Week 12 Report

The most recent version of this report can be found here.

Jessica Braddon-Parsons	13219524
Cameron Dykstra	09091068
Umesh Ravji	03338908
James Rumbal	12231075

Contents

1. Introduction **2**

2. Design **3**

2.0.1. Component Diagram 3

2.1. Server 4

2.1.1. Database Diagram 4

2.1.2. Server API 4

2.2. Mobile Application 5

2.2.1. Class Diagram 5

3. Quality Assurance Reports **6**

3.1. Mobile Application Tests 6

3.2. Website and Server Tests 6

3.3. Field Testing 7

3.3.1. Results 7

3.4. User Testing 8

4. Project Statistics **10**

4.1. Server 10

4.1.1. Repository Activity 10

4.1.2. Issue Tracking 11

4.1.3. Coverage 11

4.2. Mobile Application 12

4.2.1. Repository Activity 12

4.2.2. Issue Tracking 12

4.2.3. Jacoco 13

4.2.4. PMD 13

5. Installation **14**

6. Building the Product **15**

6.1. Server and Website 15

6.2. Mobile Application 15

7. Limitations **16**

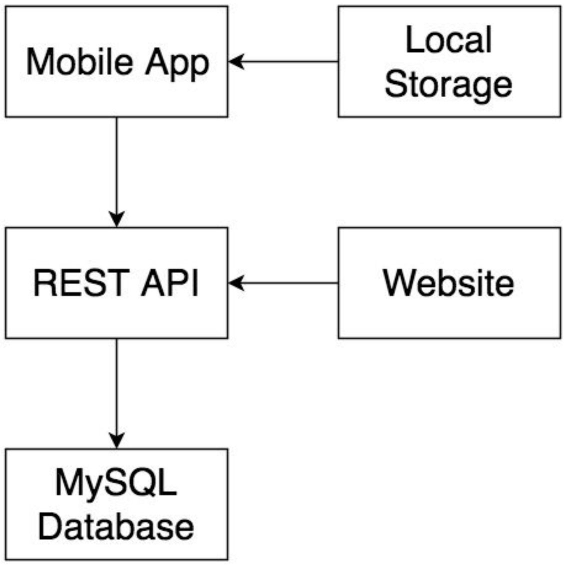
8. Commercialisation Plan **17**

1. Introduction

At the beginning of semester we were presented with the issue of pest trap tracking in areas such as the Manawatu Gorge. Traps in these areas can be difficult to find, and records made on paper can can be difficult to compile. People who volunteer to check the pest traps would benefit from a cross-platform mobile application showing them where the traps are located, and aid in collecting the data they record about the catches. We have created such an application, along with a corresponding server to store the data, and website to make it accessible. The application fetches the list of lines from the server, allows the user to select a line and their start and end positions, then shows them their location and each trap's location along with data entry to record catches. This makes it easier to find the traps, and once a user has finished a line the data is sent to the server so all data is compiled in one place. Once sent to the server, users can view the catches on the website, and download the data as a CSV file. If they have administrator access for the line they can view trap locations and perform maintenance on the line data, and set the most common animals caught on the line to make catch entry easier in the mobile application.

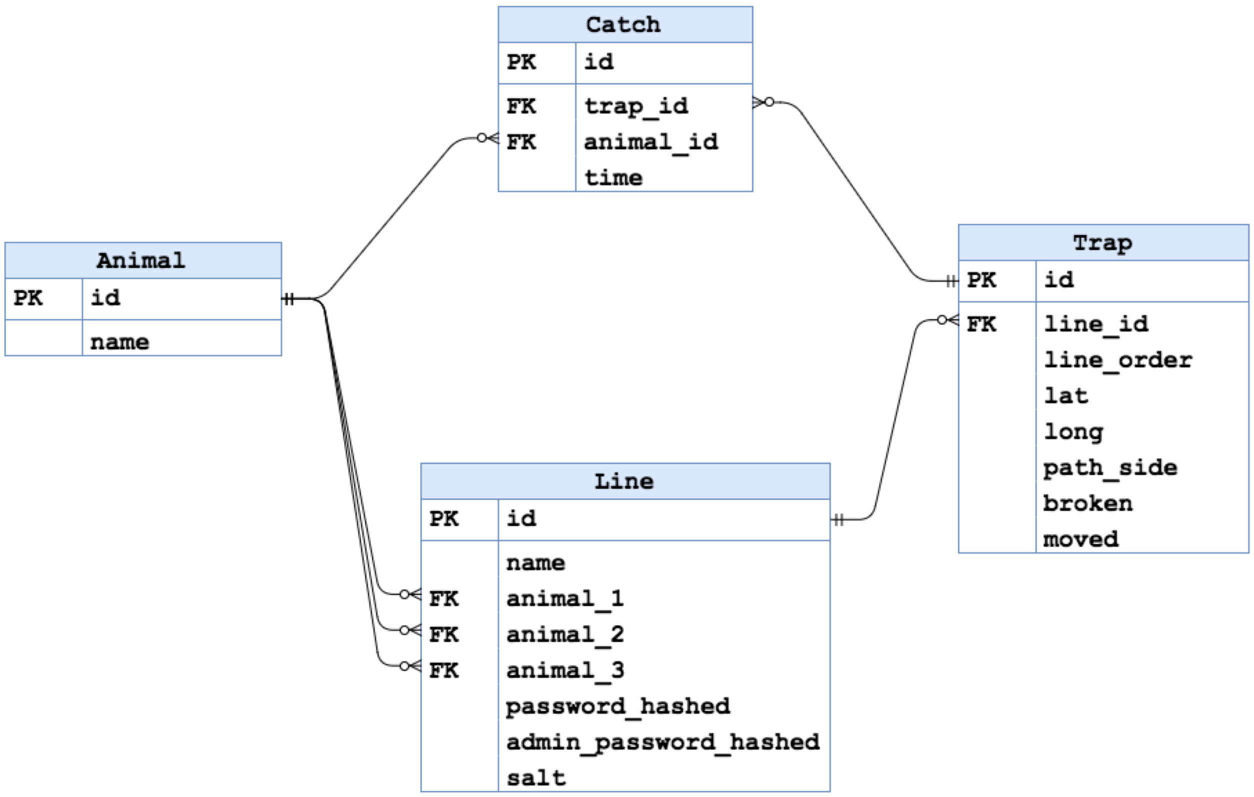
2. Design

2.0.1. Component Diagram



2.1. Server

2.1.1. Database Diagram



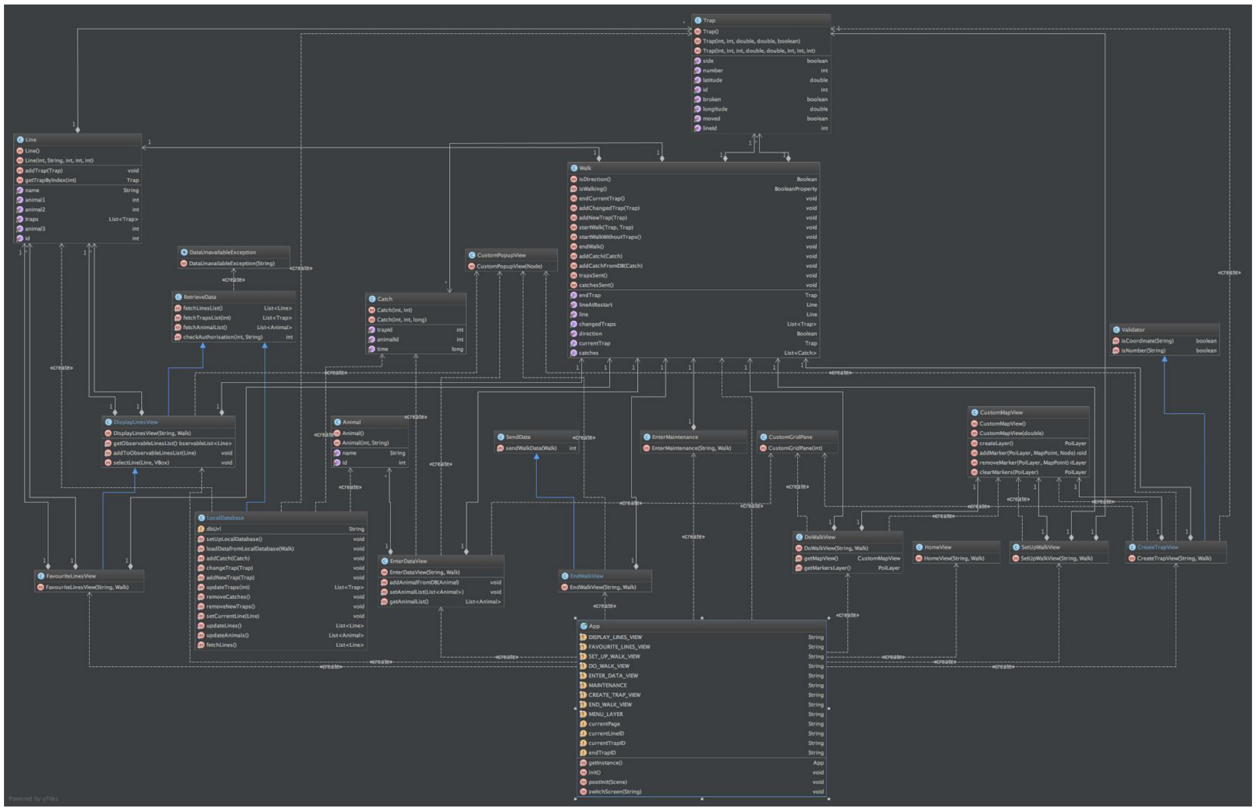
View the larger diagram on draw.io

2.1.2. Server API

View the REST API documentation.

2.2. Mobile Application

2.2.1. Class Diagram



View the larger diagram image.

3. Quality Assurance Reports

3.1. Mobile Application Tests

Evosuite has been used to automatically create unit tests for classes. Evosuite did not produce tests for all classes due to issues accessing external libraries. The classes that did not have tests produced were the user interface classes and the 'custom' classes where we extended an existing class to provide specialised functionality. When running the evosuite tests we found issues automating them due to problems accessing the evosuite library, so we edited the automatically produced files so they did not require this library. The evosuite tests have been kept separate from the manually written tests.

Manual unit tests were written for the classes which did not have unit tests produced by evosuite (except for the user interfaces) and for the critical classes. Critical classes are those which send data to the server, retrieve data from the server, and interact with the local database on the mobile device.

To test the user interfaces we created a manual checklist. These are to be walked through by a person who can verify that the expected result is produced and/or observed when viewing each page in the mobile application. These tests were used because we had difficulty trying to test the interfaces through unit tests, and checklists allows the user to check that buttons, fonts, and colour are suitable and do not make the application difficult to use.

Jacoco has been used to report on test coverage. Statistics from this tool and from GitHub can be found below in project statistics.

3.2. Website and Server Tests

The API was tested using unit tests. There were two different unit test modules to test the API. One for checking that the API returns the correct responses when given the correct data, and the other module for testing exceptions thrown during the processing of a request to the API. These tests were used with Travis CI and were run every time code was pushed to the repository, making sure that any additions to the server would not break the functionality of the API, which the mobile application depends on. Coverage was used to check the code coverage of the tests over the API.

Selenium was used to automate the website testing by testing the functionality of the website through different browsers. The tests are run by running the Selenium Server on a local host and running the test module. Difficulty with setting up the environment for testing the browser restricted which browser we were able to test, and Selenium presented some limitations preventing the whole website being tested, such as the driver not being able to insert data into fields due to the field being invisible despite being tested and found visible beforehand, and failing the recaptcha test. The portions of the website that weren't tested through Selenium had a manual checklist created to test them, where a specific guide was given so developers can follow the steps and confirm the expected result is produced.

3.3. Field Testing

The mobile application has been tested in the field, to verify its functionality. It was run on two Android devices at the same time, and the two walks (one in an open space and one in a more covered space to simulate conditions such as the Manawatu Gorge walk) were completed immediately after one another to prevent conditions such as weather from affecting the results.

We found the map displayed on the screen did not always match the area that we decided to conduct the test in (e.g. the path on the map did not match the path we walked). The map may mislead the user as paths may have moved, been closed off, or additional paths may have been added since the map was created. This is something that we are unable to change, as it is part of the map provider and not our project.

We also found the GPS, while accurate in open spaces such as the university ring road, was less accurate as the environment became less exposed (e.g. under trees, next to inclines). Additionally, this accuracy also seemed to depend on the phone being used. The location also seemed to only update after the user had travelled a minimum distance from the last reported location, which can be confusing and misleading to the user. Unfortunately, as the GPS information is provided and updated by the device, we cannot remedy these issues. Hopefully future devices and operating systems have more accurate GPS reporting in both open and covered areas, but until then, the GPS location reported by the mobile application is limited by the device's capabilities.

3.3.1. Results

Average distance between actual location and location reported by the app on each phone in the open and closed space.

	Open space	Closed space
Phone 1 (HTC1 M7 - Android 5.0.2)	6.25m	27.5m
Phone 2 (1st Gen Motorola Moto G - Android 5.1)	5.2m	7.2m

3.4. User Testing

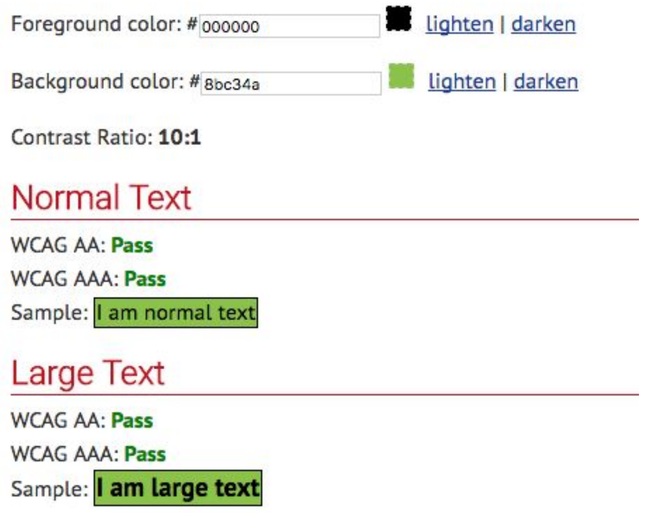
The mobile application and website were tested by a user to verify user acceptance and functionality.

Some issues raised during user testing of the mobile application were:

- The original colour scheme made the text difficult to read due to low contrast.
- Button and font sizes not always being large enough to read and use with ease.
- Some screens had a lot of white space that could be better used to provide larger buttons or map displays.
- It was not clear what the dot colourings (trap and position markers) on the map meant, so a key would be helpful.
- The original colour of the dot indicating the user's location was hard to spot, so a brighter colour with more contrast against the other dots and rest of the map would be good.
- It was not necessarily clear to the user that the map data had or had not been cached and preloaded, or why the map would disappear if the user zoomed in. It would be helpful to notify the user about what's going on.
- Submission button text on various pages changes if all required data has not been supplied, but does not change back once it is supplied, which may confuse users.
- It is not clear to the user that data has successfully been sent to the server. A confirmation page or popup would be useful.

We have tried to find a solution for all of these issues, however, were unable to address some. These include using all white space on a page (as we could not get the screen size to know what white space exists), and notifying the user of map caching (as there was not enough time to produce an adequate solution). The contrast between the text and background has been improved to meet recommended Web Content Accessibility

Guidelines 2.0 by W3C (see image below).



Foreground color: # 000000 ■ lighten | darken

Background color: # 8bc34a 🟩 lighten | darken

Contrast Ratio: **10:1**

Normal Text

WCAG AA: **Pass**

WCAG AAA: **Pass**

Sample: I am normal text

Large Text

WCAG AA: **Pass**

WCAG AAA: **Pass**

Sample: **I am large text**

Some issues raised during the testing of the website were:

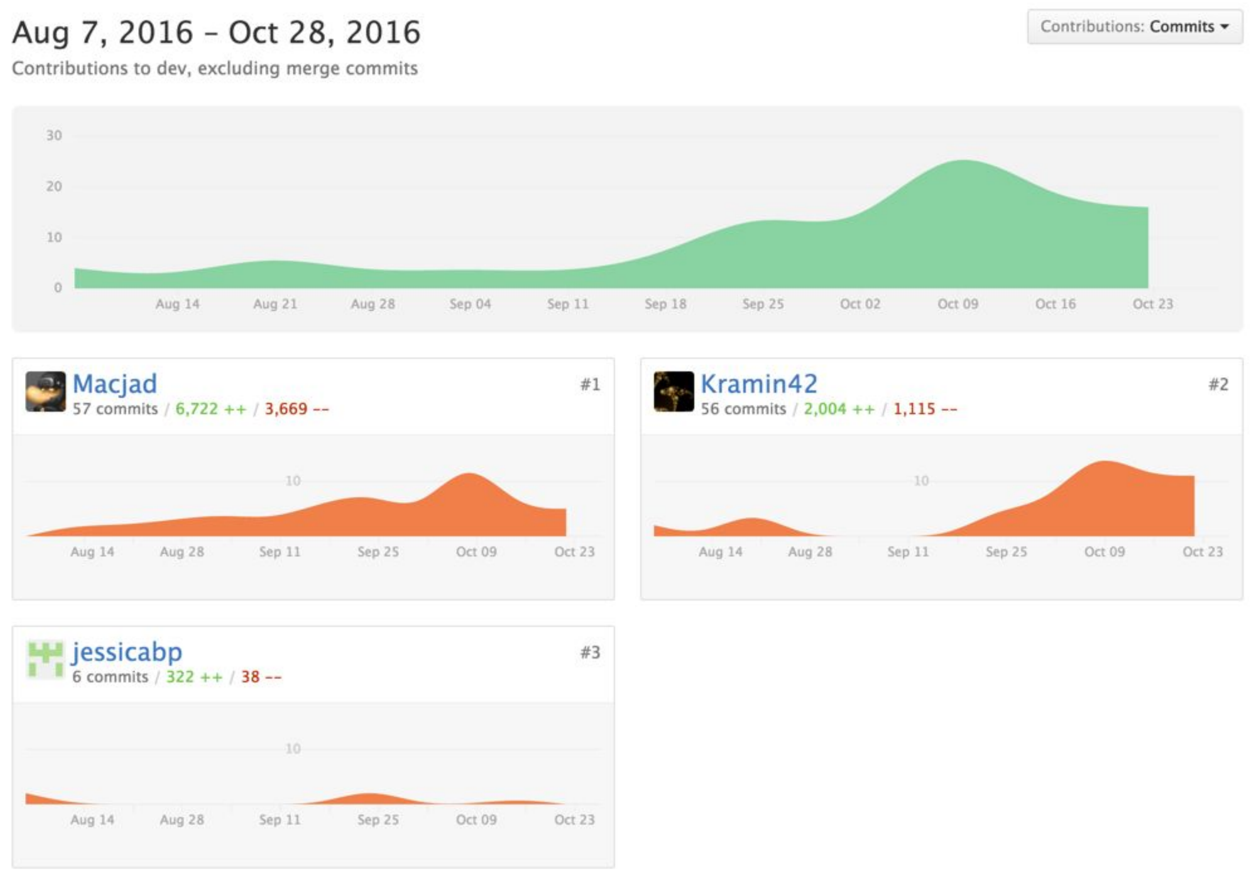
- When adding a trap, the map did not change to show where the trap was located, or include a marker for the new trap. It would be good if it readjusted when a new trap was added to show all existing traps, and if a new trap marker was created for the new trap.
- The page showing a list of catches showed the animal id for each catch. This is not something the user knows about or understands, so should be changed to the animal name. The date was also shown in a dd/mm/yyyy format (e.g. 01/01/2016), which could be confusing depending on the date and user's locality. Instead it should be dd/MMM/yyyy (e.g. 01/Jan/2016) to avoid confusion.
- The website styling was very plain, so has been made "sexier" to be more visually appealing.
- The "about" page was very plain and did not provide much useful information. It could be altered to include an instructional video and links to open source repositories for the mobile application and/or website/server.

As with the mobile application, we have tried to find a solution for all of these issues, however were unable to address some. These include showing new traps on the map when creating them online (due to limitations of the map API), and adding all desired features to the "about" page (as we did not complete all of the components, such as a video).

4. Project Statistics

4.1. Server

4.1.1. Repository Activity

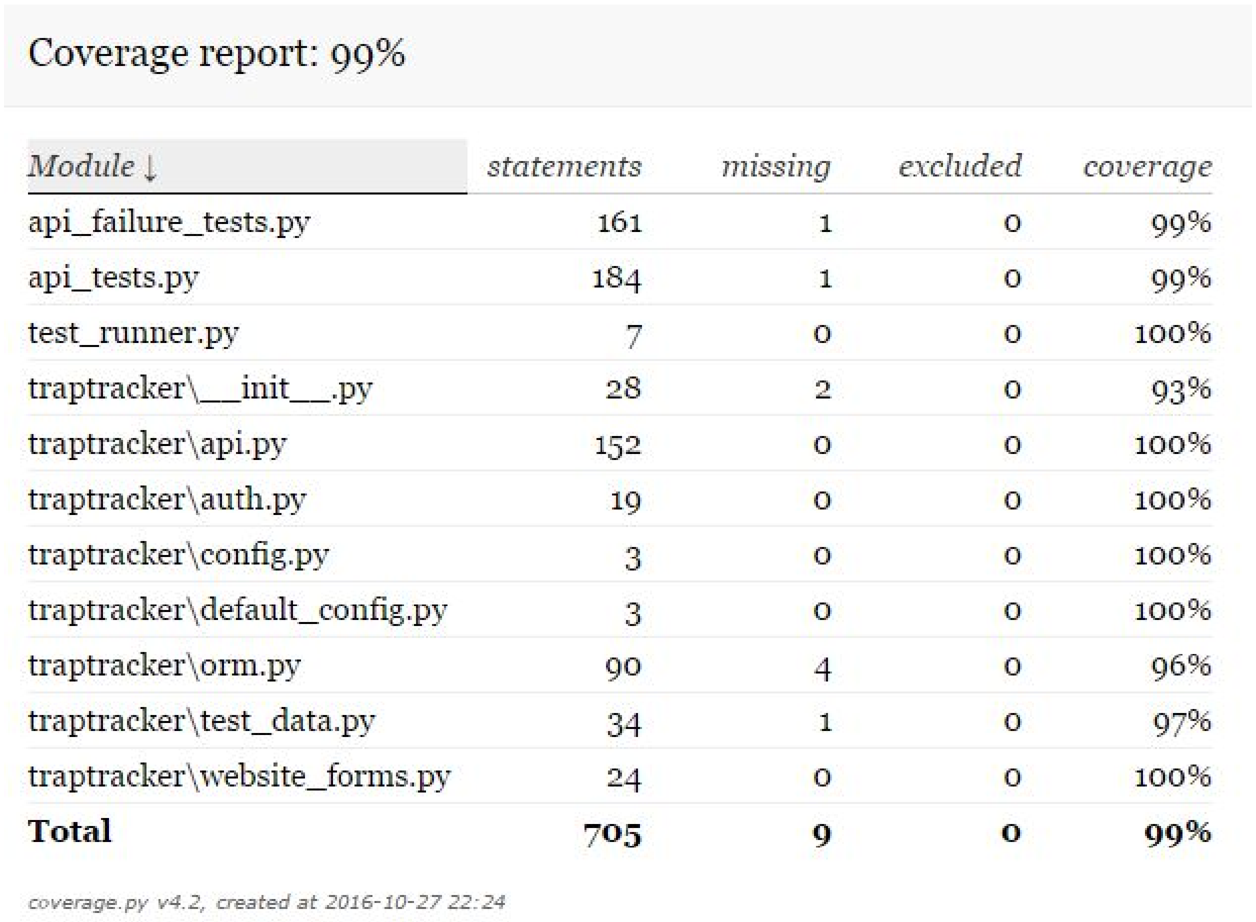


Aug 7, 2016 – Oct 28, 2016
Contributions to dev, excluding merge commits
Contributions: Commits
30
20
10
0
Aug 14
Aug 21
Aug 28
Sep 04
Sep 11
Sep 18
Sep 25
Oct 02
Oct 09
Oct 16
Oct 23
Macjad
57 commits / 6,722 ++ / 3,669 --
#1
Kramin42
56 commits / 2,004 ++ / 1,115 --
#2
jessicabp
6 commits / 322 ++ / 38 --
#3

4.1.2. Issue Tracking

The server has 41 issues (all of which are closed), 16 marked as a bug, and 22 as an enhancement, and 4 as a question (an issue can have multiple tags).

4.1.3. Coverage



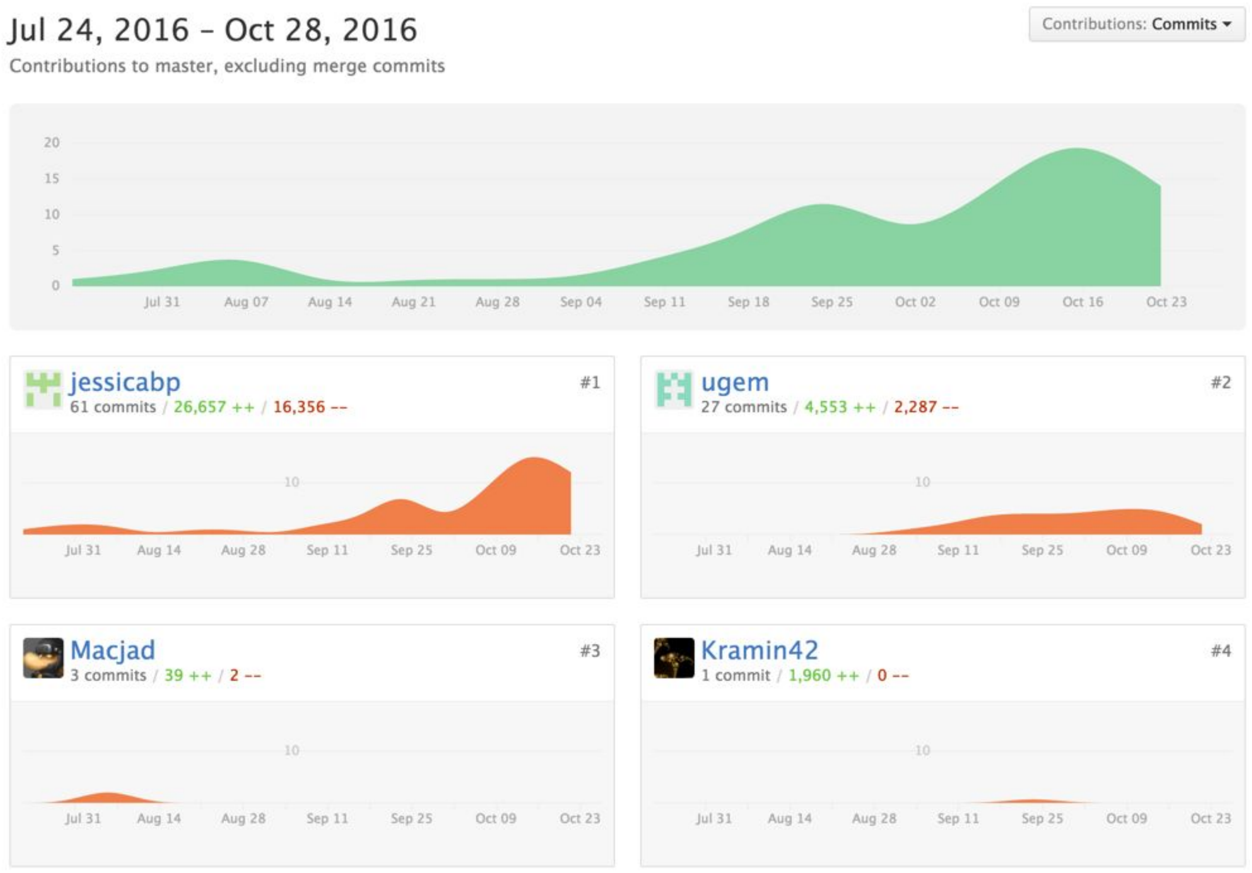
Coverage report: 99%

Module ↓	statements	missing	excluded	coverage
api_failure_tests.py	161	1	0	99%
api_tests.py	184	1	0	99%
test_runner.py	7	0	0	100%
traptracker\__init__.py	28	2	0	93%
traptracker\api.py	152	0	0	100%
traptracker\auth.py	19	0	0	100%
traptracker\config.py	3	0	0	100%
traptracker\default_config.py	3	0	0	100%
traptracker\orm.py	90	4	0	96%
traptracker\test_data.py	34	1	0	97%
traptracker\website_forms.py	24	0	0	100%
Total	**705**	**9**	**0**	**99%**

coverage.py v4.2, created at 2016-10-27 22:24

4.2. Mobile Application

4.2.1. Repository Activity

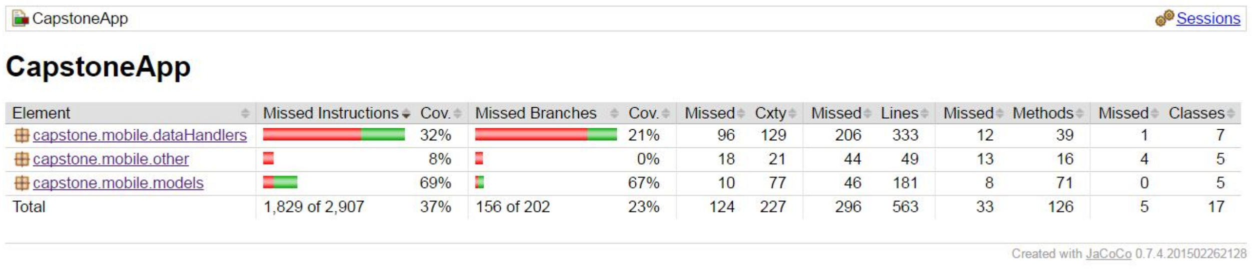


4.2.2. Issue Tracking

The mobile app has 5 remaining open enhancement issues which contain planned features we were not able to implement by the end of the 12 weeks. There are 14 closed issues, 9 of which are labelled bugs, and 5 which are labelled enhancements. The majority of features for the mobile app were planned directly on the trello board, and were not added as github issues.

4.2.3. Jacoco

While we had many tests that ran properly when run manually from Intellij (as shown in the screenshot below), the jacoco task required them to run properly when executed using the 'gradle test' task in order to generate the code coverage statistics. Unfortunately, the gradle task has problems using some of the imported libraries which caused many of the tests to fail, so the jacoco reports do not accurately reflect the coverage levels (as shown in the screenshot below). We tried to remedy this by initially Googling the problems and looking for fixes, however, none of them worked for us. Attempting to use the classpath from a successful single run (to ensure all of the dependences were available) was also unable to give us the desired result. Various other methods were also tried, such as adding additional dependencies to testCompile in the build.gradle script, but none of these were successful.

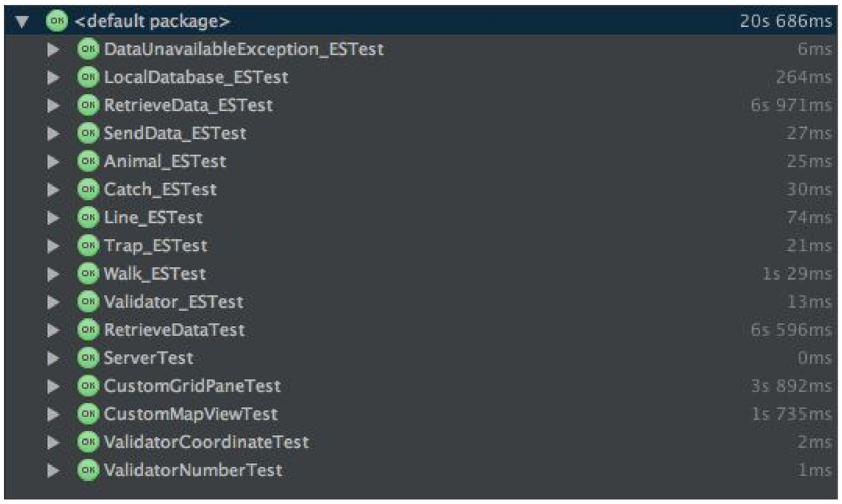


CapstoneApp

CapstoneApp

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
capstone.mobile.dataHandlers		32%		21%	96	129	206	333	12	39	1	7
capstone.mobile.other		8%		0%	18	21	44	49	13	16	4	5
capstone.mobile.models		69%		67%	10	77	46	181	8	71	0	5
Total	1,829 of 2,907	37%	156 of 202	23%	124	227	296	563	33	126	5	17

Created with JaCoCo 0.7.4.201502262128



```
<default package>                    20s 686ms
  DataUnavailableException_ESTest          6ms
  LocalDatabase_ESTest                   264ms
  RetrieveData_ESTest                 6s 971ms
  SendData_ESTest                         27ms
  Animal_ESTest                           25ms
  Catch_ESTest                            30ms
  Line_ESTest                             74ms
  Trap_ESTest                             21ms
  Walk_ESTest                          1s 29ms
  Validator_ESTest                        13ms
  RetrieveDataTest                    6s 596ms
  ServerTest                               0ms
  CustomGridPaneTest                  3s 892ms
  CustomMapViewTest                   1s 735ms
  ValidatorCoordinateTest                  2ms
  ValidatorNumberTest                      1ms
```

5. Installation

To start using Trap Tracker, first visit traptracker.pythonanywhere.com to create a new line. Each line requires a user password (for users who walk the trap line and need to save catch data), an administration password (for the trap line administrator to maintain the line data and change line settings such as passwords), and the three most popular animals caught along the trap line. These three animals are the first options offered to users in the mobile application when entering data for a catch. Then to add traps to the line, someone with the administrator password can either add them on the website, or download the mobile application and add traps from there.

The mobile application can be installed on an iOS or Android device. Once installed, a user can view all available trap lines, or just those they've accessed previously. To view or add traps for a line, users must enter a valid password, or have a valid password saved from a previous use. If there are no traps on the line, they are then prompted to add a new trap using their current location or manually entered GPS coordinates, otherwise they pick what part of the trap line they're going to cover, and the mobile application takes them through the walk, showing the location and side of the path for each trap. At each pre-existing trap they enter whether or not they've found the trap, and if so they enter what type of animal is caught in the trap, or "empty" if there is none. They can also enter maintenance data to mark the trap as broken or moved. When finished, users select a button which sends all data back to the server (as long as they have an internet connection and their password is correct).

On the website, the catches made can then be viewed and downloaded in a CSV by anyone, and the trap line administrator can view the location of the traps and see if any have been marked moved or broken. They can also change the line passwords and popular animals if necessary.

How to videos for users can be found on the Trap Tracker Project YouTube channel:

- Mobile Application
- Website

6. Building the Product

6.1. Server and Website

The source code for the server and website can be obtained by cloning the server repository. Python 3.5 must be installed on the machine, and it is recommended that Pip is also installed. Please note flask defaults to port 5000.
All modular dependances can be installed in the repository's root directory by running:

```
pip3 install -r requirements.txt
```

The server can then be run through:

```
python run.py
```

6.2. Mobile Application

The source code for the mobile application can be obtained by cloning the application repository. The gradle build script (build.gradle) is located within the project root directory. The Gluon plugin (version 3.0.1) is required.

The app can be launched on desktop by running:

```
gradle run
```

A working (debug) APK for android can be created by running:

```
gradle apkDebug
```

A working iOS simulator for iPad or iPhone can be started (on a Mac with Xcode and any other requirements - see the Gluon documentation) by running:

```
gradle launchIPadSimulator
gradle launchIPhoneSimulator
```

7. Limitations

Limitations of the product include:

- Issues scaling mobile application to different screen sizes (due to no way to detect screen size).
- Some browser compatibility issues. The browser must be running javascript, the drop down menus (datalists) to select an animal type do not work in Safari
- The map in the mobile application will not load without an internet connection and the user manually moving the map until all desired areas are loaded. Only once manually loaded is an area of the map cached and available for offline use.
- The GPS location reported by the mobile device is not reliable, and tends to get less accurate as the area it is being used in becomes less open. The inaccuracies are due to the hardware limitations of the devices and GPS, so cannot be corrected in the mobile application.
- We were unable to make the app produce sound alerts to let the user know they’re close to the current trap. Gluon does not currently support this, and we did not have enough time or knowledge to create native solutions for iOS and Android.
- The map on the website does not change to focus on the currently selected trap, and if the user adds a new trap on the website, it is not added to the map until the page is reloaded.

8. Commercialisation Plan

If this project is to continue in the future, it should be passed on to companies and organisations to share with their contacts, employees, and volunteers. The intention of this is to generate a larger user base and spark interest for developers to improve upon and maintain the project (as it is open source), or possibly produce funding for this development. Examples of New Zealand groups who could fulfill this role are the Department of Conservation, Oroua Blue Duck Project, and Predator Free NZ, all of whom are doing work to eliminate pests and may therefore have a use and user base for this project.

If further development occurs on this project in the future, some features we suggest including are:

- The ability to add new catches on the website
- Sounds and vibration in the mobile application to notify the user they are close to the current trap, or that they have passed it
- Automatically detect when the mobile device has a network connection and send data to the server without the user's input
- Ability to add images to captures in the mobile application
- Addition of loading screens or clearer indications that the mobile application has received the user's instruction
- Sizing user interface components based on device screen size
- Collecting device metadata from devices using the mobile application to detect the accuracy of different device types and provide correction through machine learning
- Better browser compatibility
- Caching of all relevant map area when a line is selected, and/or notifications to alert the user to the caching status
- Showing new traps immediately on the map when adding them on the website
- Graphs of the data on the website
- Ability to download XLSX files from the website, not just CSV
- Ability to create new animal types in the mobile application

These features were not completed by us for a variety of reasons including the limited time, capabilities of our tools (such as the Gluon plugin and map APIs), and limited knowledge and resources.