
# Test Plan and Report

---

MarketLead.io
Client: RetroRabbit

---

Team: Valknut Solutions

Version: 1.4


- 13054903 - Charl Jansen van Vuuren
- 13044924 - Kevin Heritage
- 13176545 - Quinton Weenink

Department of Computer Science


October 22, 2016

# Revision History

| Revision | Date | Author(s) | Description |
|---|---|---|---|
| 1.0 | 27.7.2016 | CJvV,KH,QW | created |
| 1.1 | 04.9.2016 | CJvV,KH,QW | Updated format based on template on CS web, added new test information |
| 1.2 | 27.9.2016 | CJvV,KH,QW | Added to conclusion, testing in general and test summary, updated heading to new product name |
| 1.3 | 30.9.2016 | CJvV,KH,QW | Added Facebook messenger unit tests, updated testing tables, added detailed overview of new tests, updated paths to tests |
| 1.4 | 05.11.2016 | CJvV,KH,QW | Added Logo, added Validation tests to all necessary sections, added new figures for messenger and validation tests, added Authentication test with figure |

# Contents

# 1 Introduction

This document contains information related to Testing of the Insurance Profiling project that is being developed for RetroRabbit as part of the COS301 module at the University Of Pretoria.
This document is based on the IEEE 829 Standard for Testing Documentation IEEE 829 The structure of this document includes an initial scope and purpose, a Unit test plan and the final Unit test report.

## 1.1 Purpose

This document combines the unit test plan and report into a single coherent artefact. The overall purpose of the system is to allow the analysis of different social media inputs (Facebook in particular) by means of a marketing and risk analysis standpoint. Test driven development is crucial to our system as it ensures every added feature is compatible with the current version of the system and that feature is working as intended.

## 1.2 Scope

The scope of this document is structured as follows. The features that are considered for testing are listed as per section 2.
Tests that have been identified from the requirements are discussed in detail in section 3.
Furthermore, this document outlines the test environment and the risks involved in the testing approaches that will be followed as per section 1.3. Assumptions and dependencies of this test plan will also be mentioned as per section 1.4 . Section 3 , 5 and 8 outlines, discusses and concludes on the results of the tests, respectively.

## 1.3 Test Environment

The testing environment is mainly software based and includes:

- Programming languages:
  - Javascript
- Testing Frameworks:
  - MochaJS - Unit tests
  - Travis CI - Continuous integration for deployment
- Coding Environment:
  - NodeJS based web server
  - ExpressJS
  - PostgreSQL database with Sequelize ORM
  - Terminal based test outputs
- Operating system:
  - Ubuntu Linux
  - Microsoft Windows 10
- Internet Browsers:
  - Mozilla Firefox
  - Chrome web browser

### 1.4 Assumptions and Dependencies

Assumptions:

- An instance of the server is running locally(for Unit tests)
- External server is active (for Deploy tests only)
- Active internet connection (for Deploy tests only)
- NodeJS and Node package manager is installed works as intended
- Mocha is installed and works as intended

Dependencies:

- Travis CI
- Mocha framework
- Chai (part of Mocha)
- Heroku platform as a service (Integration tests)
- Assert package from npm
- validatationTest.js
- authTest.js
- fbController.js test file
- graphTest.js test file
- test.js test file

## 2 Test items

### 2.1 The items included in this test document:

- Backend API calls
- Functional unit tests
- Database management functions
- Environmental tests
- Deployment and compatibility tests

# 3 Functional Features to be Tested

#### 3.0.0.1 Test estimation can be categorized as fast (nearly instant tests) or slow (timely tests with external dependencies) for our system.

## 3.1 Overall approach with regards to adequate testing of :

- Feature unit tests:
  - Mocking out the non essential dependencies to be able to test the feature
  - Include the essential dependencies that cannot be mocked out
  - Estimation of this feature group can be concluded as "fast".
- Backend API tests:
  - Estimation of this feature group can be concluded as "slow" as it is dependent on the external server.
- Environmental tests:
  - Estimation of this feature group can be concluded as "fast" as it is a simple check for current environment.
- Deployment tests:
  - Estimation of this feature group can be concluded as "slow" as it is dependent on the external Travis CI server.

## 3.2 The included features to be tested:

- Feature unit tests:
  - The ability to receive data from a Facebook lead ad and save this data in the database.
  - The ability to receive data from any other integration point as per the specified format.
  - The ability to receive data from the Facebook messenger integration point.
  - These feature relates to the getLeadData use case in the Functional requirements documentation
  - The calculation of Age from a datestring object
  - The calculation of Months from a datestring object
  - Validation test for user input from all integration points
  - Authentication test to verify an analyst's identity
- API calls to do:
  - Database insertions
  - Database deletions
  - Database retrievals
  - This test relates to the Analysis use case in the Functional requirements documentation, it forms the underlying persistence with regards to the data analysis is done on.
- Environmental test
  - Integration test to determine the current environment (development or deployment) to react accordingly

- Deployment/Build testing including Compatibility
  - Automated testing by the Travis CI framework to ensure the current feature is compatible with different versions of the NodeJS framework.

Table 2: Test cases

| Feature ID | RDS Source | Summary | Test Case ID |
|---|---|---|---|
| 1 | Functional requirements - getLeadData | Receive data from a Facebook lead ad | 1 |
| 2 | Functional requirements - getData | Receive data from integration source | 2 |
| 3 | Functional requirements - Analysis | Database insertions | 3 |
| 4 | Functional requirements - Analysis | Database deletions | 4 |
| 5 | Functional requirements - Analysis | Database retrievals | 5 |
| 6 | Functional requirements - generateReport | Extract age from datestring | 6 |
| 7 | Functional requirements - generateReport | Extract month from datestring | 7 |
| 8 | Architecture design - Section 5 | Environmental test | 8 |
| 9 | Architecture design - Section 5.1.1 | Deployment test | 9 |
| 10 | Functional requirements - getMessengderData | Receive data from messenger | 10 |
| 11 | Functional requirements - getData | Validate input data | 11 |
| 12 | Architectural requirements - Security/Auditability | Authentication testing | 12 |

# 4 Test instructions

- Unit Tests can be run with the command "npm test" which will run Mocha.
- API tests can be run with the command "npm run-script apitest"
- Integration tests can be run with the command "npm run-script integrationtest".
- If these tests are ran locally they will connect to the local database.
- Travis CI runs these test to the external database hosted on Heroku. The Travis tests are ran when a new branch is merged to ensure compatibility with all versions of Node.

# 5 Test cases

## 5.1 Test Case 1: getLeadData

### 5.1.1 Condition 1: Ad is submitted with all valid fields

#### 5.1.1.1 Objective:

The purpose of this test is to extract data from a filled in Facebook lead ad and save this data if valid.

#### 5.1.1.2 Input:

The following inputs will be used:

1. An non-existing entity - based on Facebook user data:
   (a) firstName
   (b) lastName
   (c) mobileNumber
   (d) maritalStatus
   (e) dateOfBirth
   (f) gender
   (g) location
   (h) email

#### 5.1.1.3 Outcome:

The following outcomes are expected for a pass result:

1. A JSON object of a User passed to the Condition 3

### 5.1.2 Condition 2: Invalid inputs

#### 5.1.2.1 Objective:

The purpose of this test is to return an empty user if the inputs are not valid

#### 5.1.2.2 Input:

The following inputs will be used:

1. A unpopulated user object

#### 5.1.2.3 Outcome:

The following outcomes are expected for a pass result:

1. Returns an empty user

### 5.1.3 Condition 3: Valid data inputs

#### 5.1.3.1 Objective:

The purpose of this test is to return a valid JSON object of the data for saving purposes.

#### 5.1.3.2 Input:

The following inputs will be used:

1. An existing object of User as per Condition 1

#### 5.1.3.3 Outcome:

The following outcomes are expected for a pass result:

1. A new User passed to the database

## 5.2 Test Case 2: getData

### 5.2.1 Condition 1: Valid inputs

#### 5.2.1.1 Objective:

The purpose of this test is to extract data from any integration source.

#### 5.2.1.2 Input:

The following inputs will be used:

1. A non-existing entity - based on user data from any integration:
   (a) firstName
   (b) lastName
   (c) mobileNumber
   (d) maritalStatus
   (e) dateOfBirth
   (f) gender
   (g) location
   (h) email

#### 5.2.1.3 Outcome:

The following outcomes are expected for a pass result:

1. A JSON object of a User

## 5.3 Test Case 3: Insertion

### 5.3.1 Condition 1: Valid inputs

#### 5.3.1.1 Objective:

The purpose of this test is to test insertion of data into database.

#### 5.3.1.2 Input:

The following inputs will be used:

1. An existing entity of User object

#### 5.3.1.3 Outcome:

The following outcomes are expected for a pass result:

1. The data will be saved into the database

## 5.4 Test Case 4: Deletion

### 5.4.1 Condition 1: Valid field to remove

#### 5.4.1.1 Objective:

The purpose of this test is to test deletion of data in the database

#### 5.4.1.2 Input:

The following inputs will be used:

1. An existing entity of User in the database

#### 5.4.1.3 Outcome:

The following outcomes are expected for a pass result:

1. The existing entity gets deleted from the database

## 5.5 Test Case 5: Retrieval

### 5.5.1 Condition 1: If field exists

#### 5.5.1.1 Objective:

The purpose of this test is to test retrieval of data from the database.

#### 5.5.1.2 Input:

The following inputs will be used:

1. An existing entity

#### 5.5.1.3 Outcome:

The following outcomes are expected for a pass result:

1. The existing entity gets returned from the database

### 5.5.2 Condition 2: If field does not exists

#### 5.5.2.1 Objective:

The purpose of this test is to test retrieval of data from the database.

#### 5.5.2.2 Input:

The following inputs will be used:

1. A non existing entity

#### 5.5.2.3 Outcome:

The following outcomes are expected for a pass result:

1. The test will return null

## 5.6 Test Case 6: Extract age from datestring

### 5.6.1 Condition 1: Valid datestring object

#### 5.6.1.1 Objective:

The purpose of this test is to retrieve a valid age from a datestring.

#### 5.6.1.2 Input:

The following inputs will be used:

1. "1994-06-05T22:00:00.000Z"

#### 5.6.1.3 Outcome:

The following outcomes are expected for a pass result:

1. A valid age will be returned

### 5.6.2 Condition 2: Correct age is returned

#### 5.6.2.1 Objective:

The purpose of this test is calculate the age from a datestring in the format "1994-06-05T22:00:00.000Z".

#### 5.6.2.2 Input:

The following inputs will be used:

1. "1994-06-05T22:00:00.000Z"

#### 5.6.2.3 Outcome:

The following outcomes are expected for a pass result:

1. The age of 22 is returned as a valid outcome

### 5.6.3 Condition 3: Correct age is returned

#### 5.6.3.1 Objective:

The purpose of this test is calculate the age from a datestring in the format "1998-06-05T22:00:00.000Z".

#### 5.6.3.2 Input:

The following inputs will be used:

1. "1998-06-05T22:00:00.000Z"

#### 5.6.3.3 Outcome:

The following outcomes are expected for a pass result:

1. The age of 18 is returned as a valid outcome

## 5.7 Test Case 7: Extract month from datestring as named month

### 5.7.1 Condition 1: Valid datestring object

#### 5.7.1.1 Objective:

The purpose of this test is to retrieve a valid month from a datestring.

#### 5.7.1.2 Input:

The following inputs will be used:

1. "1994-09-05T22:00:00.000Z"

#### 5.7.1.3 Outcome:

The following outcomes are expected for a pass result:

1. A valid month will be returned as a named month

### 5.7.2 Condition 2: Correct age is returned

#### 5.7.2.1 Objective:

The purpose of this test is calculate the month from a datestring in the format "1994-06-05T22:00:00.000Z".

#### 5.7.2.2 Input:

The following inputs will be used:

1. "1994-06-05T22:00:00.000Z"

#### 5.7.2.3 Outcome:

The following outcomes are expected for a pass result:

1. The string of the month of June is returned

### 5.7.3 Condition 3: Correct age is returned

#### 5.7.3.1 Objective:

The purpose of this test is calculate the month from a datestring in the format "1998-09-05T22:00:00.000Z".

#### 5.7.3.2 Input:

The following inputs will be used:

1. "1998-09-05T22:00:00.000Z"

#### 5.7.3.3 Outcome:

The following outcomes are expected for a pass result:

1. The string of the month of September is returned

## 5.8 Test Case 8: Environmental test(Integration test)

### 5.8.1 Condition 1: If environment is set to Deployment

#### 5.8.1.1 Objective:

The purpose of this test is to set the Environment variable based on the current state of the system, Deployment in this case.

#### 5.8.1.2 Input:

The following inputs will be used:

1. Environment set to deployment

#### 5.8.1.3 Outcome:

The following outcomes are expected for a pass result:

1. Set the hosting port to 80

### 5.8.2 Condition 2: If environment is set to Development

#### 5.8.2.1 Objective:

The purpose of this test is to set the Environment variable based on the current state of the system, Development in this case.

#### 5.8.2.2 Input:

The following inputs will be used:

1. Environment set to development

#### 5.8.2.3 Outcome:

The following outcomes are expected for a pass result:

1. Set the hosting port to 3000 to avoid conflicts

## 5.9 Test Case 9: Travis CI Deployment test (Integration test)

### 5.9.1 Condition 1: If hosting version is compatible

#### 5.9.1.1 Objective:

The purpose of this test is to ensure the current deployed version is compatible with the external hosting service. This includes dependency versions.

#### 5.9.1.2 Input:

The following inputs will be used:

1. A valid NodeJS and NPM version is used
2. Mocha tests pass

#### 5.9.1.3 Outcome:

The following outcomes are expected for a pass result:

1. The build will pass

### 5.9.2 Condition 2: If hosting version is incompatible

#### 5.9.2.1 Objective:

The purpose of this test is to ensure the current deployed version is compatible with the external hosting service. This includes dependency versions.

#### 5.9.2.2 Input:

The following inputs will be used:

1. An unsupported version of NodeJS or NPM is used
2. Mocha Tests fails

#### 5.9.2.3 Outcome:

The following outcomes are expected for a pass result:

1. The build will error/fail

## 5.10 Test Case 10: Facebook messenger test

### 5.10.1 Condition 1: Valid user message

#### 5.10.1.1 Objective:

The purpose of this test is to ensure the Facebook messenger integration data is extracted from the user message.

#### 5.10.1.2 Input:

The following inputs will be used:

1. An empty user to be populated
    - messageId = 0
    - firstname = ”
    - lastname : ”
    - phonenumber : ”
    - maritalstatus : ”
    - dateofbirth : ”
    - gender : ”
    - city : ”
    - email : ”

#### 5.10.1.3 Outcome:

The following outcomes are expected for a pass result:

1. A populated user with the values
    - messageId : 1
    - firstname : ’Quinton’
    - lastname : ’Weenink’
    - phonenumber : ’071 555 9858’
    - maritalstatus : ’Married’
    - dateofbirth : ’08/03/1995’
    - gender : ’male’
    - city : ’Cape Town’
    - email : ’fake@gmail.com’

### 5.10.2 Condition 2: Valid response

#### 5.10.2.1 Objective:

The purpose of this test is to ensure the correct response is given for a received message.

#### 5.10.2.2 Input:

The following inputs will be used:

1. A correct response to the given message.
    - messageId : 1
    - firstname : 'Quinton'
    - lastname : 'Weenink'
    - phonenumber : '071 555 9858'
    - maritalstatus : 'Married'
    - dateofbirth : '08/03/1995'
    - gender : 'male'
    - city : 'Cape Town'
    - email : 'fake@gmail.com'

#### 5.10.2.3 Outcome:

The following outcomes are expected for a pass result:

1. 8:'You will be contacted shortly...',
2. 0:'Please reply with your Name:',
3. 1:'Please reply with your Last name:',
4. 2:'Please reply with your Phone number:',
5. 3:'Please reply with your Marital status:',
6. 4:'Please reply with your Date of birth:',
7. 5:'Please reply with your Gender:',
8. 6:'Please reply with your City:',
9. 7:'Please reply with your Email:'

## 5.11 Test Case 11: Validation Test for user input

### 5.11.1 Condition 1: Valid user input

#### 5.11.1.1 Objective:

The purpose of this test is to ensure the input data is correct and valid. Input data can be from any integration source.

#### 5.11.1.2 Input:

The following inputs will be used:

1. A empty user with a valid input field :
    - valid firstname
    - valid lastname
    - valid phonenumber
    - valid maritalstatus
    - valid dateofbirth
    - valid gender
    - valid city
    - valid email

#### 5.11.1.3 Outcome:

The following outcomes are expected for a pass result:

1. A valid user with :
    - A success object is returned per item with a true flag.
    - firstname = 'Quinton'
    - lastname = 'Weenink'
    - phonenumber = '0718596641'
    - maritalstatus = 'married'
    - dateofbirth = '2-2-1994'
    - gender = 'female'
    - city = 'PE'
    - email = 'quinton@gmail.com'

### 5.11.2 Condition 2: invalid user input

#### 5.11.2.1 Objective:

The purpose of this test is to ensure that incorrect input returns the correct error message. Input data can be from any integration source.

#### 5.11.2.2 Input:

The following inputs will be used:

1. A empty user with an invalid input field :
    - invalid firstname
    - invalid lastname
    - invalid phonenumber
    - invalid maritalstatus
    - invalid dateofbirth
    - invalid gender
    - invalid city
    - invalid email

#### 5.11.2.3 Outcome:

The following outcomes are expected for a pass result:

1. A success object is returned per item with a false flag.
2. An error message is returned

## 5.12 Test Case 12: Authentication Test

### 5.12.1 Condition 1: invalid token provided

#### 5.12.1.1 Objective:

The purpose of this test is to ensure that only the authorized users gain access to certain key functions. This test will reject a wrong token.

#### 5.12.1.2 Input:

The following inputs will be used:

1. Invalid authentication token

#### 5.12.1.3 Outcome:

The following outcomes are expected for a pass result:

1. An error message stating "Invalid token"
2. No access.

### 5.12.2 Condition 2: valid token provided

#### 5.12.2.1 Objective:

The purpose of this test is to ensure that only the authorized users gain access to certain key functions. This test will accept a correct token.

#### 5.12.2.2 Input:

The following inputs will be used:

1. Valid authentication token

#### 5.12.2.3 Outcome:

The following outcomes are expected for a pass result:

1. Access is granted

# 6 Item Pass/Fail criteria

If the specified Pre-conditions are not met the post condition will result as false and the test will fail.
If the specified Pre-conditions are met the post-condition will hold true and the test will pass.

- Extract age from datestring
  - Condition 1:
    - Pre-condition - Valid input datestring object
    - Post-condition - Valid object is returned
  - Condition 2 and 3:
    - Pre-condition - Valid input datestring object
    - Post-condition - Correct age is returned
- Extract month from datestring
  - Condition 1:
    - Pre-condition - Valid input datestring object
    - Post-condition - Valid object is returned
  - Condition 2 and 3:
    - Pre-condition - Valid input datestring object
    - Post-condition - Correct month is returned
- Creating row in Database
  - Pre-conditions:
    - Database is connected and created
    - The following values are specified:
      - firstName
      - lastName
      - mobileNumber
      - maritalStatus
      - dateOfBirth
      - gender
      - location
      - email
  - Post-condition - The row gets persisted
- Retrieving row in Database
  - Pre-condition - The currentUser exists
  - Post-condition - The currentUser is retrieved
- Removing row in Database based on currentUser
  - Pre-condition - The currentUser exists
  - Post-condition - The currentUser is removed
- Valid messenger responses for valid inputs
  - Pre-condition - The inputs as specified are valid

  - Post-condition - The output messages are correct based on the inputs.

- Validation of user inputs
  - Condition 1:
    * Pre-condition - The inputs as specified are valid
    * Post-condition - A success = true message
    * Post-condition - A valid user is created
  - Condition 2:
    * Pre-condition - The inputs as specified are invalid
    * Post-condition - A success = false message
    * Post-condition - Error is returned

- Authentication of user
  - Condition 1:
    * Pre-condition - An invalid access token is provided
    * Post-condition - No access is granted.
    * Post-condition - Error message
  - Condition 2:
    * Pre-condition - A valid access token is provided
    * Post-condition - Access is granted.

- Travis CI build test for Node version 5.11
  - Pre-condition - The changes to the branch are compatible with version 5.11
  - Post-condition - The build will pass successfully

- Travis CI build test for Node version 6.2
  - Pre-condition - The changes to the branch are compatible with version 6.2
  - Post-condition - The build will pass successfully

# 7 Test Deliverables

Artefacts to be produced as part of testing include the following:

- Test Plan Section 2 - Section 6
- Test Report Section 8
- Test code Github Link to testing directory

# 8 Detailed Test Results

## 8.1 Overview of Test Results

All tests were run multiple times to ensure success with the inclusion of the Travis Continuous integration tests to ensure compatibility. Two of the tests failed as a result of the external server not being active at that specific time, these cases form part of integration testing and the normal unit tests make use of mocked out information.

The Mocha testing framework was chosen as it integrates well with the Node Package manager system and simplifies the creation of writing a new unit test.
Travis CI was chosen as integration testing framework to ensure compatibility and deployment capabilities. Travis CI integrates with Github to run tests on all branch merges to ensure compatibility of new features in the main deployment branch.

Travis CI runs automated tests per branch merge without the need for developer assistance, this greatly decreases deployment time and ensures compatibility.

## 8.2 Functional Requirements Test results

| Test ID | Test Name | Github Link to File |
|---|---|---|
| 1 | getLeadData | fbControllerTest.js |
| 2 | getData | fbControllerTest.js |
| 3,4,5 | API tests/Database functions | test.js |
| 6 | getAge | graphTest.js |
| 7 | getMonths | graphTest.js |
| 8 | Environmental test | test.js Line 9-15 |
| 9 | Compatibility test | Travis CI build nr 130 |
| 10 | addToUser()/ getMessage() | fbMessengerTest.js |
| 11 | validate()/ objectValidate() | validationTest.js |
| 12 | authenticate()/ generateToken() | authTest.js |

Table 3: Links to test files

## 8.3 Test summary

#### 8.3.0.1 All tests are located in the Github repo's testing directory:Repo

#### 8.3.0.2 The following results were obtained from the tests conducted.

### 8.3.1 Test Case 1 - TC 5.1

Test case 1 ran with valid inputs. The data extraction functionality for Facebook works as intended.

1. All pre-conditions were met and as a result the post condition remained true
2. Valid inputs were passed as parameter

**Result** : Pass

### 8.3.2 Test Case 2 - TC 5.2

Test case 2 ran with valid inputs. The data extraction functionality for any integration works as intended.

1. All pre-conditions were met and as a result the post condition remained true
2. Valid inputs were passed as parameter

**Result** : Pass

### 8.3.3 Test Case 3 - TC 5.3

Test case 3 created a new user in the database. The API for insertion works as intended.

1. All pre-conditions were met and as a result the post condition remained true
2. Server was live to accept new connections and database functions.

**Result** : Pass

### 8.3.4 Test Case 4 - TC 5.4

Test case 4 deleted the new user in the database. The API for deletion works as intended.

1. All pre-conditions were met and as a result the post condition remained true
2. Server was live to accept new connections and database functions.

**Result** : Pass

### 8.3.5 Test Case 5 - TC 5.5

Test case 5 retrieved a user in the database. The API for retrieval works as intended.

1. All pre-conditions were met and as a result the post condition remained true
2. Server was live to accept new connections and database functions.

**Result** : Pass

#### 8.3.6 Test Case 6 - TC 5.6

Test case 6 returned the age of 22 and 18. The getAge functionality works as intended.

1. All pre-conditions were met and as a result the post condition remained true
2. Correct inputs were passed as parameter
3. Correct outputs were returned

**Result** : Pass

#### 8.3.7 Test Case 7 - TC 5.7

Test case 7 returned the month of June and September. The getMonths functionality works as intended.

1. All pre-conditions were met and as a result the post condition remained true
2. Valid inputs were passed as parameter
3. Correct inputs were passed as parameter
4. Correct outputs were returned

**Result** : Pass

#### 8.3.8 Test Case 8 - TC 5.8

The local environment is set to development as a result of the environment test.

1. Environment returned as development for local

**Result** : Pass

#### 8.3.9 Test Case 9 - TC 5.9

The build for number 130 passed in Travis CI's log

1. The current packages and node version were compatible with the external Heroku server.
2. The build completed without errors.

**Result** : Pass

#### 8.3.10 Test Case 10 - TC 5.10

Test case 10 returned a populated user and sent the correct responses based on the inputs. The Facebook messenger tests work as intended.

1. All pre-conditions were met and as a result the post condition remained true
2. Valid inputs were populated.
3. Correct inputs were passed as parameter
4. Correct messages were returned as outputs.

**Result** : Pass

#### 8.3.11 Test Case 11 - TC 5.11

Test case 11 returned a success = true message for the valid condition and a success = false for the invalid condition. The validation controller works as intended.

1. All pre-conditions were met and as a result the post condition remained true
2. Valid inputs were checked.
3. Invalid inputs were detected.

**Result** : Pass

#### 8.3.12 Test Case 12 - TC 5.12

Test case 12 returned a valid access granted for a valid token and Access denied with an error message for the invalid token. The authentication controller works as intended.

1. All pre-conditions were met and as a result the post condition remained true
2. Authentication was denied and granted based on the correct tokens

**Result** : Pass

### 8.4 Test disussion

1. The mapping of contracts onto tests increases the testability of a system. Mocking out objects eliminates external dependencies which greatly benefit testability. A form of mocking can be seen in Test 5.1 and Test 5.12
2. As with the continuous integration model we ensure that the system always stay in a deployable state by means of Travis CI tests and environmental tests. Continuous integration further ensures total compatibility of all new features in each environment (deployment or development).

# 9 Other

## 9.1 Mocha build screenshots

```
> mocha test/fbControllerTest.js



  Facebook Controller
    extractUserTest
      ✓ should return json object of a user that can be added to the database
      ✓ should return empty user


  2 passing (8ms)
```

Figure 1: getLeadData unit test

```
> mocha test/graphTest



  Graph Date Test
    getAge()
      √ Returns a valid age from a datestring
      √ Returns the age 22 from a datestring 1994-06-05T22:00:00.000Z
      √ Returns the age 18 from a datestring 1998-06-05T22:00:00.000Z
    getMonths()
      √ Returns a valid month from a datestring
      √ Returns June from the datestring 1994-06-05T22:00:00.000Z
      √ Returns September from the datestring 1994-09-05T22:00:00.000Z


  6 passing (6ms)
```

Figure 2: getAge and getMonth unit test

```
Facebook Messenger Test
  addToUser()
    √ Returns a valid empty user
    √ Places the first name in the user
    √ Places the last name in the user
    √ Places the phone number in the user
    √ Places the marital status in the user
    √ Places the date of birth in the user
    √ Places the gender in the user
    √ Places the city in the user
    √ Places the email in the user
    √ Tests concurrent users
  getMessage()
    √ Returns the default message if the user is null
    √ Constructs name message
    √ Constructs last name message
    √ Constructs phone message
    √ Constructs marital message
    √ Constructs date of birth message
    √ Constructs gender message
    √ Constructs city message
    √ Constructs email message
  getMessage()
    √ Returns the default message if the user is null
```

Figure 3: Messenger input Test

```
Validation Testing
  validate()
✓ Test of logic not really a unit test
✓ isName test with failing first_name
✓ isName test with passing first_name
✓ isName test with failing last_name
✓ isName test with passing last_name
✓ isName test with passing last_name
✓ isPhoneNumber test with failing phone_number
✓ isPhoneNumber test with passing phone_number
✓ isPhoneNumber test with passing phone_number
✓ isMarital test with passing marital_status
✓ isMarital test with passing marital_status
✓ isDate test with failing date_of_birth
✓ isDate test with passing date_of_birth
✓ isGender test with failing gender
✓ isGender test with passing gender
✓ isName test with failing City
✓ isName test with passing City
✓ isEmail test with failing email
✓ isEmail test with passing email
  objectValidate()
✓ Validate passing user
✓ Test of logic not really a unit test
```

Figure 4: Validation Test

```
Authentication Test
  authenticate()
✓ Does not enter the bad zone if no token provided
✓ Does not enter the bad zone if bad token is provided
✓ Does enter the bad zone if the correct token is provided
  generateToken()
✓ If no user it returns bad request
✓ If user exists generate token
```

Figure 5: Authentication Test

```
> mocha


  Testing API
    URL: localhost/api/user/
      POST : /api/user/
        ✓ should insert and return the inserted user (209ms)
      GET : /api/user/:currentUser
        ✓ should get user with id
      DELETE : /api/user/:currentUser
        ✓ should delete the just inserted user


  3 passing (272ms)
```

Figure 6: Api Tests

## 9.2 Travis build

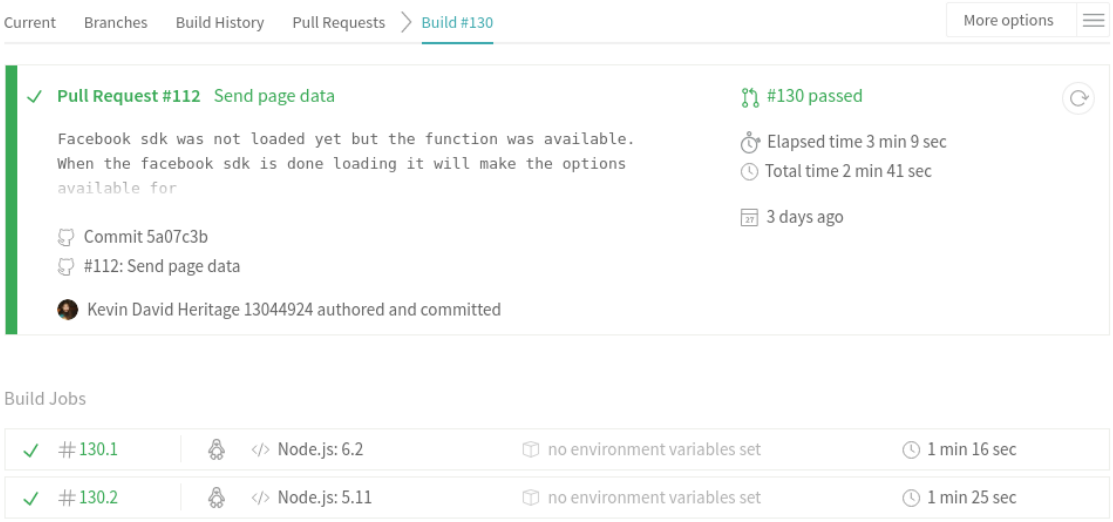


Figure 7: Travis success

# 10 Conclusions

## 10.1 General testing conclusions

- This project follows a Continous Integration model with regards to testing. Any new feature to be merged with the main branch is first tested, then reviewed by the person merging the branch and only then the merge is closed.
- The amount of tests are limited in the current version of the system, this is as a result of the complexity of testing external API calls, such as Facebook, on which our system heavily relies.
- The ability to modularize functions in NodeJS also complicates the ability to test functionality, this is as a result of the asynchronous callback structure NodeJS uses.
- The integration tests will ensure these tests always pass before deploying a new version of the system.

## 10.2 Test Summary

- All the unit tests completed successfully. Once a test fulfills its pre-condition(s), that test will pass as intended, fulfilling the post-condition.
- Test 1 through 12 all passed as intended, as a result of these tests fulfilling their pre-conditions

# 11 Benefits of Testing

Unit tests in general increases the reliability of a system. Once a new feature is added, that feature must pass its intended tests before even considering integration.

This test-driven development method ensures compatibility of new features (integration tests) and that the new feature acts as intended (unit tests).
Testing ensures full functionality, reliability and helps other team members understand the added feature.
The Mocha testing frameworks displays test results a easily understandable way.