TESTING POLICY

WILDLIFE DRONES AND FLIGHT PLANS

Stakeholders

Matthew Evans
Reinhardt Eiselen
Bryan Janse van Vuuren
Andreas Louw
Deane Roos

Team DR BAM
drbam301@gmail.com

Contents

1. Introduction

Testing the software is critical to achieve a fully working system. The testing of the system ensures that the system as a whole works as expected. Individual functions and systems are also tested to be able to ensure that the components of the system work as expected and that if there is a fault it can be easily be identified. To achieve this Travis CI, Jest, Karma and Jasmine are used for testing.

Jest (server-side), Karma and Jasmine (client-side) will be used for the unit testing and integration testing and testing locally to ensure that the functions written are sufficient before the functions are merged with the rest of the system.

Travis will be used for testing the new functions with the system as a whole, where after the test have passed to make the new system live. Travis will also help with continuous testing as it builds and tests every time new code is pushed. Travis has been setup in such a way that it only allows error free code to be pushed to the master branch.

2. Description of test process

Unit tests are used to identify functions and smaller components that do not comply with the outcome of that function/ component. Each function that is used by another class or component should be tested to ensure that the function does function as predicted, these functions do not have to be tested extensively. Integration testing of different systems will be tested extensively since the integration of systems is crucial for the system as a whole to function.

Tests should be written as a function would be written so that the function can be tried and tested as soon as possible, and faults can be addressed. After the tests have been passed locally the code will be uploaded to the git repository, where it will be merged with the rest of the system. Travis will test the system as a whole, after all the Travis tests have passed the new functions will be made live.

All tests will be written in a .spec.ts file that accompanies the file it is testing - this should be in the same directory as the file being tested. (e.g. map.service.ts and map.service.spec.ts are in the same folder).

3. Test guidelines

Jest/Karma/Jasmine allow descriptive naming of tests (using sentences), rather than naming functions. This provides the ability to have more readable test descriptions.

Tests should be named with the following in mind:

1. What the function being tested does
2. What the outcome of the test should be

Examples of good test names:

1. Login should fail with incorrect credentials
2. Login should succeed with correct credentials
3. Authorization should fail when a token is not provided

These names all describe what is being tested, and what the conditions of the test are.

Tests should not call other tests. All tests should be self-contained units that do not call other unit tests.

Tests should keep in mind the state of the database if it is needed. Anything expected to be in the database should be manually added.

When creating example data for testing, it is better to write a large single object than it is to create functions that can do the same in less code. This makes it easy to see what kind of data is being passed to the test and evaluated when errors occur.

4. Test Evaluation

Tests will be evaluated depending on the function/subsystem importance functions/subsystems that are less important will only be tested a maximum of 3 times, while functions/subsystems of importance will be tested at least 5 times. Different tests that will fail and succeed will be written to ensure that with input that is not expected the system will not fail.

When refactoring code, all tests must be run.
If modifying a certain part of code, any tests that apply to code affected by changes must be re-run.
When dependencies are removed, all tests must be re-run (this is because dependencies may cause a chain reaction of errors).

5. Test Cases

Integration test cases

Drone route tests

https://github.com/cos301-2019-se/Wildlife-Drones-and-Flight-Plans/blob/development/server/src/controllers/drone-route.controller.spec.ts

Drone tests

https://github.com/cos301-2019-se/Wildlife-Drones-and-Flight-Plans/blob/development/server/src/controllers/drone.controller.spec.ts

Map tests

https://github.com/cos301-2019-se/Wildlife-Drones-and-Flight-Plans/blob/development/server/src/controllers/map.controller.spec.ts

Ranger tests

https://github.com/cos301-2019-se/Wildlife-Drones-and-Flight-Plans/blob/development/server/ranger.controller.spec.ts

Poaching incident tests

https://github.com/cos301-2019-se/Wildlife-Drones-and-Flight-Plans/blob/development/server/src/controllers/poaching-incident-controller.spec.ts

6. Test Reports

Direct link to test reports from Travis CI https://travis-ci.org/cos301-2019-se/Wildlife-Drones-and-Flight-Plans