Week Eight Milestone Report

Project Nest

Massey University Capstone 2016 - Group Two

Team Members:

Zoe Purdon-Udy (13047928)

Francis Greatorex (14136398)

Sam Hunt (14216618)

MJ Lee (13214654)

Contents

Process

Updated Project Plan

The table below was produced for the Week Four Milestone Report. Each requirement is now coloured according to its development:

Red - the requirement has been removed from the project plan, it will no longer be developed during the initial project timeline (ending in week 12)
Orange - the development of the requirement is ongoing
Green - the requirement is completed
Uncoloured - development is yet to begin on this requirement

Requirement	Note	Priority Level	Expected Delivery
Mobile app. available on iOS & Android (NOTE: this does not mean available to the public but rather compatible with both iOS and Android)	This is a core requirement specified by the capstone project outline and necessary to achieve the non-functional requirements of usability and accessibility	1	Week Four Prototype (14/08/16)
Support for mobile OS beginning at Android V4.4 and iOS 7	Requirement specified by capstone project outline	2	Week Four Prototype (14/08/16) Will not be tested on multiple versions until later in the project
Mobile app. can be used to locate traps	This is the most essential functionality of our app.	1	Week Four Prototype (14/08/16) With features and fine tuning ongoing throughout the project

Master data for a trapline can be configured	Essential functionality for the app.	2	Week Eight Milestone
Security for the trapline data	Requirement specified by capstone project outline	2	Week Four Prototype (14/08/16)
App. can be used to capture data, data is held in a central store, data can be migrated to another storage service	This is essential to satisfy the purpose of the app. and ensure the extensibility of the app.	1	Week Four Prototype (14/08/16)
Data can be exported as a CSV		3	Week Eight Milestone
Web app. can be used to modify user and trapline information		2	Week Eight Milestone
Web page displays aggregated data		4	Week Eight Milestone
Mobile app. can be used to record audio and send to other open source projects	This feature is above and beyond the initial scope set out by the capstone project outline, however, the developers think it would add value to the mobile product presented to DOC	N/A	Week Eight Milestone
Photos can be attached to captured data	This feature could add significant value to the data recorded	2	Week Eight Milestone
The mobile app. can be used to record damage to traps and associated hazards	This feature could add significant value to the application	2	Week Four Prototype (14/08/16)
The mobile and web applications can be displayed in both English and Te Reo Maori		5 (NOTE: priority level has changed)	Week Eight Milestone

The Cacophony Project

The idea to include support for audio recording, which could be sent to the open source Cacophony Project, has been abandoned due to the limited time remaining for development.

Open Tasks

The open tasks come from the requirements that still need to be delivered and open issues on the Github issue tracker

Support for mobile OS beginning at Android V4.4 and iOS 7

The mobile app needs to be tested on Android V4.4 and iOS 7

Mobile app. can be used to locate traps

Non-visual cues for locating the trap are yet to be implemented

Data can be exported as a CSV

Historical data from specific traps or entire traplines can be exported from the web app.

Web page displays aggregated data

The ability to display aggregated data on the web app would add value to the project, however, it is not a core functional requirement so its priority is low.

Photos can be attached to captured data

This needs to be added to the mobile app for the following task (damage logging) to be implemented in the way planned.

The mobile app. can be used to record damage to traps and associated hazards

Recording damage to trap units is currently not implemented in the mobile app.

The mobile and web applications can be displayed in both English and Te Reo Maori

This feature has very low priority. Its addition to the project has no impact on a successful product delivery.

Send logged catches to the API

Tagged as App, core #75

Automatically load traplines/traps

Tagged as App #74

Implement automated tests in the API

Tagged API, core #68

Fetch new session on expiry

Tagged App #66

Show catch names on catch selection screen

Tagged App, core #65

404 expired session token

Tagged API, bug, core #64

Logout functionality

Tagged App, enhancement #63

Image referencing

Tagged Web, core #62

Login page doesn't shift focus to username form

Tagged Web, enhancement #55

Second trapline entry appears on refresh

Tagged App, bug #49

Session timeout

Tagged API, enhancement #44

Support API DELETE requests

Tagged API, enhancement #35

Support API PUT requests

Tagged API, core #34

Non-visual distance cues

Tagged App, enhancement #24

Custom error pages

Tagged App, Web, enhancement #23

Use larger buttons

Tagged App #22

Authentication failed

Tagged API, WEB, bug #12

Completed Tasks

Please view the closed issues for this project at:
https://github.com/FrancisG-Massey/Capstone2016/issues?q=is%3Aissue+is%3Aclosed

Mobile app compatible with iOS and Android

Mobile app can be used to locate traps

Master data for a trapline can be configured

Security for the trapline data

Mobile app can be used to capture data, data is held in a central store, data can be migrated to another storage service

Web app can be used to modify user and trapline information

Architecture / High Level Design

Comments from Week 4 Report:

Component Design

Your software has three central components: API, Web App Rendering and Mobile-Front End. Database storage could be listed as a fourth component, but neither the web-browser nor the file system are components of your application. The multi-tier hierarchy is already shown in your Architecture diagram, and that's where it belongs.

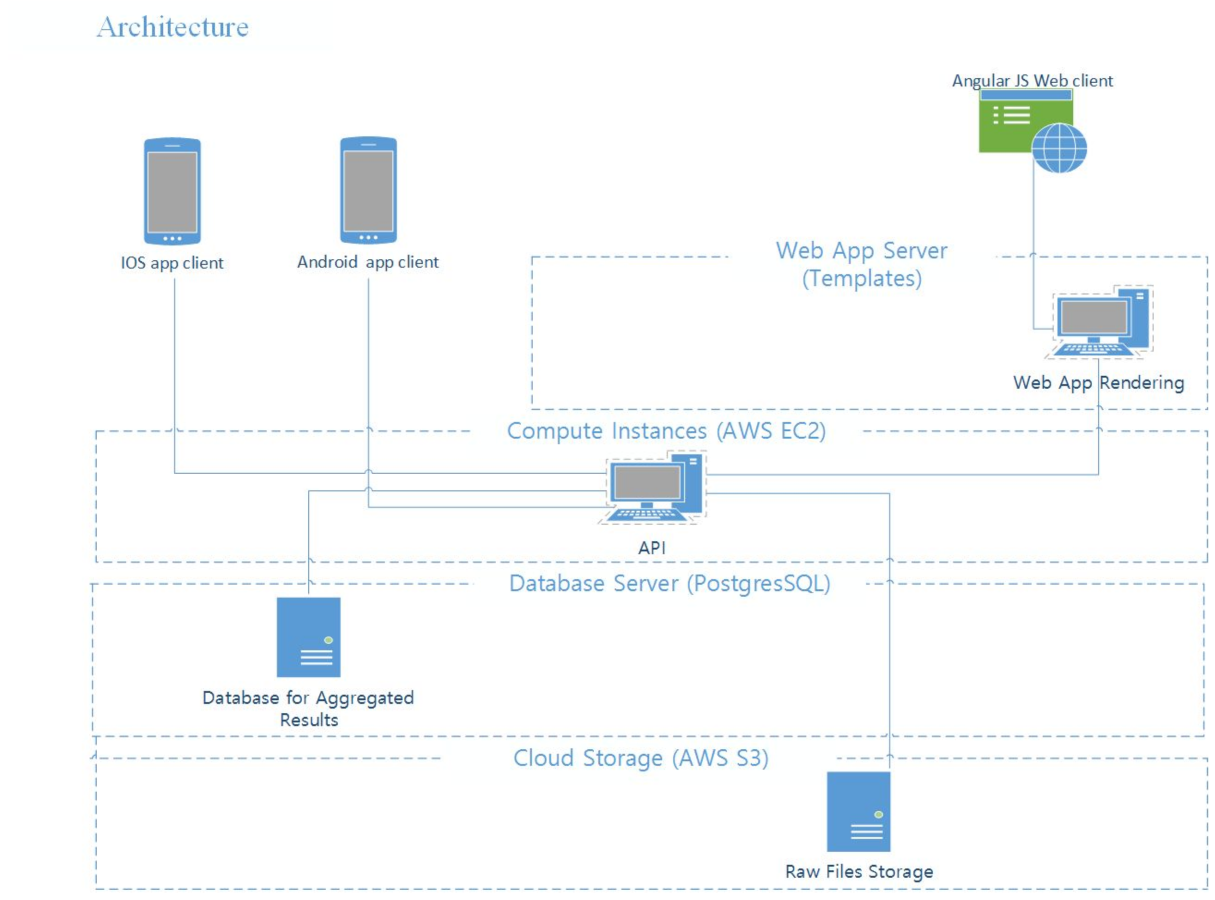
Deployment Diagram

You may want to show different users using the iOS, Android and website, and give them names (e.g. trap-setter, admin)

Database Design

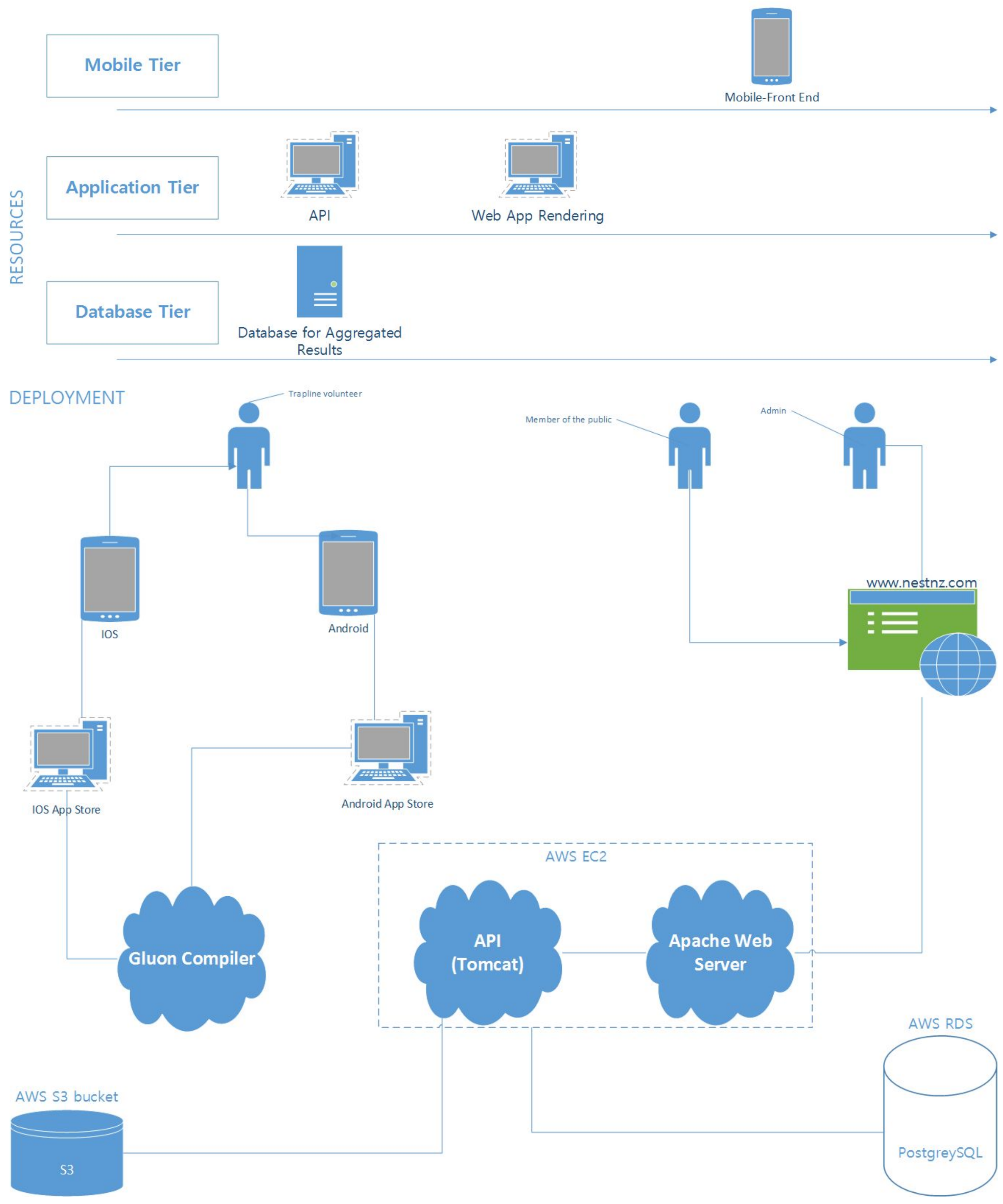
The link between Trap and Trapline seems problematic, and I would identify traps via trapline + number.

Updated Diagrams

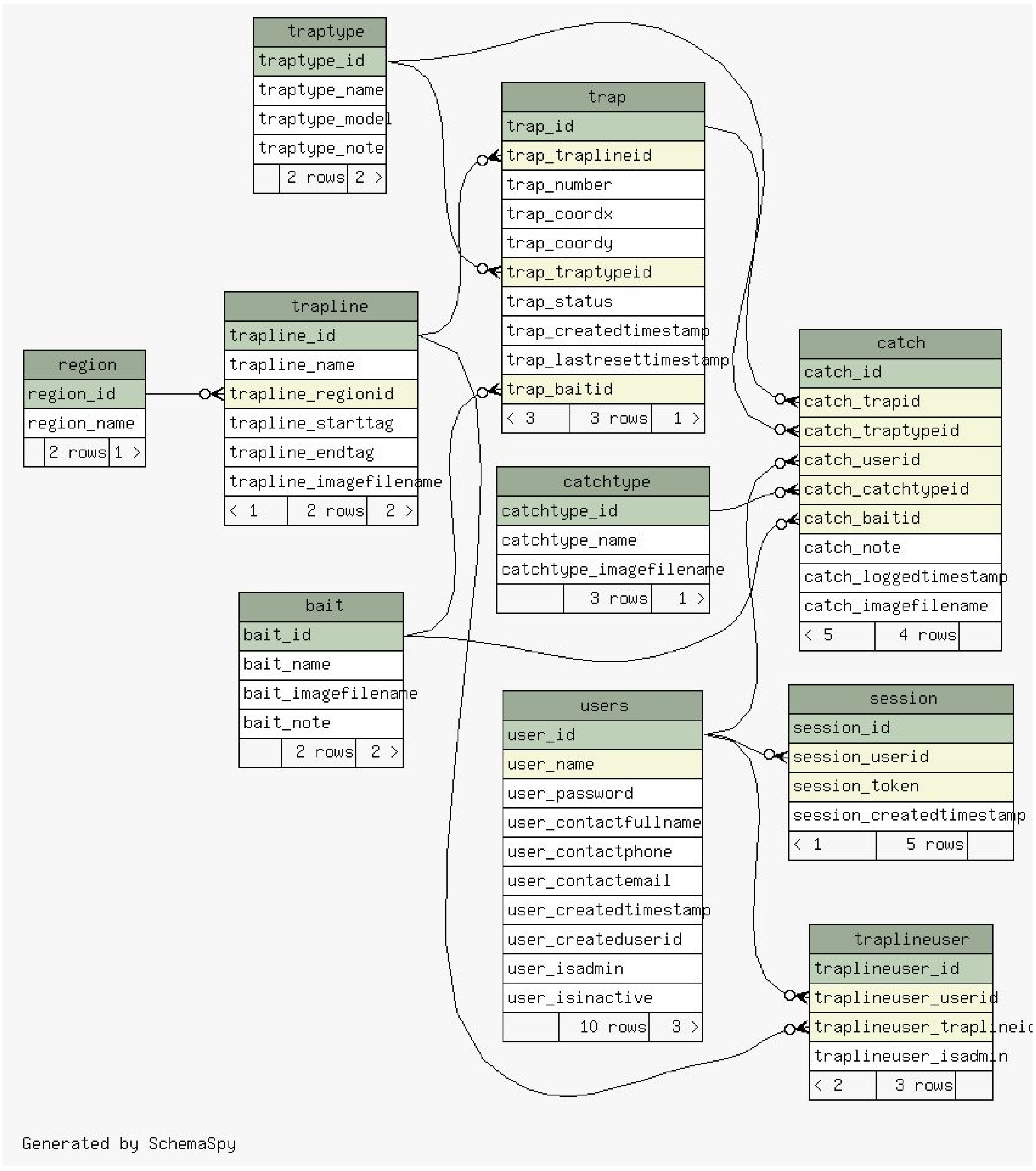


Note: No changes have been made to the overall architecture design show above

COMPONENTS



Mobile Tier
Mobile-Front End
Application Tier
API
Web App Rendering
RESOURCES
Database Tier
Database for Aggregated Results
DEPLOYMENT
Trapline volunteer
Member of the public
Admin
IOS
Android
www.nestnz.com
IOS App Store
Android App Store
AWS EC2
Gluon Compiler
API (Tomcat)
Apache Web Server
AWS RDS
AWS S3 bucket
S3
PostgreySQL


traptype
traptype_id
traptype_name
traptype_model
traptype_note
2 rows 2 >
trap
trap_id
trap_traplineid
trap_number
trap_coordx
trap_coordy
trap_traptypeid
trap_status
trap_createdtimestamp
trap_lastresettimestamp
trap_baitid
< 3 3 rows 1 >
trapline
trapline_id
trapline_name
trapline_regionid
trapline_starttag
trapline_endtag
trapline_imagefilename
< 1 2 rows 2 >
region
region_id
region_name
2 rows 1 >
catch
catch_id
catch_trapid
catch_traptypeid
catch_userid
catch_catchtypeid
catch_baitid
catch_note
catch_loggedtimestamp
catch_imagefilename
< 5 4 rows
catchtype
catchtype_id
catchtype_name
catchtype_imagefilename
3 rows 1 >
bait
bait_id
bait_name
bait_imagefilename
bait_note
2 rows 2 >
users
user_id
user_name
user_password
user_contactfullname
user_contactphone
user_contactemail
user_createdtimestamp
user_createduserid
user_isadmin
user_isinactive
10 rows 3 >
session
session_id
session_userid
session_token
session_createdtimestamp
< 1 5 rows
traplineuser
traplineuser_id
traplineuser_userid
traplineuser_traplineid
traplineuser_isadmin
< 2 3 rows
Generated by SchemaSpy

Usage of Project Infrastructure

Issue Tracking System

The initial plan for tracking issues in this project was to separate 'formal' issue tracker from informal idea generation and progress tracking. Formal issue tracking was to be completed through the built in Github issue tracking system and Trello was planned to be used for informal issue tracking, progress tracking, and as a place to record ideas for future improvements. However, Trello was not often updated by members of the group and so Github became the main issue tracking system for all issues and plans. The project Trello board is still in use, primarily as a place for the project manager to document the progress of each team member and to keep track of the reports and documentation to be produced. A more in depth look at the use of the Github issue tracker follows.

Github

Issues classified by which component of the project they affect

- App
- Web
- API

Issues tagged with their category

- Bug
- Enhancement
- Question (meaning point for discussion)
- Core

Enhancements are features which will be added if time permits. The 'enhancements' which are not completed by the time of the final presentation and product submission, will be recorded in a report detailing future plans for Project Nest.
Issues with the tag 'core' are important bug fixes or features with a high priority. These issues must be resolved before week twelve.

Patterns and trends in the issue tracker

Total commits: 211
Branches: 4

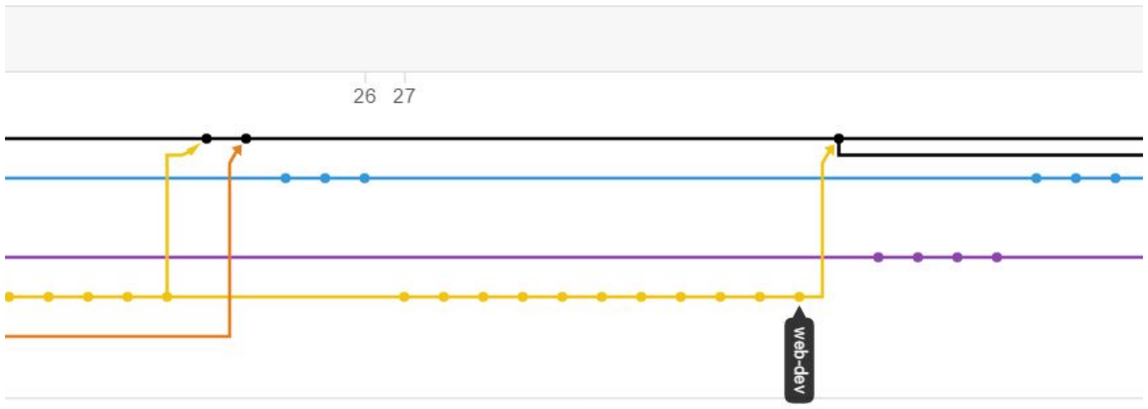
Team member	Issues opened	Assigned issues (open)	Assigned issues (closed)
Francis	21	9	11
MJ	11	4	5
Sam	26	9	30
Zoe	8	3	3

The patterns that can be seen in the table above suggest that the more prolific the team member is at creating issues, the more he or she will have closed. Most of the issues logged are self assigned by the contributor.

	Core	Bug	Enhancement	Question
Web	1:0	1:1	2:0	0:0
App	3:0	2:1	2:0	0:0
API	4:3	2:14	3:13	1:3

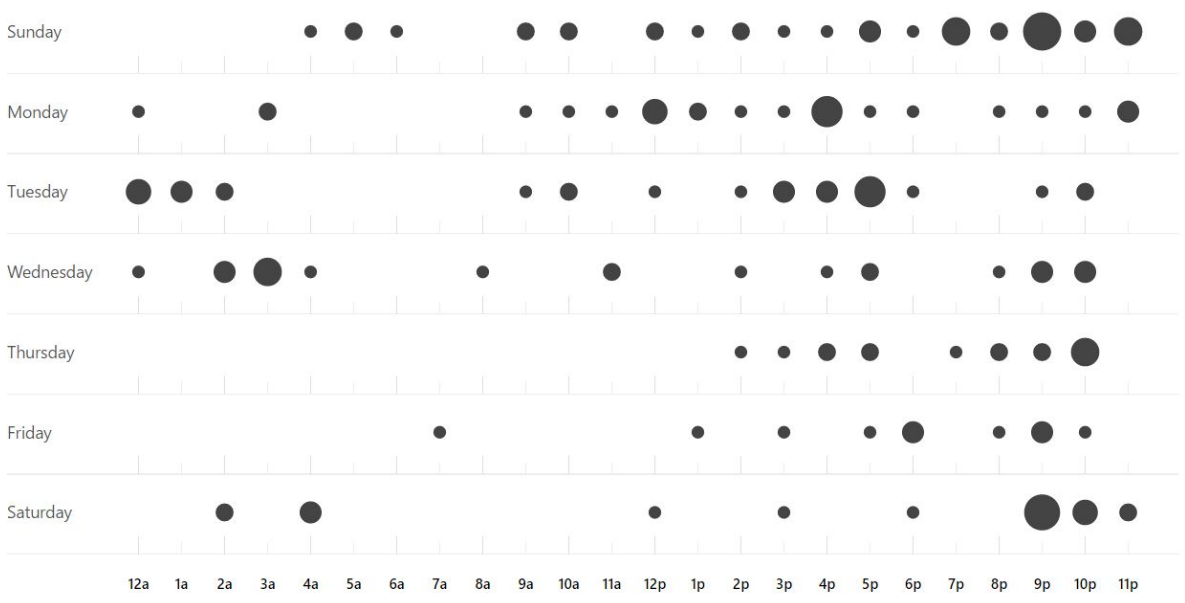
Note: x:y means x open issues and y closed issues

Network Graph



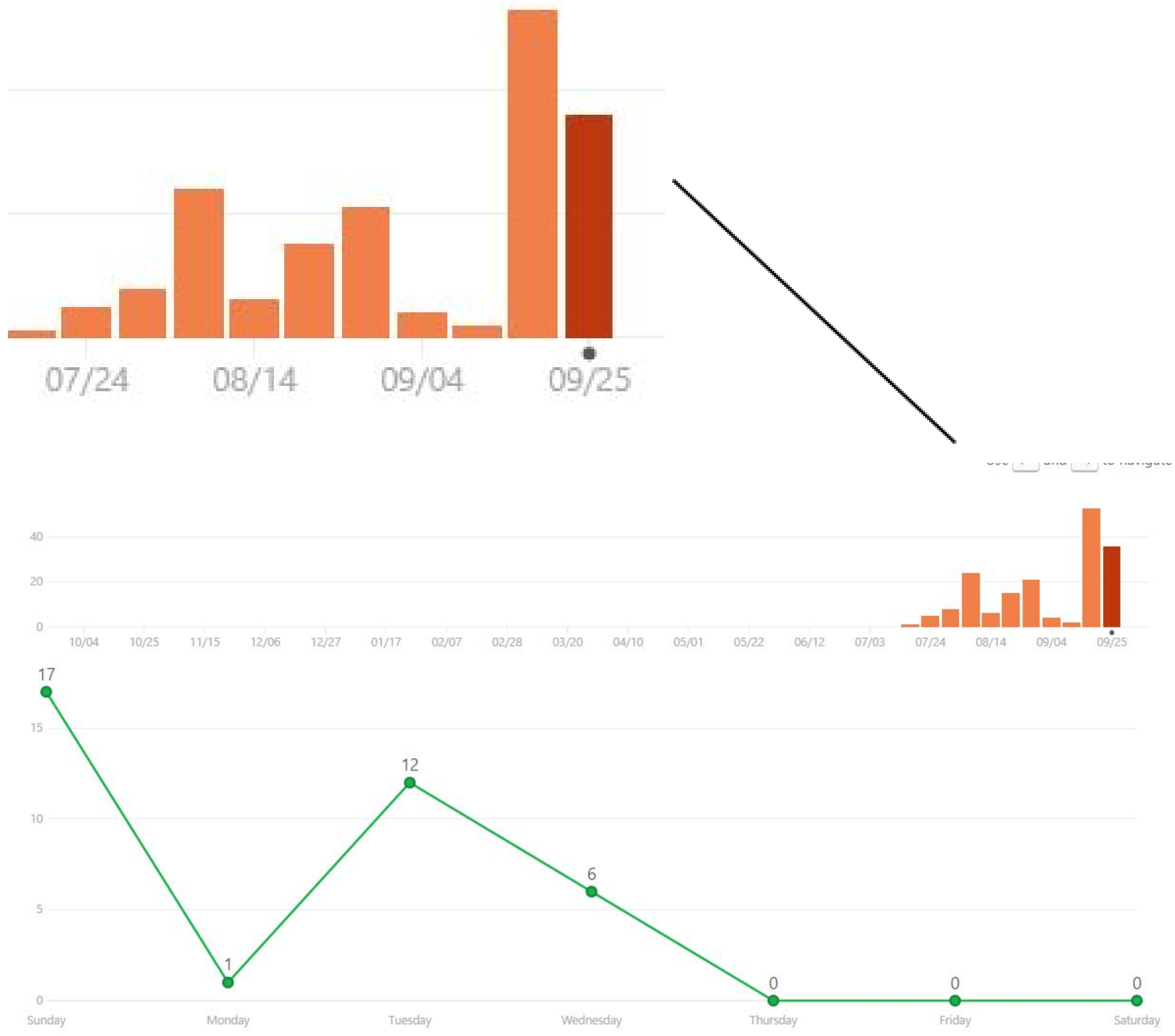
Note: The Network graph shows the branch history of the entire repository network, including branches of the root repository and branches of forks that contain commits unique to the network. (*from: https://help.github.com/articles/about-repository-graphs/)*

Punch Card Graph



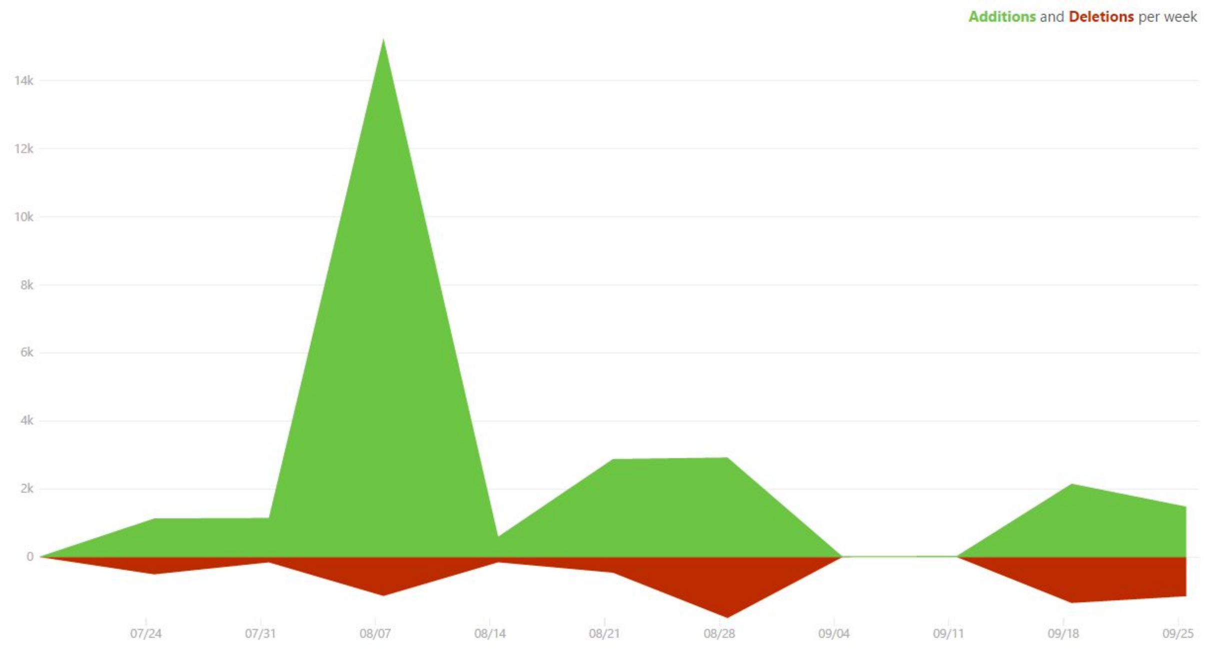
Note: The Punch card graph shows the frequency of updates to your repository based on day of week and time of day. The size of the black circle indicates commit frequency. This graph doesn't include merge commits.
(*from: https://help.github.com/articles/about-repository-graphs/*)

Github-generated Commit Graph

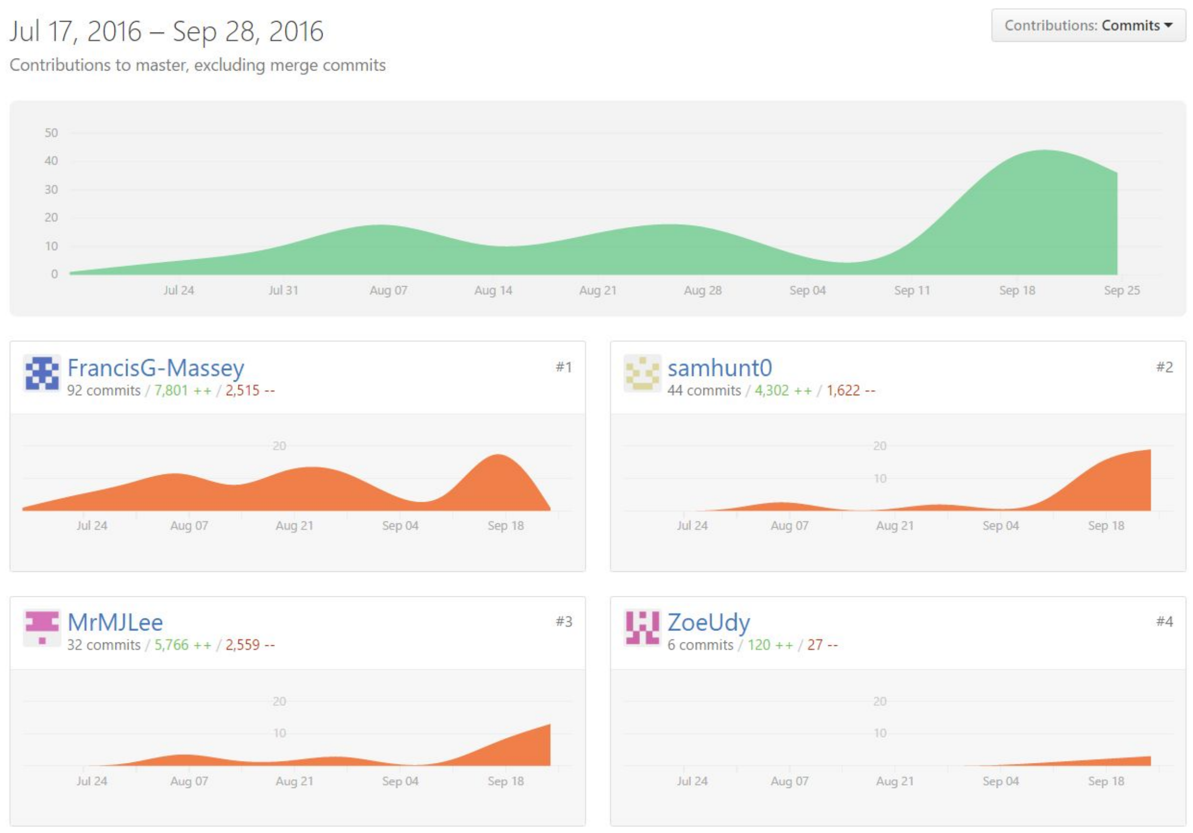


Note: the green line shows the commits from the most recent week (beginning 26th September)

Additions and Deletions (per week)



Member Contributions (Commits)


Jul 17, 2016 – Sep 28, 2016

Contributions to master, excluding merge commits

Contributions: Commits

FrancisG-Massey #1
92 commits / 7,801 ++ / 2,515 --

samhunt0 #2
44 commits / 4,302 ++ / 1,622 --

MrMJLee #3
32 commits / 5,766 ++ / 2,559 --

ZoeUdy #4
6 commits / 120 ++ / 27 --

There was a significant rise in contributions from each of the four team members in the week from the 18th of September. This reflects the team's efforts to achieve their initial goals (outlined in the week four milestone report) before the submission of *this* report and the prototype demonstration on the 30th of September.
The patterns of contributions reflect the coding styles of each team member. There were less commits made around the semester break (28th August - 11th September).
Zoe's commits are significantly fewer than the other team members as her role throughout the project has been to coordinate the team's activities, complete documentation, design user interface components, and write reports and assignments for the universities requirements.
The commit log also shows the success of the teams coordinating roles. Each of the three development branches have been almost solely committed to by their coordinator:

1. web-dev branch : MJ Lee
2. api-dev branch : Sam Hunt
3. app-dev : Francis Greatorex

Dividing the development into clear sections has allowed for rapid and uninterrupted progress, and for each coordinator to incorporate technologies in which they did not have prior experience. If each team member was required to contribute to every section of the codebase, they would all need to learn and understand several different technologies. Therefore, they would be unable to achieve the same level of in-depth knowledge possible by focusing mainly on their assigned areas/technologies.

Quality Assurance

Automated Regression Testing

Mobile

The regression testing that we have implemented currently consists of unit tests which are performed automatically when the Gradle 'check' task is run. This test suite currently contains unit test sets for three components:

- The disk caching service, which saves traplines on the local device for use where the device does not have internet access.
- Ordering of traps in the navigation view
    - 'Next' button appears if and only if there are more traps in the trapline
    - 'Previous' buttons appears if and only if there are traps prior to the current trap
    - 'Next' and 'Previous' buttons display the correct traps
- The trap position map layer
    - The user's current position appears on the map in the correct location
    - All traps in the trapline appear on the map
    - 'Active' traps are marked with the correct icon (a different icon to 'inactive' traps)

Whitebox testing has been employed for the testing of the mobile app as it isolates the component to be tested from any outside factors, such as requests to the API or GPS information from the device.

Web

Selenium IDE is a Firefox plugin which is being used to record user actions and to export them as a reusable test script which imitates the user behaviour. The tests in the IDE are limited to Firefox, however, when the test cases are exported as JUnit4 this limitation of Selenium IDE can be overcome by using WebDriver. WebDriver automates browser compatibility testing by launching the various web browsers from JUnit. This forms the basis of regression testing for the web component.

Current tests exist for the home page, the 'main' template layout (which is used across all of the web pages), and the login forms.

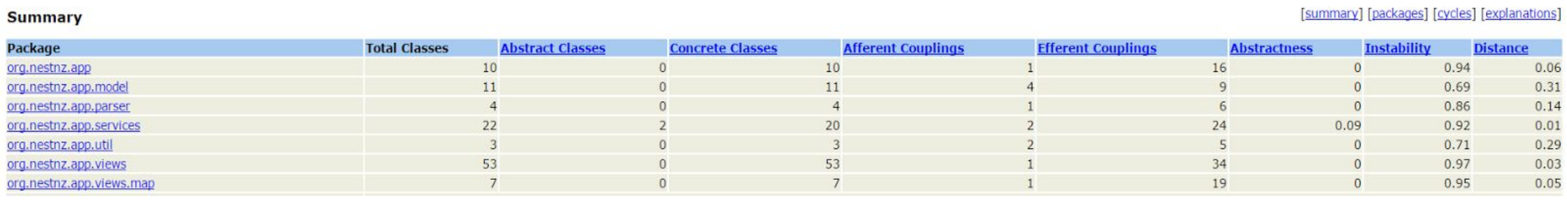
API

JUnit is being used to test the API, in addition to rigorous manual black box tests using ARC Advanced Rest Client (cURL + GUI as a browser extension) until these can also be automated into the project.

Other Quality Assurance

JDepend

JDepend generates design quality metrics for Java packages, it measures quality in terms of extensibility, reusability, and maintainability.
JDepend has been used to analyse the codebase for the mobile app.



Summary

[summary] [packages] [cycles] [explanations]

Package	Total Classes	Abstract Classes	Concrete Classes	Afferent Couplings	Efferent Couplings	Abstractness	Instability	Distance
org.nestnz.app	10	0	10	1	16	0	0.94	0.06
org.nestnz.app.model	11	0	11	4	9	0	0.69	0.31
org.nestnz.app.parser	4	0	4	1	6	0	0.86	0.14
org.nestnz.app.services	22	2	20	2	24	0.09	0.92	0.01
org.nestnz.app.util	3	0	3	2	5	0	0.71	0.29
org.nestnz.app.views	53	0	53	1	34	0	0.97	0.03
org.nestnz.app.views.map	7	0	7	1	19	0	0.95	0.05

It ensures the app is designed in a way which minimises inter-package dependencies (particularly within concrete classes), so changes made to classes within one package are unlikely to affect classes in a different package. The metrics given show, in most cases, this has been achieved quite successfully, with only the "model" and "util" packages holding a high distance value, due to the model being used to communicate between the services & views, and util being a general package for miscellaneous classes.

Usability Testing Plan (Mobile)

Goal

The goal of the user testing is to gain feedback about the suitability of the mobile application for the target audience of Department of Conservation (DOC) volunteers.

Questions

- Is the interface clear - do users know what each button does?
- Is it easy to navigate - do users know how to get to the task they want to complete?

Equipment/Environment

There will be two possible environments in which the testing can occur; field testing, or 'home' testing. Home testing will be completed by the user in their home (or any non-bush/non-trapline location), they will be asked to imagine the scenario of resetting traplines and perform tasks on the application. Field testing will be completed by users when they are resetting traps or otherwise walking along a trapline.

Home testing will not be in the natural environment for application usage, however, it is a suitable environment for gathering the users opinions on the application design and layout. Field testing will be essential to gain information about the overall usability of the app.

The equipment used will be mobile devices owned by the participants and the Project Nest mobile app which will be made available for download.

Participants

Participants will be volunteers for the Manawatu Trapline and will have access to a mobile device with internet connectivity. They will be of varying ages and with varied technological abilities.

Contact

Our contact for recruiting test participants from the volunteer group will be Dr Mike Shepard, the coordinator of the Manawatu Gorge trapping project.

Pre Test Questions

These questions will be asked to determine the environment in which the testing was completed, the skill level of the participant (based on their experience with mobile applications), and what their role in the volunteer project is.

- What is your involvement with the Manawatu Gorge Trap Monitoring project?
- Are you testing the app in the Gorge?
- How many mobile apps do you use on a daily basis?
- How many mobile apps do you use on a weekly basis?
- Please name the top three mobile apps that you use regularly. (Leave blank if none)
- Do you wear gloves when resetting the traps?
- Do you suffer from hearing loss?
- What information about the Manawatu Gorge Trap Monitoring project (statistics, process, how to get involved etc.) do you think should be shared with the public (via a website)?

Tasks

The following tasks have been selected to ensure the participant views many of the screens available on the Project Nest application, therefore testing their ability to navigate the application.
'Home environment' testers

- Log in with the credentials
  - username: test
  - password: testnest
- Select 'Test Trapline'
- Add a trap
- Log a catch at that trap
- Log out of the app

'Field' testers

- Log in with the credentials
  - username: gorgetest
  - password: gorgetestnest
- Select the 'Manawatu Gorge Trapline'
- Walk the trapline as normal and use app to log a catch at each trap

- Log out when you have finished the trapline
- Log in when you have a guaranteed internet connection

Test procedure

- The participants will be told about the application and its purpose
- The participants will all use a unique instance of the application
- Each participant should complete all tasks
- The tasks will be completed on a mobile device

Data collection and analysis

Qualitative data

The test participants will be asked to complete a questionnaire which will include the following questions:

- How easy was it to complete tasks?
- Is the app performance 'fast' enough i.e. are you satisfied with the speed of the apps responses?
- Does the application freeze or crash?
- Are the GPS locations on the application accurate? (for field testers only)
- What tasks did you find difficult? Why?
- Are there any other suggestions for things to add to the app?
- Are there any other comments?

Product

Mobile Application

Desktop

To launch the mobile app on a desktop computer with JDK 8, JavaFX, and Gradle installed, a command prompt should be opened to the "ProjectNest-App" folder of the repository and the command "gradle run" executed.

Android

As with desktop, JDK 8, JavaFX, and Gradle must be installed. Additionally, the android SDK must be downloaded onto the computer with Android SDK 21 & Android Build Tools 21 installed. The ANDROID_HOME variable in the user's gradle.properties file must be set to the path of the top level of the android SDK directory. See here for information on the gradle.properties file: https://docs.gradle.org/current/userguide/build_environment.html

To build the android APK file, the command "gradle android" needs to be run in the "ProjectNest-App" directory. After building, the file "build/javafxports/android/ProjectNest-App.apk" must be copied onto the device, opened on the device, and installed.

iOS

A computer with MacOS installed must be used to build and deploy the app to iOS. JDK 8, JavaFX, and Gradle must be installed, along with XCode 7 (if a free provisioning profile is used) or XCode 6 or 7 (if a paid developer profile is used).
XCode 7 must be used to create a provisioning profile for "org.nestnz.app". To create a free provisioning profile:

Note: An Apple id is required to register as a developer in XCode 7

1. Create a new project by filling product, organization names, and organization identifier
2. Select your iOS device
3. Navigate to 'No Matching provisioning profile found' and click 'fix issue' button, this will provide a free provisioning profile under your account for 7 days

After the profile is created, the desired iOS device should be connected to the computer via USB and the command 'gradle buildIOSDevice' should be run from the terminal.

Web Application

The web application is available publically from nestnz.org. It is hosted in the Apache web server on Amazon EC2 instance. It uses AWS load balancer to handle incoming requests from clients and deliver to the web server. This also listens port 80 and 443 for HTTP and HTTPS. The load balancer checks for health status of our web server every 30 seconds.

API

The API runs in a Java Servlet container in Tomcat. The following dependencies are used:

- Java JDK 1.8
- Apache web server
- Tomcat 8.0.27.0
- Postgresql DB 9.5
- Postgresql JDBC driver 9.4.1209

Once Tomcat and Apache are installed and configured, the project can built to .WAR web archive with ANT, and then deployed into Tomcat.

The API can be accessed over HTTPS at api.nestnz.org, or www.nestnz.org/api (temporary). The API blocks all requests which do not map to the supported REST entities with a “400 bad request” HTTP response code.

A complete list of REST entities, supported functions, and parameters can be found in the completed Nestnz REST Specification document on Google Drive.
Access at https://docs.google.com/document/d/1zPCohwShxziF4YAV0E3PPZWvtZiAU8KaAOtFYIzzbc4/edit?usp=sharing

Users must log in to gain access to the API functionality by making a POST request with their user credentials to /session. Upon authentication the client will receive a Session-Token header which they can append to subsequent queries to gain access to API functions.

A permissions model is used to further limit access to administrative functionality in the API.
Access at https://docs.google.com/a/ipc.ac.nz/document/d/1wn4PgrRJf3J7GmM0tbhoNCXFvE39cf8DmiY1S4eH7E8/edit?usp=sharing

References

Retrieved from:
http://clarkware.com/software/JDepend.html#uses (27/09/16)

Retrieved from:
http://www.zyxware.com/articles/4986/the-advantages-and-disadvantages-of-selenium-ide (21/09/16)

Retrieved from:
http://www.seleniumhq.org/docs/03_webdriver.jsp (21/09/16)

Retrieved from:
https://help.github.com/articles/about-repository-graphs/
(24/09/2016)