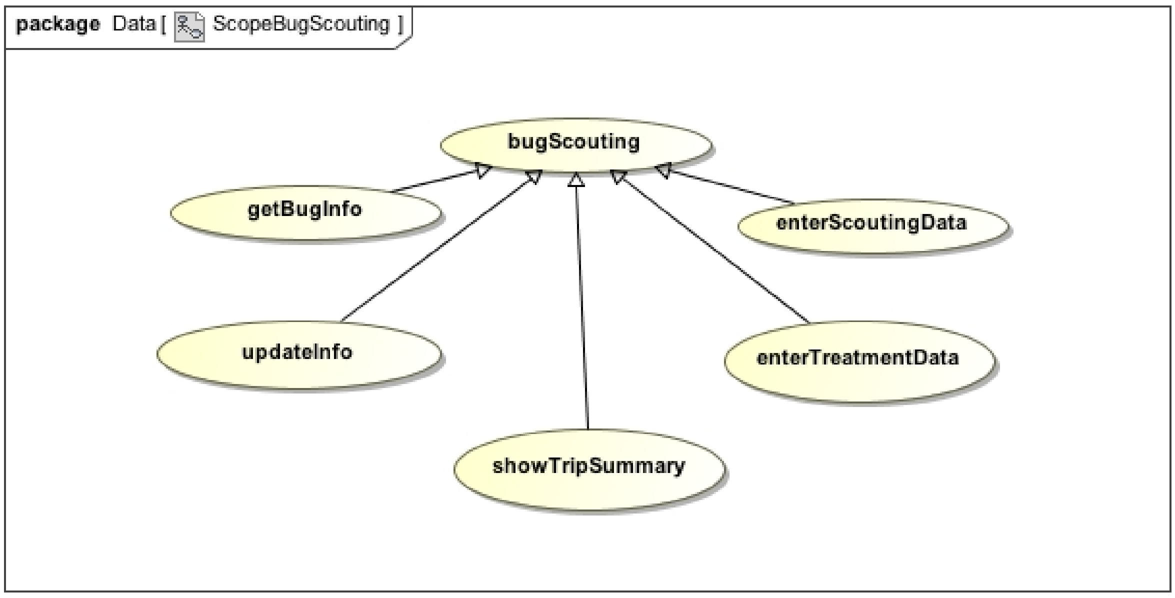
The BugScouting module has the following functionality:

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 average number of bugs per tree and the different kind of bugs specified. After which a summary should be displayed.

2. It provides the ability for a user to record the details related to spraying.

4.2.1 Module Scope

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



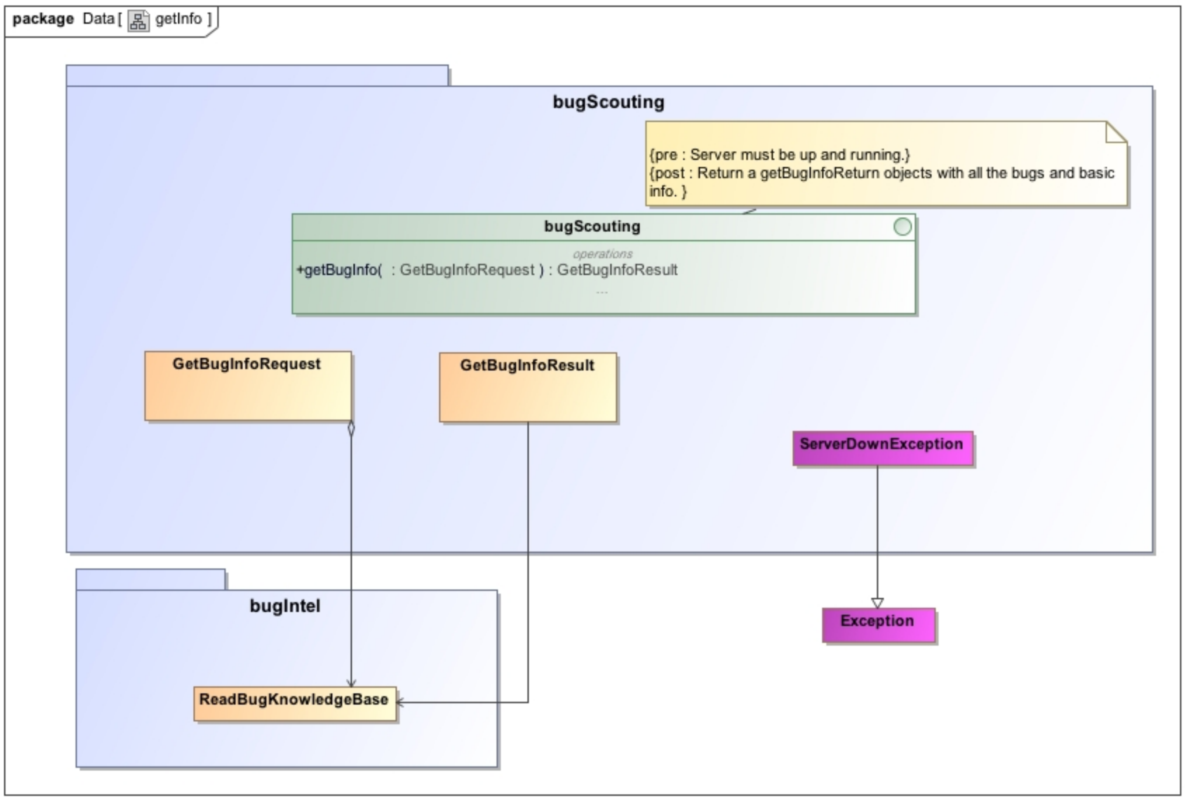
4.2.2 Use Cases

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.

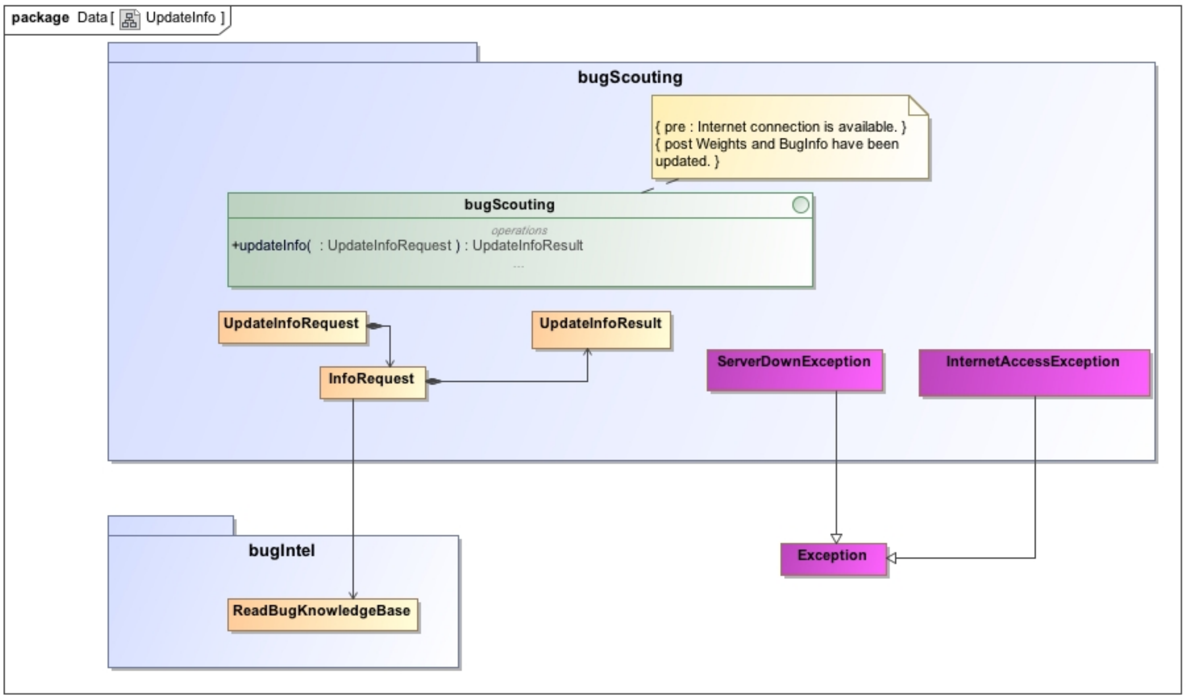
The Service Contract for the *getBugInfo* use case is shown below:


package Data [getInfo]
bugScouting
{pre : Server must be up and running.}
{post : Return a getBugInfoReturn objects with all the bugs and basic info. }
bugScouting
operations
+getBugInfo(: GetBugInfoRequest) : GetBugInfoResult
...
GetBugInfoRequest
GetBugInfoResult
ServerDownException
bugIntel
ReadBugKnowledgeBase
Exception

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.

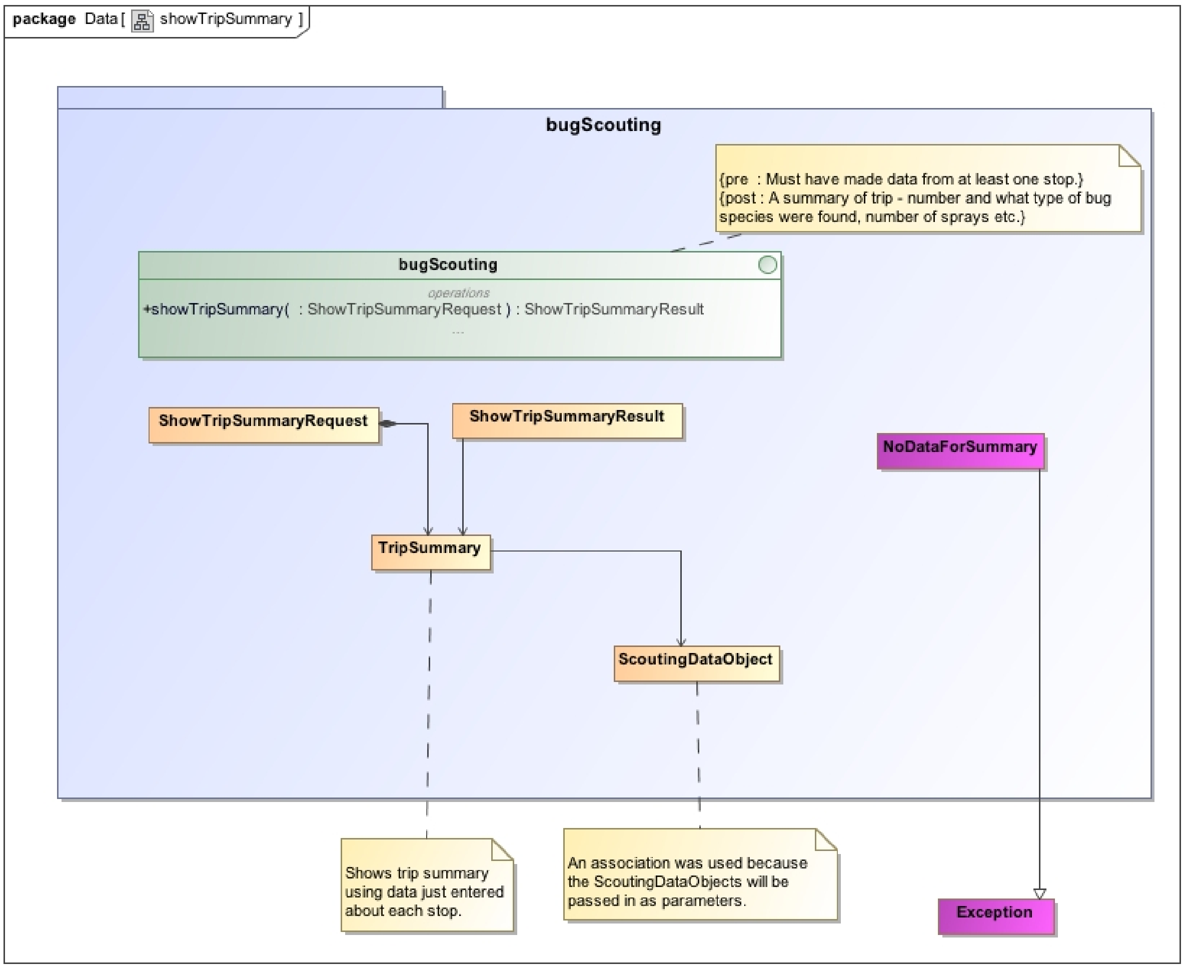
The Service Contract for the *getBugInfo* use case is shown below:


package Data [UpdateInfo]
bugScouting
{ pre : Internet connection is available. }
{ post Weights and BugInfo have been updated. }
bugScouting
operations
+updateInfo(: UpdateInfoRequest) : UpdateInfoResult
...
UpdateInfoRequest
UpdateInfoResult
InfoRequest
ServerDownException
InternetAccessException
bugIntel
ReadBugKnowledgeBase
Exception

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. The averages for the scouting trip data is used in the summary.

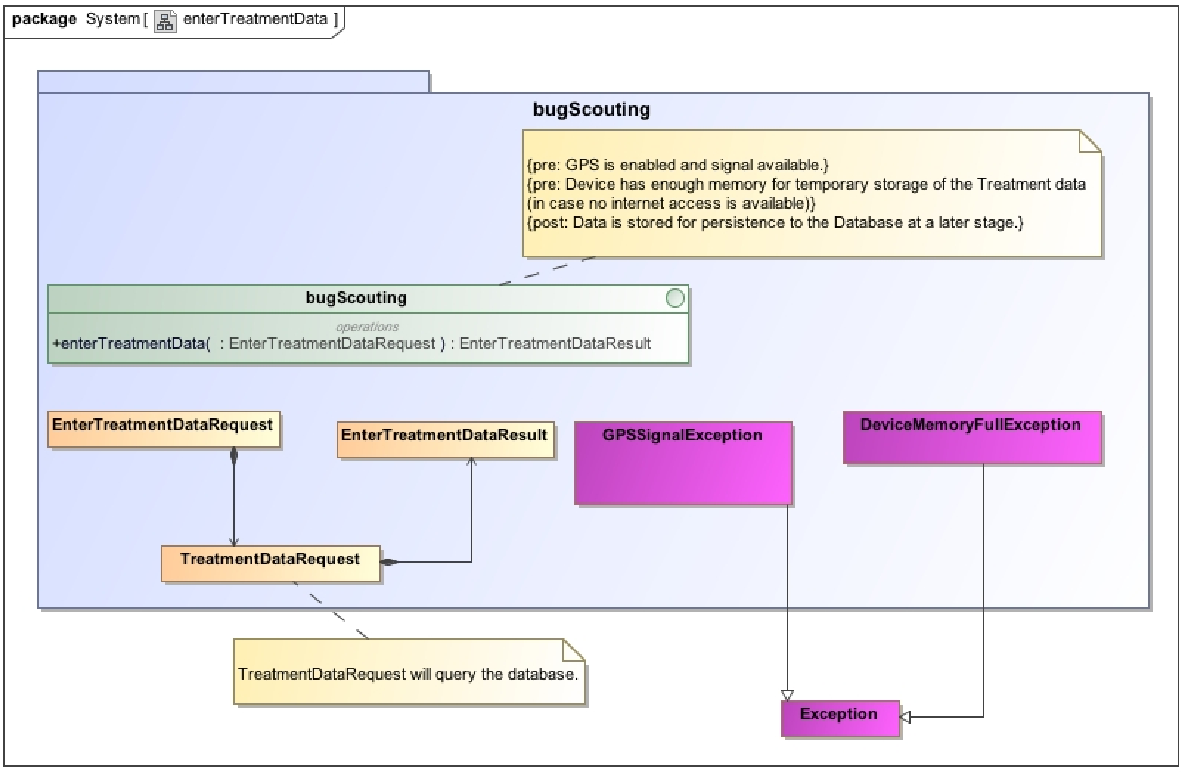
The Service Contract for the *showTripSummary* use case is shown below:


package Data [showTripSummary]
bugScouting
{pre : Must have made data from at least one stop.}
{post : A summary of trip - number and what type of bug species were found, number of sprays etc.}
bugScouting
operations
+showTripSummary(: ShowTripSummaryRequest) : ShowTripSummaryResult
...
ShowTripSummaryRequest
ShowTripSummaryResult
NoDataForSummary
TripSummary
ScoutingDataObject
Shows trip summary using data just entered about each stop.
An association was used because the ScoutingDataObjects will be passed in as parameters.
Exception

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.

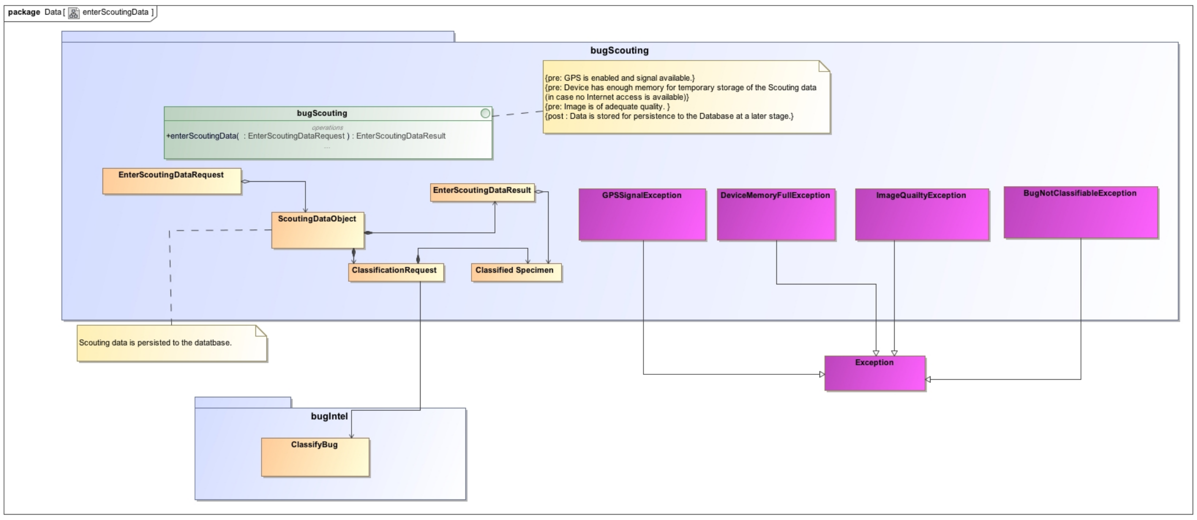
The Service Contract for the *enterTreatmentData* use case is shown below:


package System [enterTreatmentData]
bugScouting
{pre: GPS is enabled and signal available.}
{pre: Device has enough memory for temporary storage of the Treatment data
(in case no internet access is available)}
{post: Data is stored for persistence to the Database at a later stage.}
bugScouting
operations
+enterTreatmentData(: EnterTreatmentDataRequest) : EnterTreatmentDataResult
EnterTreatmentDataRequest
EnterTreatmentDataResult
GPSSignalException
DeviceMemoryFullException
TreatmentDataRequest
TreatmentDataRequest will query the database.
Exception

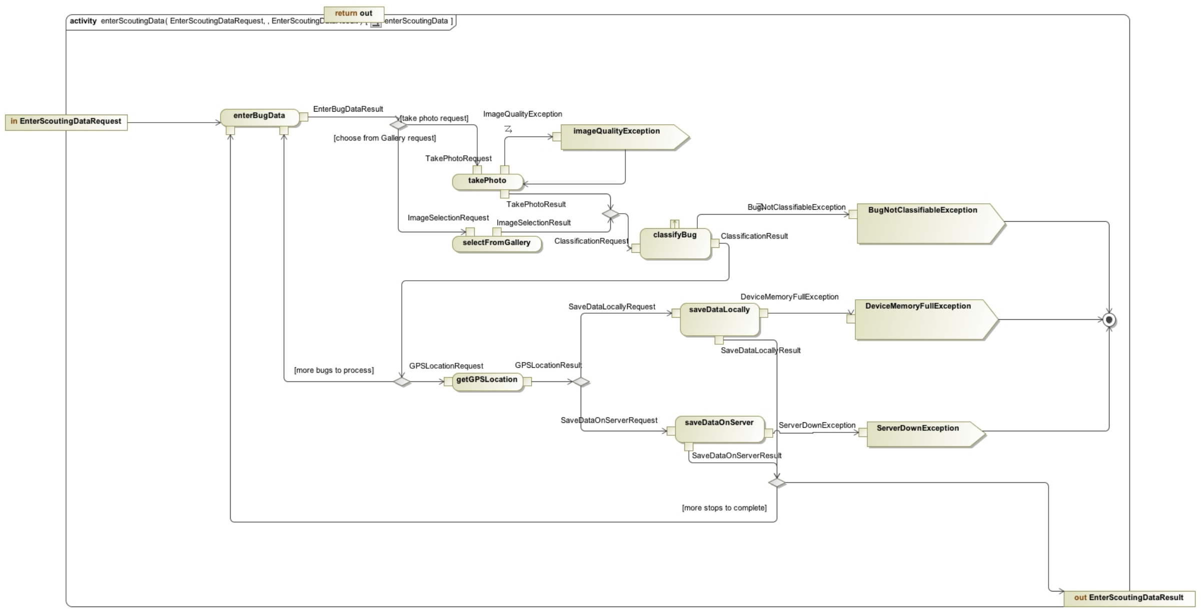
4.2.2.5 enterScoutingData . . . . [Priority - Critical]

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, number of bugs counter and the block or orchard number where the scouting took place. After entering the data, the specimen should be identified (classified) before successfully submitting it for persistence.

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


package Data [enterScoutingData]
bugScouting
{pre: GPS is enabled and signal available.}
{pre: Device has enough memory for temporary storage of the Scouting data
(in case no Internet access is available)}
{pre: Image is of adequate quality. }
{post : Data is stored for persistence to the Database at a later stage.}
bugScouting
operations
+enterScoutingData(: EnterScoutingDataRequest) : EnterScoutingDataResult
EnterScoutingDataRequest
EnterScoutingDataResult
ScoutingDataObject
ClassificationRequest
Classified Specimen
GPSSignalException
DeviceMemoryFullException
ImageQuailtyException
BugNotClassifiableException
Scouting data is persisted to the datatbase.
Exception
bugIntel
ClassifyBug

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



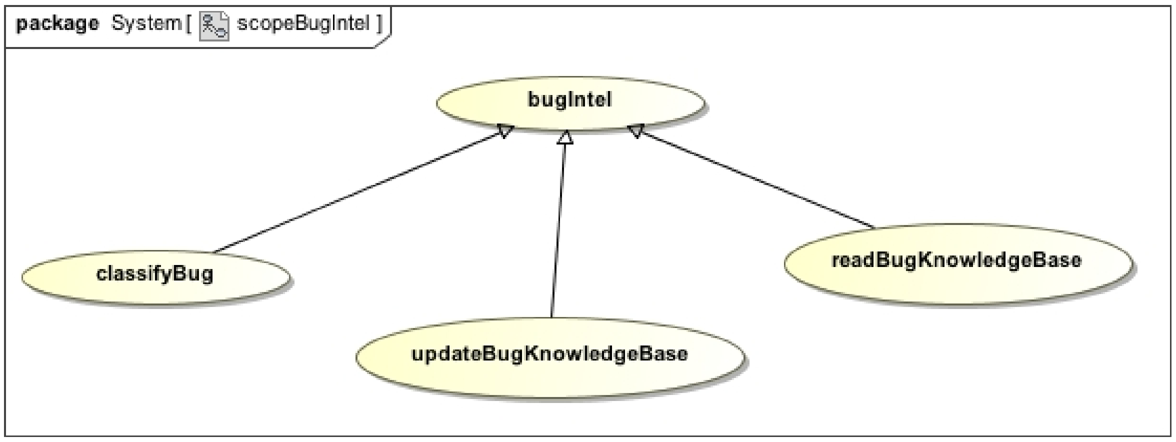
4.3 BugIntelligence

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:



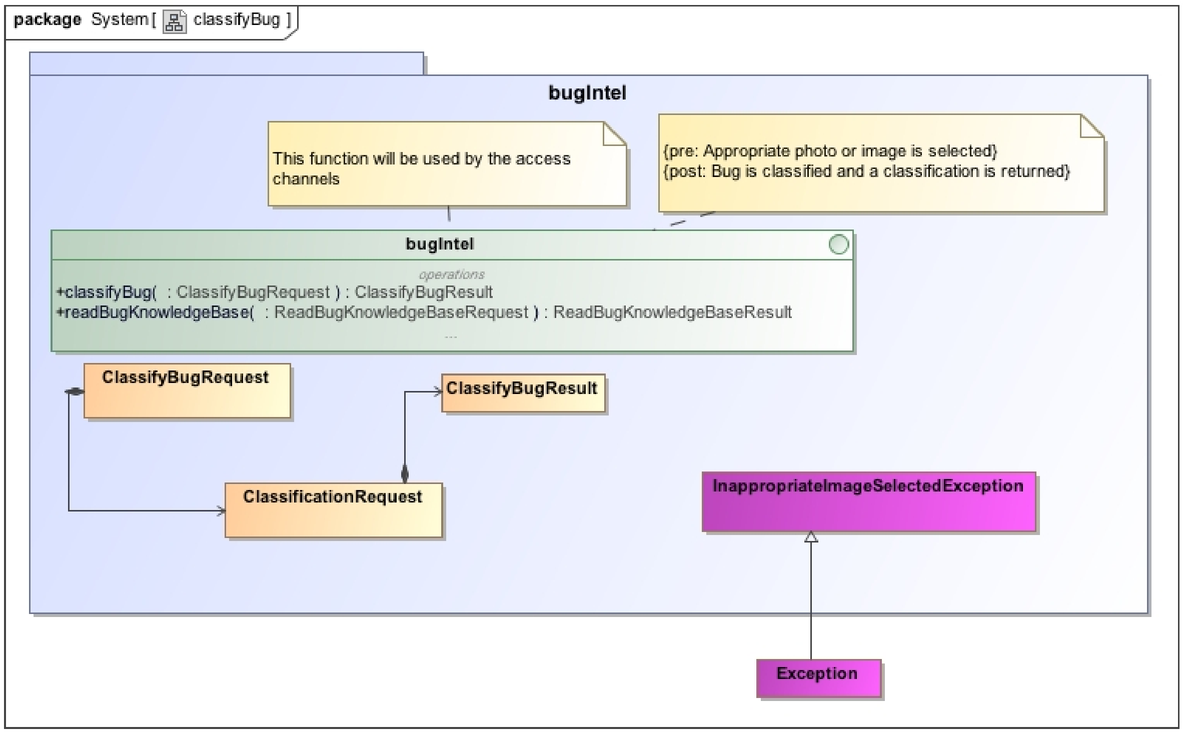
4.3.2 Use Cases

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

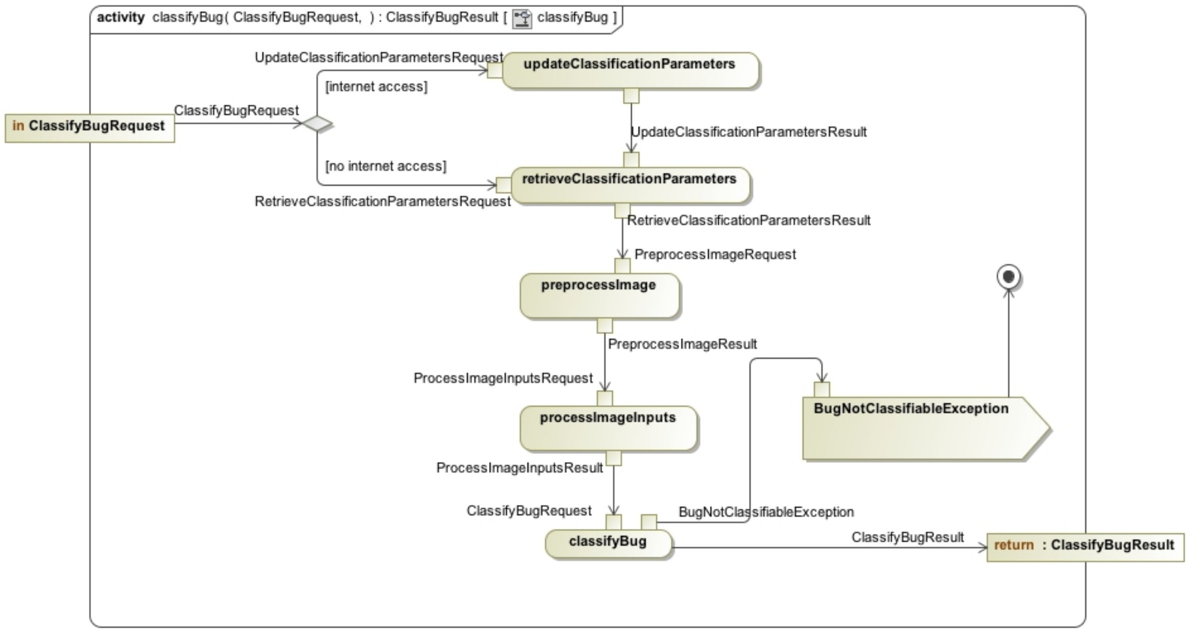
4.3.2.1 classifyBug .... [Priority - Critical]

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:

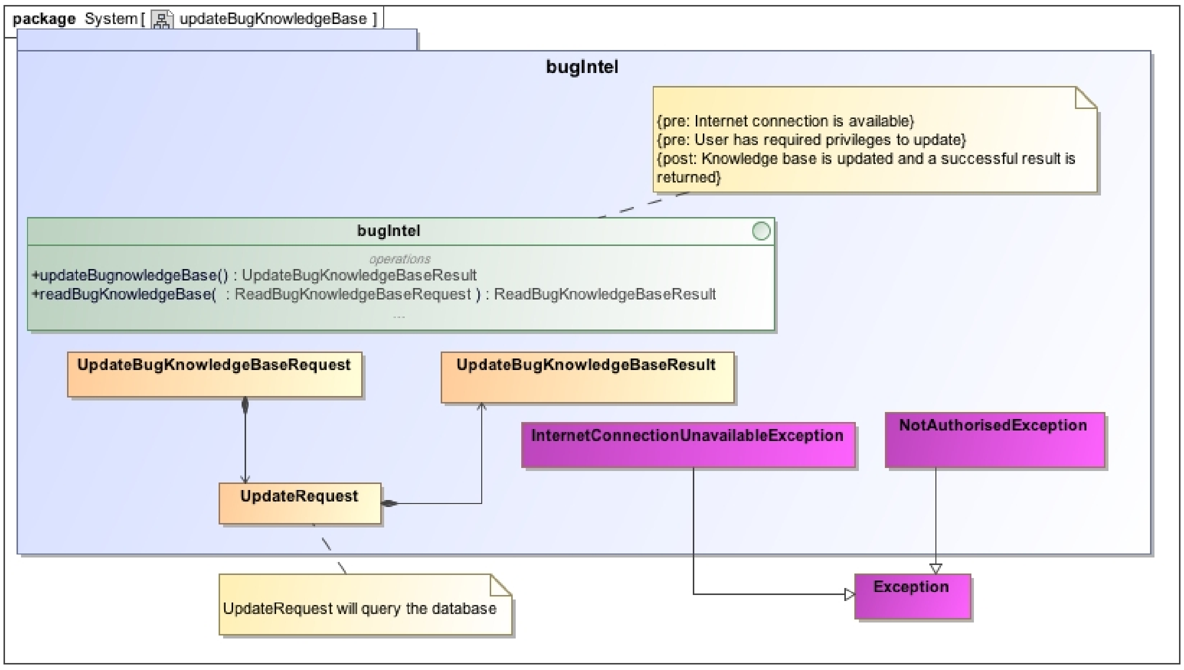

package System [classifyBug]
bugIntel
This function will be used by the access channels
{pre: Appropriate photo or image is selected}
{post: Bug is classified and a classification is returned}
bugIntel
operations
+classifyBug(: ClassifyBugRequest) : ClassifyBugResult
+readBugKnowledgeBase(: ReadBugKnowledgeBaseRequest) : ReadBugKnowledgeBaseResult
ClassifyBugRequest
ClassifyBugResult
ClassificationRequest
InappropriateImageSelectedException
Exception

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


activity classifyBug(ClassifyBugRequest,) : ClassifyBugResult [classifyBug]
in ClassifyBugRequest
ClassifyBugRequest
[internet access]
[no internet access]
UpdateClassificationParametersRequest
updateClassificationParameters
UpdateClassificationParametersResult
RetrieveClassificationParametersRequest
retrieveClassificationParameters
RetrieveClassificationParametersResult
PreprocessImageRequest
preprocessImage
PreprocessImageResult
ProcessImageInputsRequest
processImageInputs
ProcessImageInputsResult
ClassifyBugRequest
classifyBug
BugNotClassifiableException
BugNotClassifiableException
ClassifyBugResult
return : ClassifyBugResult

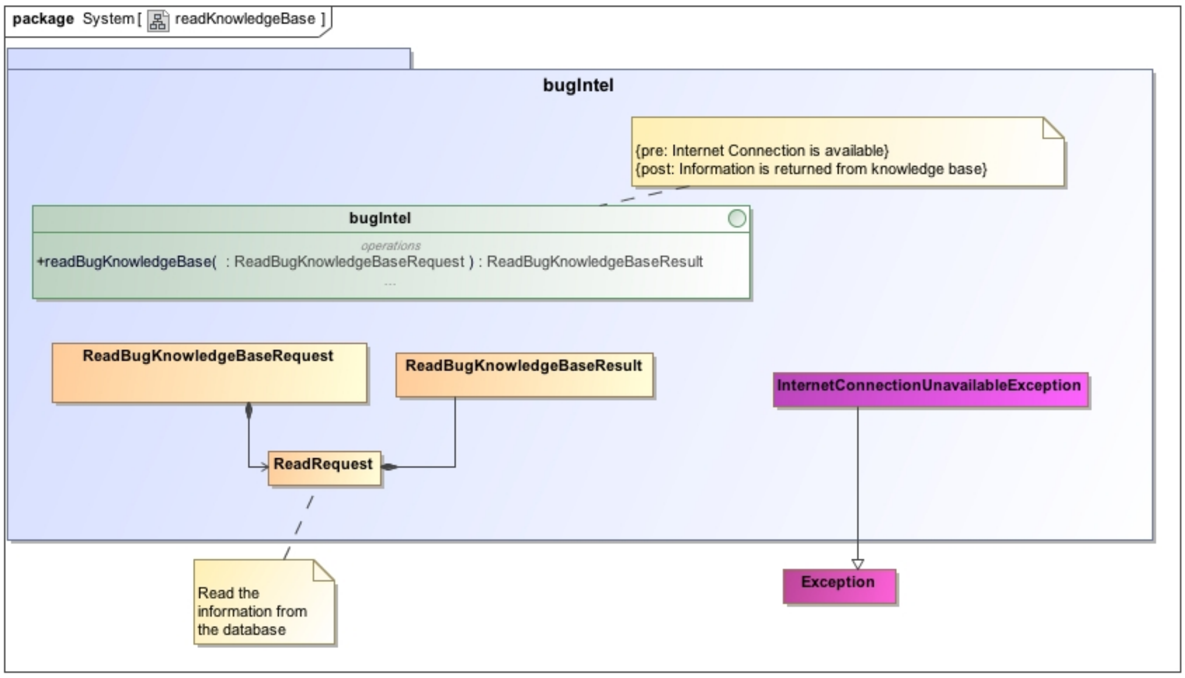
4.3.2.2 updateBugKnowledgeBase .... [Priority - Low]

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:


package System [updateBugKnowledgeBase]
bugIntel
{pre: Internet connection is available}
{pre: User has required privileges to update}
{post: Knowledge base is updated and a successful result is returned}
bugIntel
operations
+updateBugnowledgeBase() : UpdateBugKnowledgeBaseResult
+readBugKnowledgeBase(: ReadBugKnowledgeBaseRequest) : ReadBugKnowledgeBaseResult
...
UpdateBugKnowledgeBaseRequest
UpdateBugKnowledgeBaseResult
InternetConnectionUnavailableException
NotAuthorisedException
UpdateRequest
UpdateRequest will query the database
Exception

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:


package System [readKnowledgeBase]
bugIntel
{pre: Internet Connection is available}
{post: Information is returned from knowledge base}
bugIntel
operations
+readBugKnowledgeBase(: ReadBugKnowledgeBaseRequest) : ReadBugKnowledgeBaseResult
...
ReadBugKnowledgeBaseRequest
ReadBugKnowledgeBaseResult
InternetConnectionUnavailableException
ReadRequest
Read the information from the database
Exception

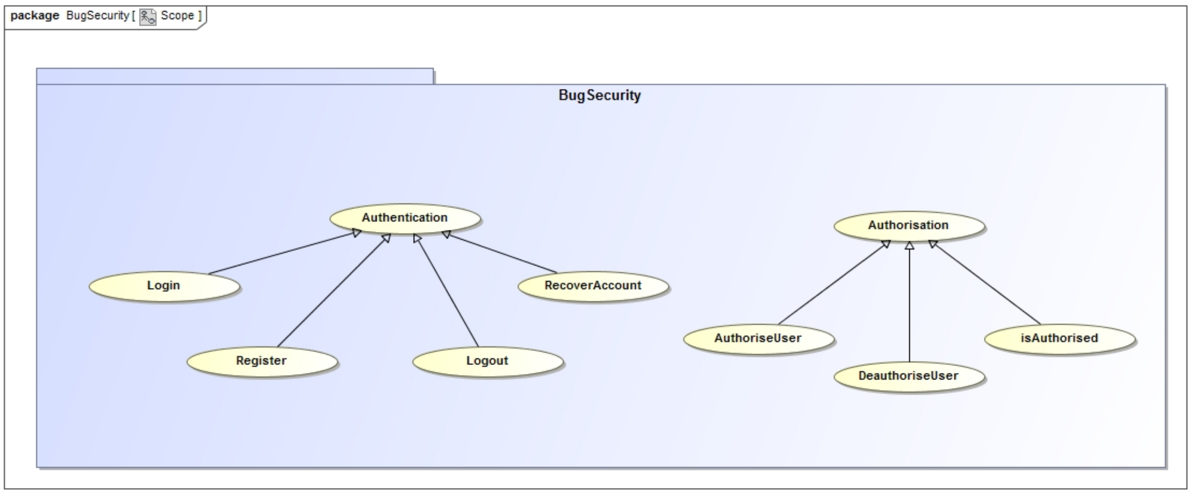
4.4 BugSecurity

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:



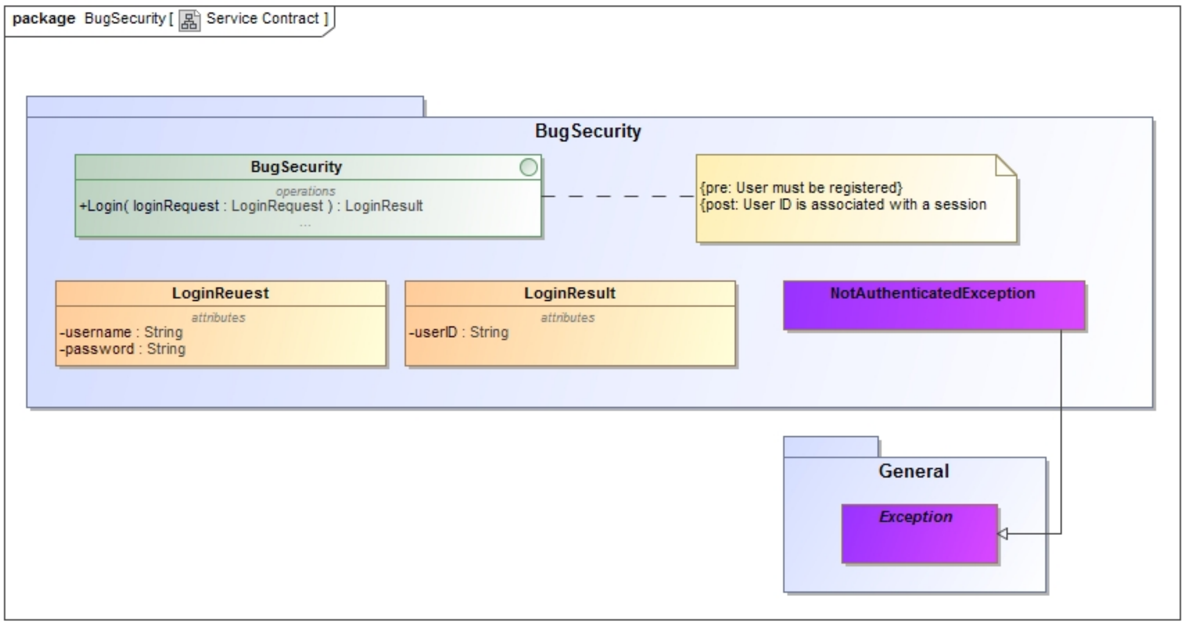
4.4.2 Use Cases

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

4.4.2.1 login .... [Priority - Critical]

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

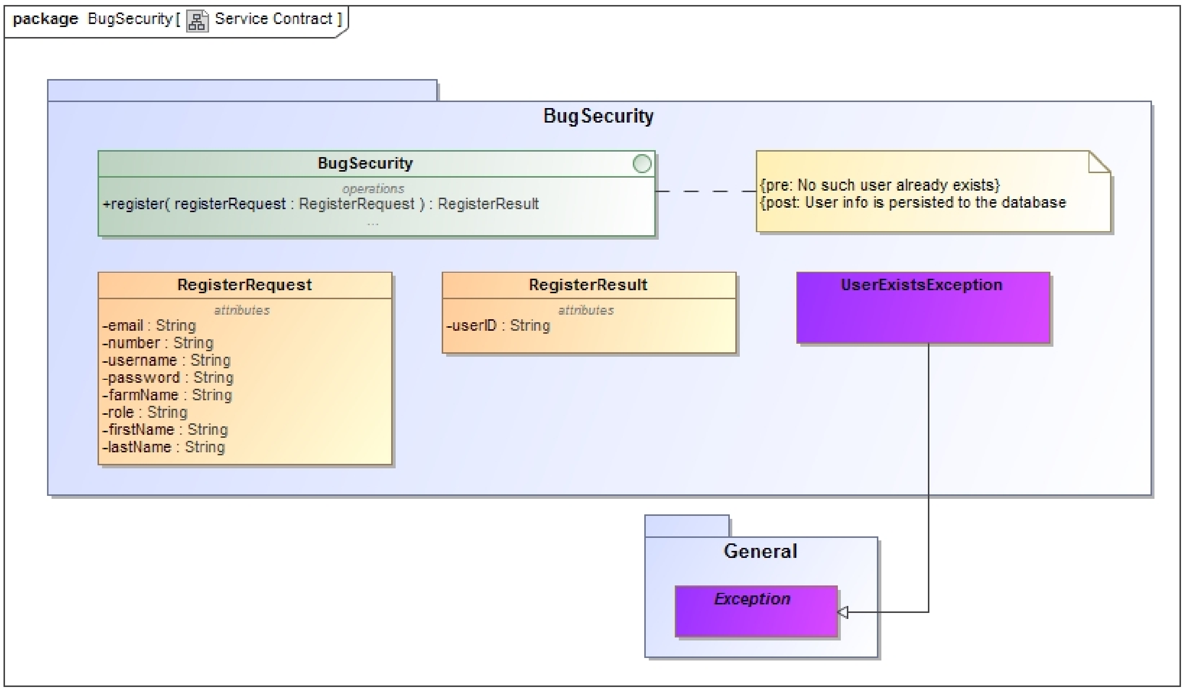
The Service Contract for the *login* use case is shown below:


package BugSecurity [Service Contract]
BugSecurity
BugSecurity
operations
+Login(loginRequest : LoginRequest) : LoginResult
{pre: User must be registered}
{post: User ID is associated with a session
LoginReuest
attributes
-username : String
-password : String
LoginResult
attributes
-userID : String
NotAuthenticatedException
General
Exception

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.

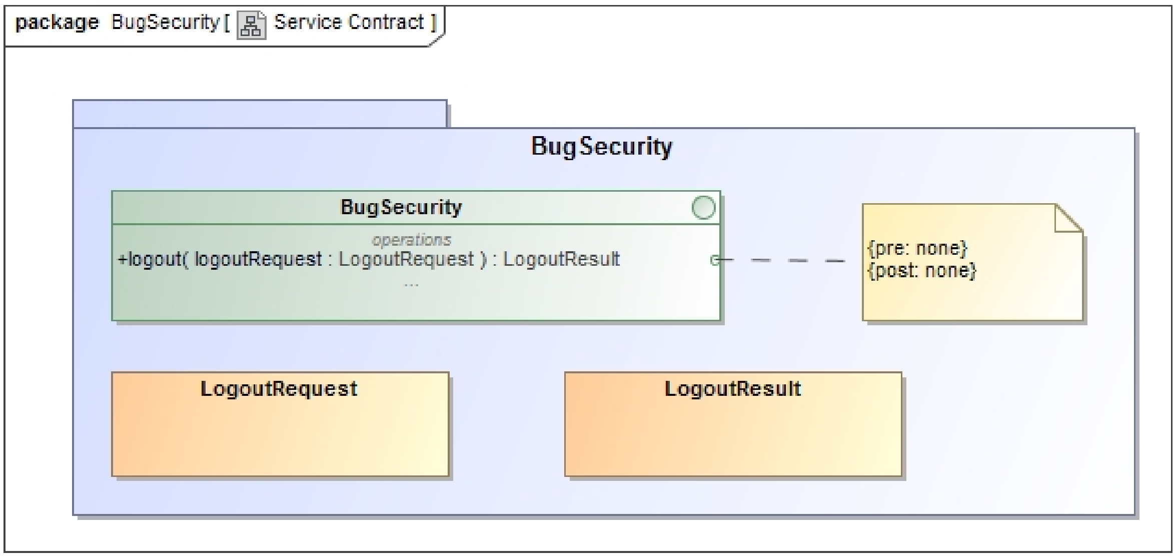
The Service Contract for the *register* use case is shown below:



4.4.2.3 logout . . . . [Priority - Medium]

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

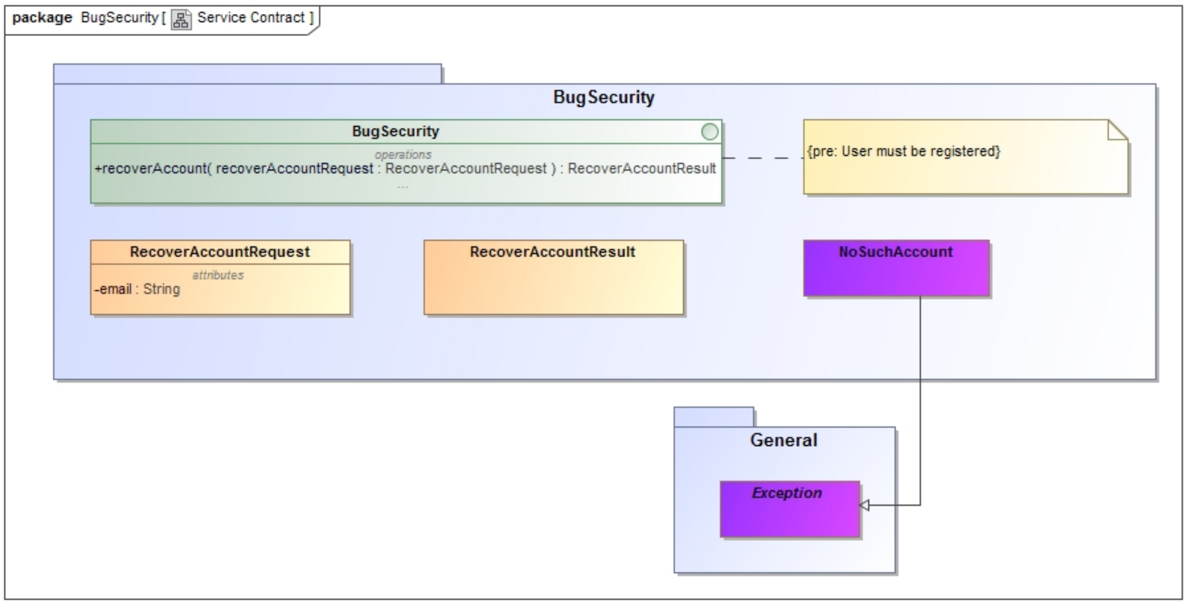
The Service Contract for the *logout* use case is shown below:


package BugSecurity [Service Contract]
BugSecurity
BugSecurity
operations
+logout(logoutRequest : LogoutRequest) : LogoutResult
...
{pre: none}
{post: none}
LogoutRequest
LogoutResult

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:


package BugSecurity [Service Contract]
BugSecurity
BugSecurity
operations
+recoverAccount(recoverAccountRequest : RecoverAccountRequest) : RecoverAccountResult
...
{pre: User must be registered}
RecoverAccountRequest
attributes
-email : String
RecoverAccountResult
NoSuchAccount
General
Exception

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:

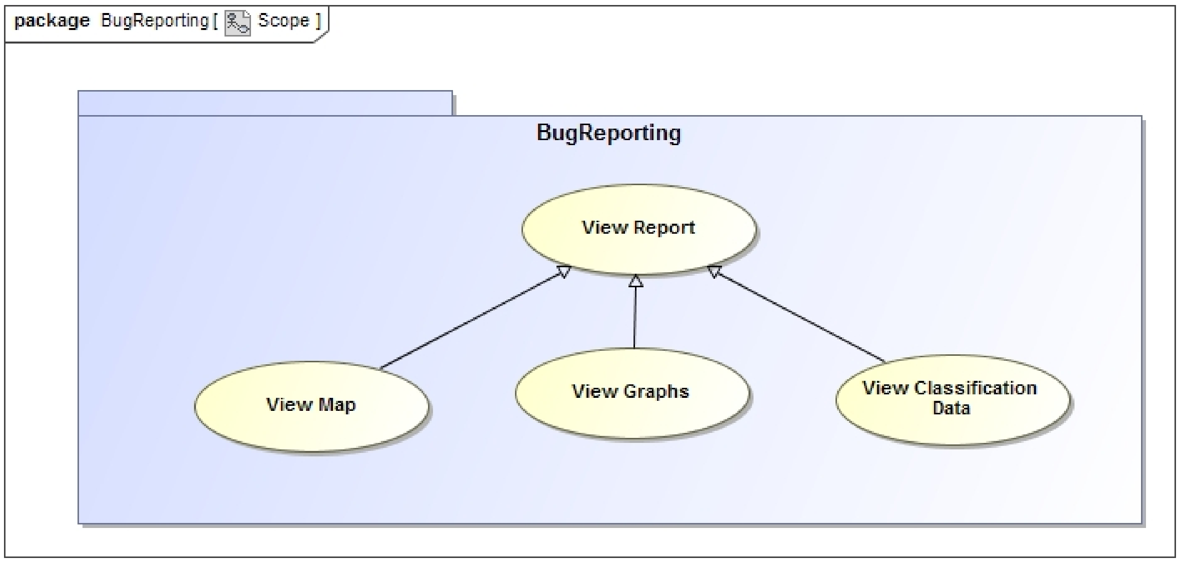
4.5 BugReporting

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:



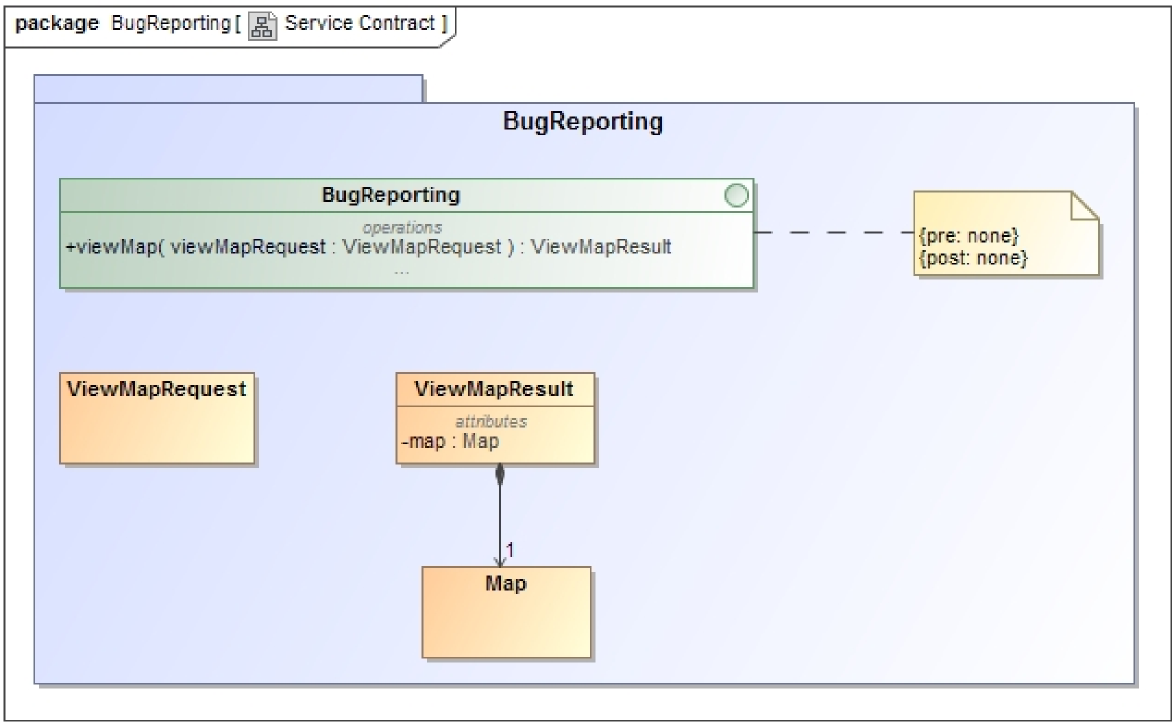
4.5.2 Use Cases

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

4.5.2.1 viewMap .... [Priority - Medium]

This use case allows for the generation of a heat map based on historical data.

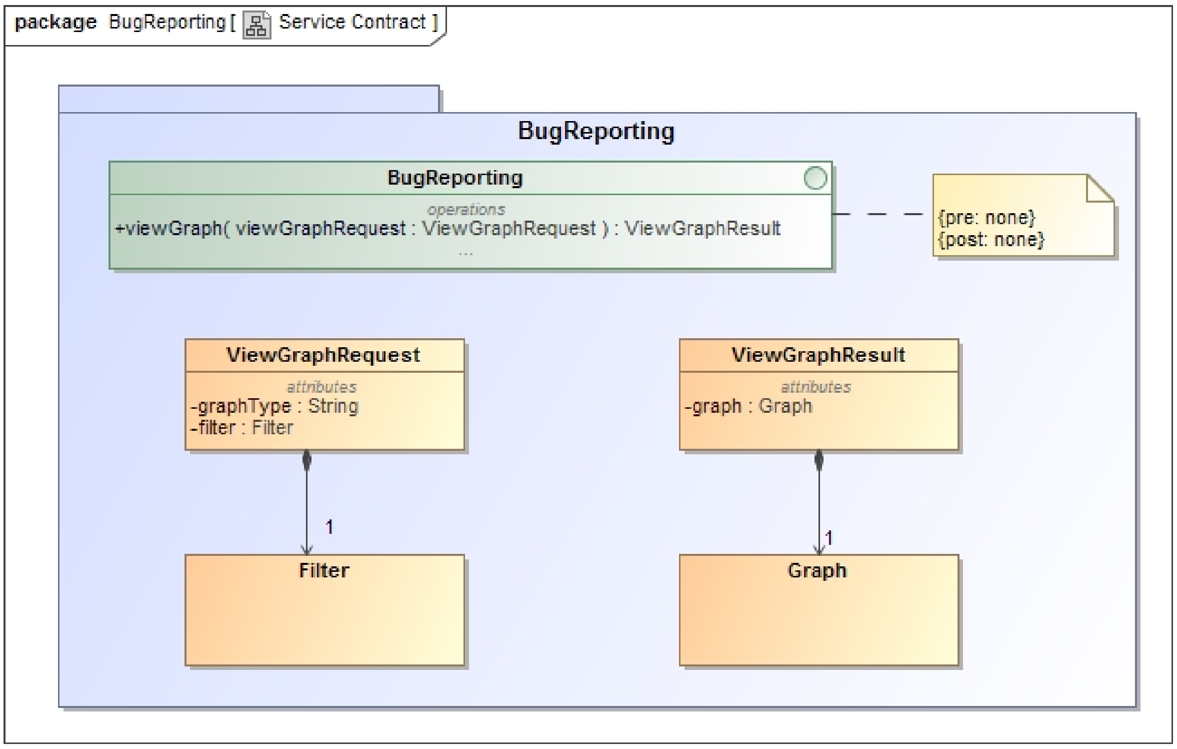
The Service Contract for the *viewMap* use case is shown below:



4.5.2.2 viewGraph .... [Priority - Medium]

This use case allows for the generation of a graphs based on historical data.

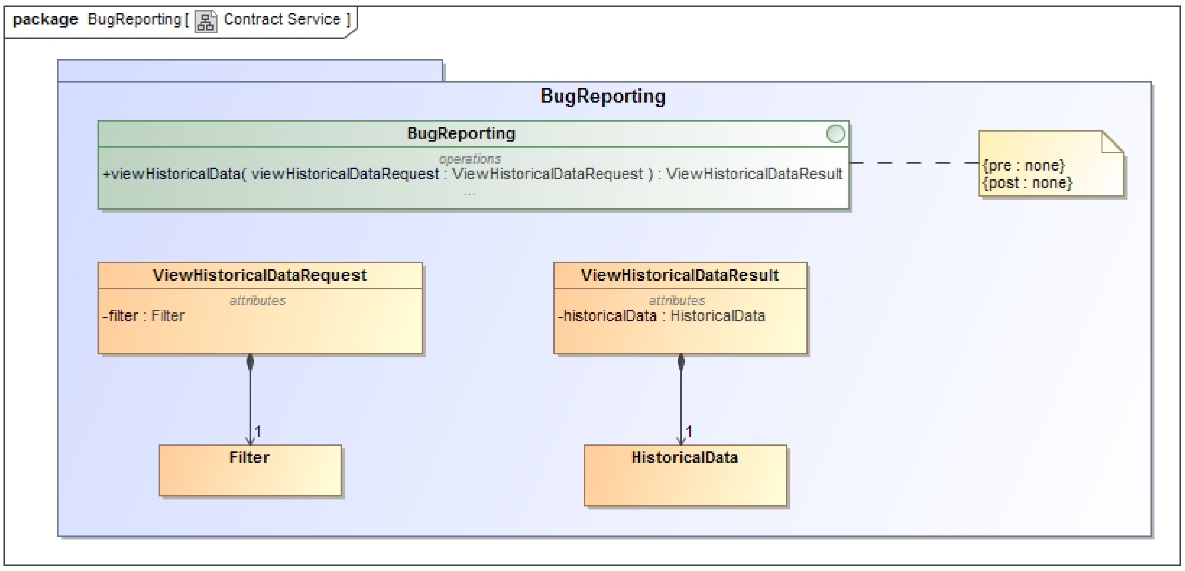
The Service Contract for the *viewGraph* use case is shown below:



4.5.2.3 viewHistoricalData . . . . [Priority - Medium]

This use case allows for the generation of historical data.

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


package BugReporting [Contract Service]
BugReporting
BugReporting
operations
+viewHistoricalData(viewHistoricalDataRequest : ViewHistoricalDataRequest) : ViewHistoricalDataResult
...
{pre : none}
{post : none}
ViewHistoricalDataRequest
attributes
-filter : Filter
ViewHistoricalDataResult
attributes
-historicalData : HistoricalData
1
1
Filter
HistoricalData

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 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

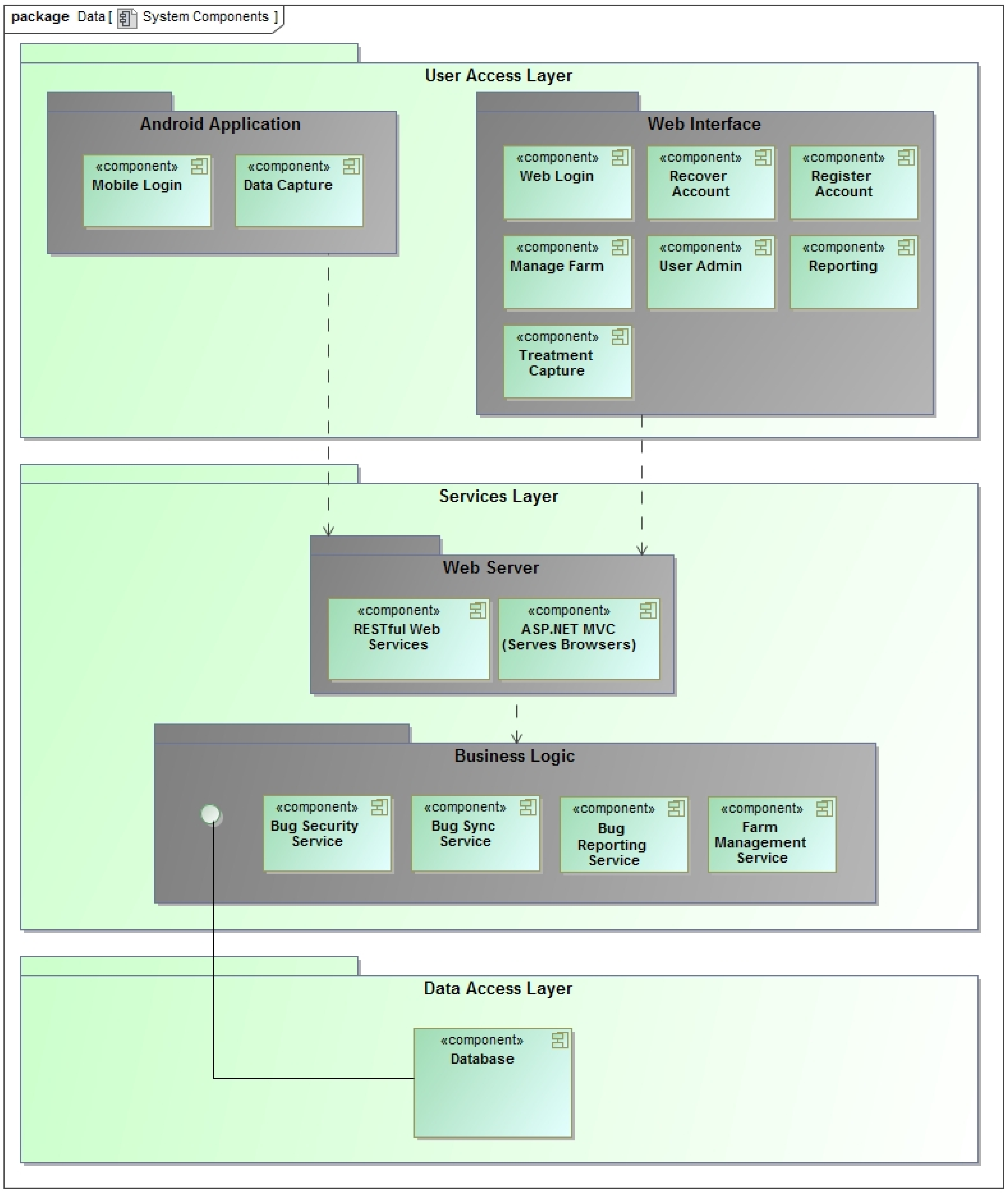
6.1 Overview

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:

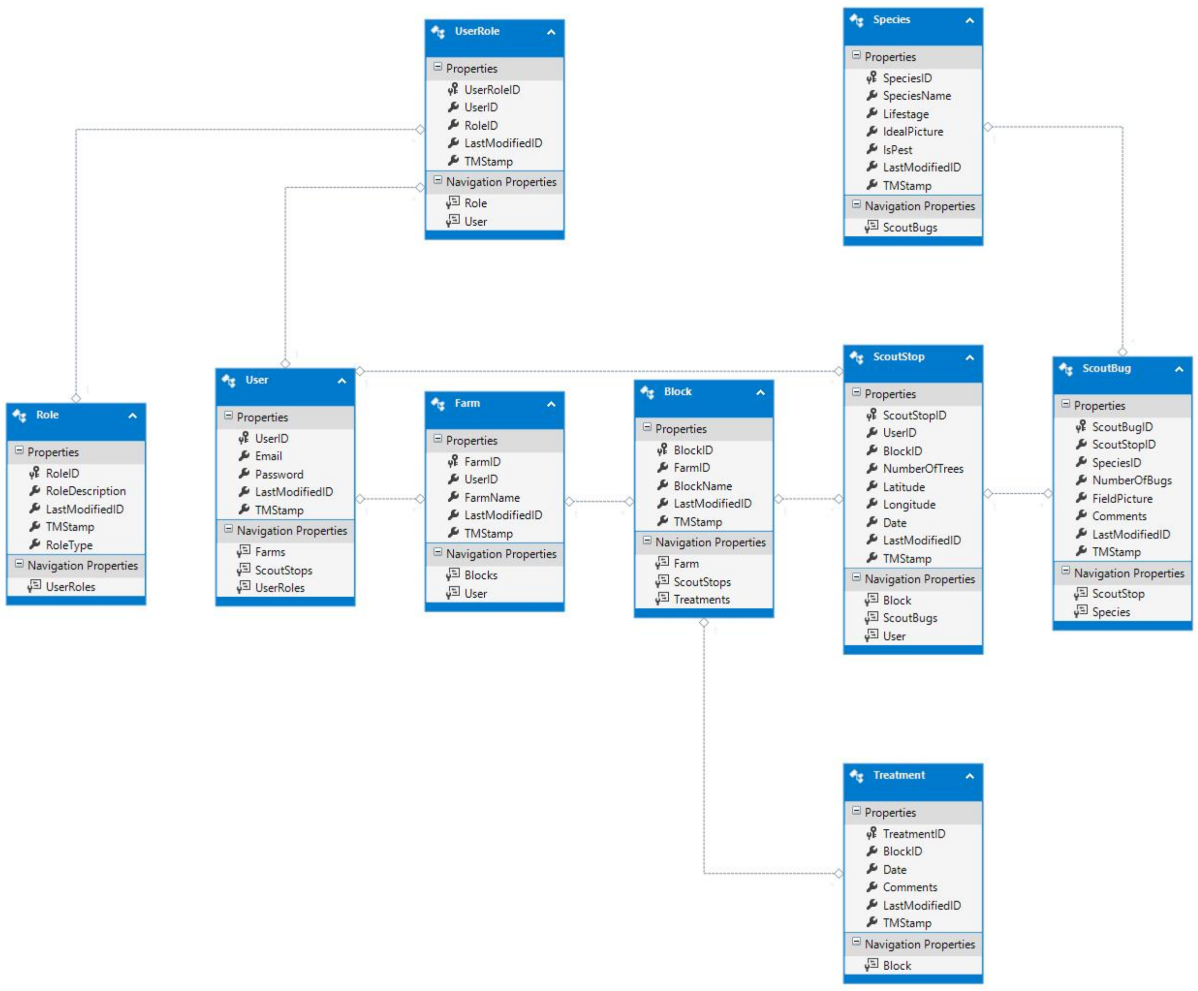
- Access Layer
  - View
    In many ways the Android application serves as an access channel to the Web server. However, it is not categorised as a "View" in the MVC sense. Our MVC pattern is only used across the Access and Services layers. Therefore, this only encompasses HTML views that are rendered 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.

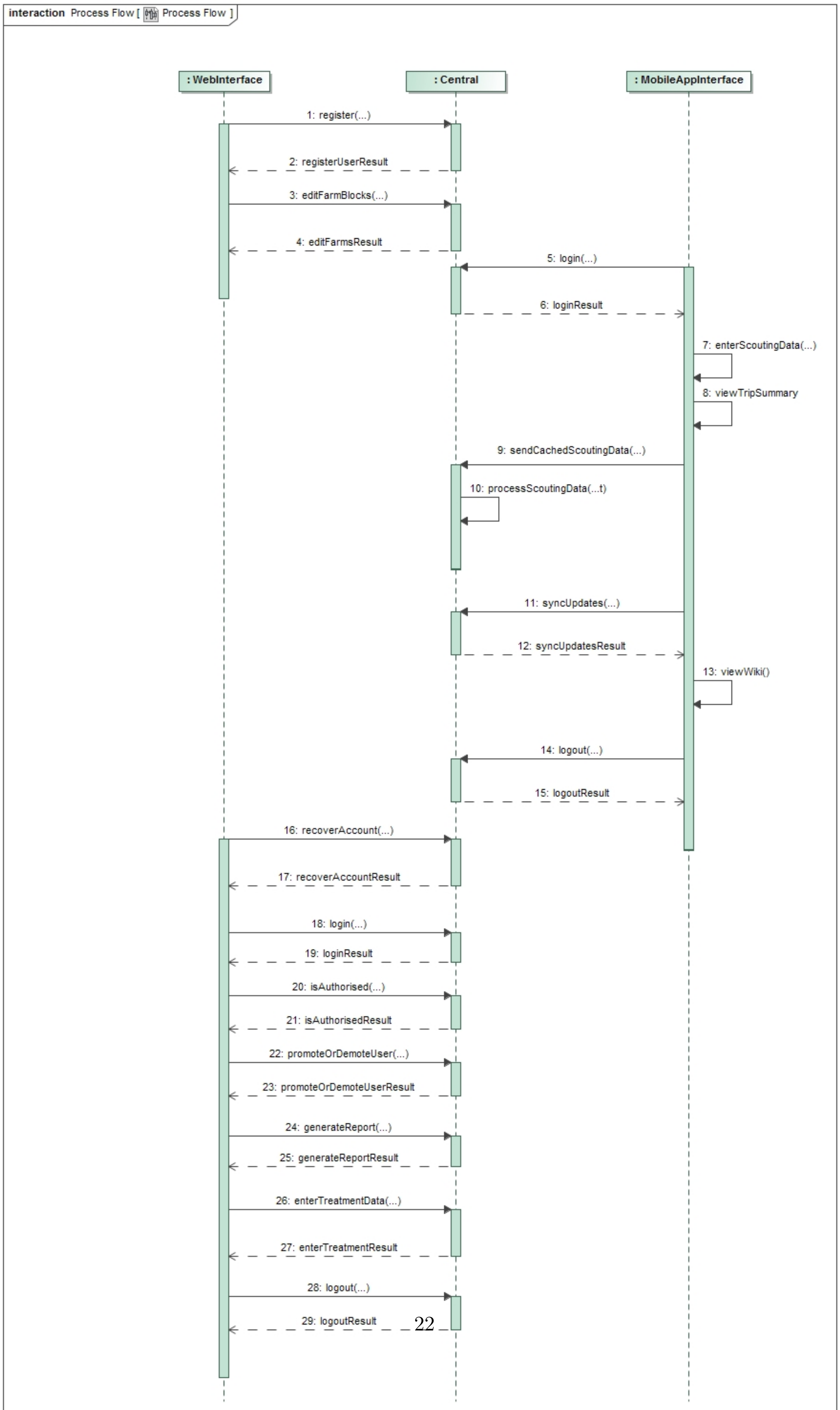
6.2 Infrastructure Layout


package Data [System Components]
User Access Layer
Android Application
«component» Mobile Login
«component» Data Capture
Web Interface
«component» Web Login
«component» Recover Account
«component» Register Account
«component» Manage Farm
«component» User Admin
«component» Reporting
«component» Treatment Capture
Services Layer
Web Server
«component» RESTful Web Services
«component» ASP.NET MVC (Serves Browsers)
Business Logic
«component» Bug Security Service
«component» Bug Sync Service
«component» Bug Reporting Service
«component» Farm Management Service
Data Access Layer
«component» Database

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 since 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.

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 are old, unpleasant and difficult to find documentation for.
- Mockito not only allows interfaces to mocked, but also allows abstract and concrete classes to mocked. Furthermore, this can be done without any extensions (e.g. EasyMock requires class extentions.)
- Since it is a single Jar, it is easy to setup.
- Mockito supports spies (partial mocks), in order to gather indirect output from a test, meaning functions can be tested and verified without giving the test input.

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 it 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.

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 .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.

6.5.2.3 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.

6.5.2.4 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.

6.5.3 Web Frontend

6.5.3.1 Aurelia.io

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

6.5.4 Builds

6.5.4.1 TeamCity

TeamCity was chosen as a automated build and continious integration environment being run on a development server to fascilitate continious integration and better development as a team.

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.

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.