University of Pretoria
Software Engineering - COS 301

Testing Policy

Singularity

Contents

1 Introduction

This document is meant as a guide for the team concerning how testing should be done on the project. This document contains the types of tests and procedure of testing that should be used throughout the project.

2 Why this guide ?

Testing is an important part of any successful project. It ensures that all core functionality as well as features are robust and work as expected. Due to testing being this important, it is essential that all members of the team test their work by following the policy of testing described in this guide.

3 The types of testing we use

Testing on our system id done continuously by running related unit tests and integration tests automatically when deploying our system. This ensures that we always have a working version that is deployed. This also shows any issues that still need to be solved, and allows us to quickly correct the mistake and prevent any other mistakes in other modules that rely on the faulty module.

3.1 Unit testing

Our system is designed to be very modular and flexible. Every module has to be tested in order to ensure that it's basic functionality works as expected. Unit tests is designed to do exactly this, ensure that every module (unit) does all that is expected of it. This testing involves testing individual classes as well as all critical (preferably all) functions.

3.2 Integration testing

Each of the modules in our system will eventually work together and communicate with one another in some way. In order to ensure that modules are well integrated and can function together, we do integration tests. This testing is done after all individual modules pass their respective unit tests.

4 What we expect our developers to cover with their testing

We expect each and every developer of our team to test the following:

- Every function should have tests that ensure the logic of the function is correct.
- All code should be tested with all possible conditions. This includes all logical paths of execution.
- All code should in some way be tested to ensure that no code is untested after passing all tests.

5 Testing frameworks

Our project is mostly implemented in TypeScript and Python. In order to keep all tests uniform and be able to integrate the tests with the continuous integration software we use (Travis CI), we have decided on using only two testing frameworks to do our testing.

5.1 jasmine-node

Jasmine-node is a solid testing framework that gives us the ability to test all TypeScript code without first compiling the code to JavaScript. It also provides the benefit of reasonably easily being able to test asynchronous functions. We require that this framework is used for the testing of all TypeScript code.

5.2 unittest (Python)

The unittest framework for Python is easy to use and is able to handle all our unit testing needs for the facial recognition testing. We require that this framework is used for the testing of all code written in Python.

6 When to write tests

Unit tests should preferably be written before any specific unit of code is implemented, with the tests designed to test all required functionality of that unit. So in other words you must write tests for all your code. All descriptions of the tests must be clear and indicate exactly what is being tested. Integration tests must be created before or directly after the integration of two subsystems.

7 Where to keep all your tests

7.1 jasmine-node

All of your test files should be stored in the 'spec' folder of the project. Spec files should be saved as eg. "anyName-spec.ts", where 'anyName' is the actual filename.

7.2 unittest

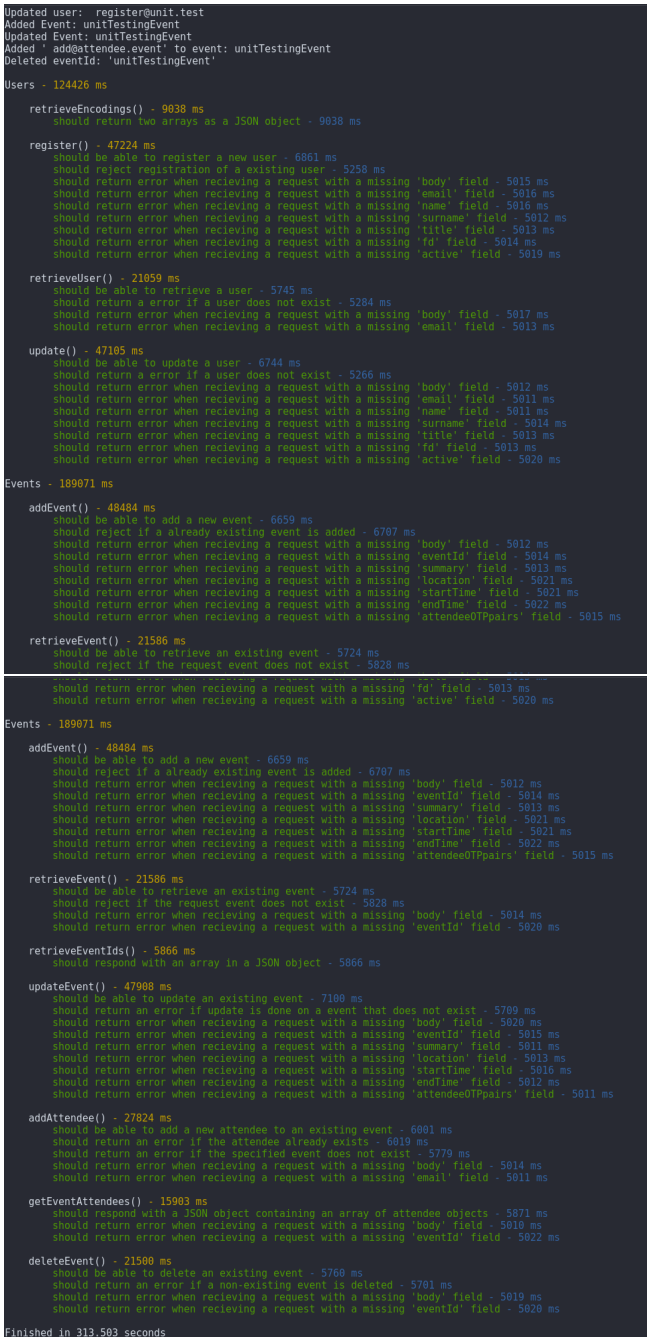
All python unit tests should be stored along with the relevant code the unit test is testing. The filenames of the unit test files should be in the format of "anyNameUnitTests.py" where 'anyName' represent the actual filename.

8 Continuous Integration Using TravisCI

All unit test and integration testing scripts and files should be added to the relevant ".travis.yml" file as soon as your tests are implemented. This is important to ensure that the system is working as intended and that error are revealed and fixed as soon as possible. Our project on TravisCI can be found at https://travis-ci.org/cos301-2019-se/A-Recognition

9 Test outcome examples

9.1 jasmin-node output



```
Updated user:  register@unit.test
Added Event: unitTestingEvent
Updated Event: unitTestingEvent
Added ' add@attendee.event' to event: unitTestingEvent
Deleted eventId: 'unitTestingEvent'

Users - 124426 ms

    retrieveEncodings() - 9038 ms
        should return two arrays as a JSON object - 9038 ms

    register() - 47224 ms
        should be able to register a new user - 6861 ms
        should reject registration of a existing user - 5258 ms
        should return error when recieving a request with a missing 'body' field - 5015 ms
        should return error when recieving a request with a missing 'email' field - 5016 ms
        should return error when recieving a request with a missing 'name' field - 5016 ms
        should return error when recieving a request with a missing 'surname' field - 5012 ms
        should return error when recieving a request with a missing 'title' field - 5013 ms
        should return error when recieving a request with a missing 'fd' field - 5014 ms
        should return error when recieving a request with a missing 'active' field - 5019 ms

    retrieveUser() - 21059 ms
        should be able to retrieve a user - 5745 ms
        should return a error if a user does not exist - 5284 ms
        should return error when recieving a request with a missing 'body' field - 5017 ms
        should return error when recieving a request with a missing 'email' field - 5013 ms

    update() - 47105 ms
        should be able to update a user - 6744 ms
        should return a error if a user does not exist - 5266 ms
        should return error when recieving a request with a missing 'body' field - 5012 ms
        should return error when recieving a request with a missing 'email' field - 5011 ms
        should return error when recieving a request with a missing 'name' field - 5011 ms
        should return error when recieving a request with a missing 'surname' field - 5014 ms
        should return error when recieving a request with a missing 'title' field - 5013 ms
        should return error when recieving a request with a missing 'fd' field - 5013 ms
        should return error when recieving a request with a missing 'active' field - 5020 ms

Events - 189071 ms

    addEvent() - 48484 ms
        should be able to add a new event - 6659 ms
        should reject if a already existing event is added - 6707 ms
        should return error when recieving a request with a missing 'body' field - 5012 ms
        should return error when recieving a request with a missing 'eventId' field - 5014 ms
        should return error when recieving a request with a missing 'summary' field - 5013 ms
        should return error when recieving a request with a missing 'location' field - 5021 ms
        should return error when recieving a request with a missing 'startTime' field - 5021 ms
        should return error when recieving a request with a missing 'endTime' field - 5022 ms
        should return error when recieving a request with a missing 'attendeeOTPpairs' field - 5015 ms

    retrieveEvent() - 21586 ms
        should be able to retrieve an existing event - 5724 ms
        should reject if the request event does not exist - 5828 ms
        should return error when recieving a request with a missing 'body' field - 5014 ms
        should return error when recieving a request with a missing 'eventId' field - 5020 ms

    retrieveEventIds() - 5866 ms
        should respond with an array in a JSON object - 5866 ms

    updateEvent() - 47908 ms
        should be able to update an existing event - 7100 ms
        should return an error if update is done on a event that does not exist - 5709 ms
        should return error when recieving a request with a missing 'body' field - 5020 ms
        should return error when recieving a request with a missing 'eventId' field - 5015 ms
        should return error when recieving a request with a missing 'summary' field - 5011 ms
        should return error when recieving a request with a missing 'location' field - 5013 ms
        should return error when recieving a request with a missing 'startTime' field - 5016 ms
        should return error when recieving a request with a missing 'endTime' field - 5012 ms
        should return error when recieving a request with a missing 'attendeeOTPpairs' field - 5011 ms

    addAttendee() - 27824 ms
        should be able to add a new attendee to an existing event - 6001 ms
        should return an error if the attendee already exists - 6019 ms
        should return an error if the specified event does not exist - 5779 ms
        should return error when recieving a request with a missing 'body' field - 5014 ms
        should return error when recieving a request with a missing 'email' field - 5011 ms

    getEventAttendees() - 15903 ms
        should respond with a JSON object containing an array of attendee objects - 5871 ms
        should return error when recieving a request with a missing 'body' field - 5010 ms
        should return error when recieving a request with a missing 'eventId' field - 5022 ms

    deleteEvent() - 21500 ms
        should be able to delete an existing event - 5760 ms
        should return an error if a non-existing event is deleted - 5701 ms
        should return error when recieving a request with a missing 'body' field - 5019 ms
        should return error when recieving a request with a missing 'eventId' field - 5020 ms

Finished in 313.503 seconds
```