UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Department of Computer Science

Unit Test Plan and Report

---

# Client: Gavin Potgieter

---

## Team: CodeBlox


Lethabo Mogase (Bsc: Computer Science)

Lorenzo Spazzoli (Bsc: Computer Science)

Bilal Muhammad (BIS: Multimedia)

Dirk de Klerk (BIS: Multimedia)


October 22, 2016

# Contents

# 1 Introduction

## 1.1 Purpose for Test

### 1.1.1 Reducing bugs in new features

We write new tests as we write new code. We believe that tests do not result in a fully bug proof system, but they drastically reduce the number of bugs as we add new code.

### 1.1.2 Reducing bugs in existing features

With quality tests in place, adding new features hardly breaks existing features. If a new feature breaks existing functionality, the existing tests fail, which makes it very easy to pinpoint where the errors occurred.

### 1.1.3 Tests improve design

When writing tests, one is forced to have testable code. We have used a strategy known as TDD(Test driven development) which ensures that you write efficient code that fulfils its basic functionality.

### 1.1.4 Testing makes development faster

Testing slows you down on a class-by-class basis, however with experience, your overall velocity increases because you need not fear breaking existing code when new features are added. With TDD, we realised that no extra code is written which saves coding hours and increases efficiency.

## 1.2 Project Outline

The main objective of this system is to allow a delivery person into a demarcated area of your house when you are not there. You should be able to give access remotely and monitor the delivery person while you they are in the area.

This document will demonstrate how the functionality of this system was tested by team codeBlox

## 1.3 Scope

The scope of this document is structured as follows. The features that are considered for testing are listed in section 3. Tests that have been identified from the requirements are discussed in detail in section 4. Furthermore, this document outlines the test environment and the risks involved in the testing approaches that will be followed. Assumptions and dependencies of this test plan will also be mentioned. Section 7 outlines, discusses and concludes on the results of the tests.

## 1.4 Test Environment

- Programming Language
  - node js
  - Angular js
  - Python
  - Java (android)
- Coding Environment
  - Node package envrionment (npm)
- Operating system
  - Linux
- Hardware
  - Hardware testing is done on a level of using a voltameter
  - Raspberry Pi 3

## 1.5 Assumption and Dependencies

- assume that user has android device and is able to use it
- assume that the Pi 3 and camera have been setup in the house

### 1.5.1 Dependencies

- angular
- bcrypt
- body-parser
- express
- jwt-simple
- mongojs
- mocha
- should
- supertest

all npm node-modules

# 2 Test Items

# 3 Functional Features to be Tested

- user registration
- notification when someone is at the gate
- open/close gate
- open camera

| Feature ID | RDS source | Summary | Test Case ID |
|---|---|---|---|
| 1 | server.get('/') | Testing connection to the server | 001 |
| 2 | server.post('/registration') | Persisting new user credentials | 002 |
| 3 | server.get('/returnUser/:email') | Verify credentials and allow/reject access | 003 |
| 4 | newUser.getActivationStatus() | Return true or false | 004 |
| 5 | newUser.getPin() | Get pin for user | 005 |
| 6 | newUser.getStaus() | Get connection status | 006 |

# 4 Test Cases

## 4.1 Test Case 1

**Test case 1**: connection to server
**Condition**: open home page
**Objective**: check if the Pi connects to the server
**Input**: web URL
**Outcome**: 200 status

## 4.2 Test Case 2

**Test case 2**: add user
**Condition**: user should not exist
**Objective**: check if user details get persisted to the database
**Input**: first name, last name, id, email, password1, password2
**Outcome**: 200 status and user added to database

## 4.3 Test Case 3

**Test case 3**: getting user email
**Condition**: user should exist

**Objective**: check if user info can be accessed
**Input**: user name
**Outcome**: print user email

## 4.4 Test Case 4

**Test case 4**: user status
**Condition**: user should exist and be active
**Objective**: check if correct status returned is active
**Input**: end Activation Date
**Outcome**: active

**Condition 2**: user should exist and be inactive
**Objective**: check if correct status returned is inactive
**Input**: end Activation Date
**Outcome**: inactive

## 4.5 Test Case 5

**Test case 5**: generating pin
**Condition**: request pin
**Objective**: generate pin for the back up system
**Input**: getpin()
**Outcome**: 8-digit random pin

## 4.6 Test Case 6

**Test case 6**: pin status
**Condition**: pin has been generated and un-used
**Objective**: check if pin has not been used
**Input**: getPinStatus()
**Outcome**: un-used

**Condition 2**: pin has been generated and used
**Objective**: check if pin has been used
**Input**: getPinStatus()
**Outcome**: used

# 5 Item Pass Criteria

- The home raspberry pi should be running
- The raspberry pi should be connected to the internet
- The raspberry should have a connection to the server

# 6 Test Deliverables

# 7 Detailed Test Results

## 7.1 Overview of Test Results

All the tests that were carried out passed and meet the expected results. These tests were carried out using the Mocha framework along with an assertion library called Chai.

## 7.2 Functional Requirements Test Results

Two separate tests were carried out to ensure that everything works. The first set of tests were done over the internet to ensure connection to server and correct communication with the server. The second set of tests, tested the functionality of the individual functions in the module locally. The tests are on Github at :https://github.com/billibongers/CodeBlox—Main-Project/tree/master/Code/Interfaces/tests and https://github.com/billibongers/CodeBlox—Main-Project/tree/master/Code/NodeJs/Services/personModule/test

### 7.2.1 Test Case 1 (4.1)

- The website opened and the video stream started
- Result: Pass

### 7.2.2 Test Case 2 (4.2)

- New user was added to the system
- server returned status code of 200
- Result: Pass

### 7.2.3 Test Case 3 (4.3)

- User email was returned
- The data was returned in the correct format
- Result: Pass

### 7.2.4 Test Case 4 (4.4)

- condition 1 returned active and condition 2 returned inactive
- Result: Pass

### 7.2.5 Test Case 5 (4.5)

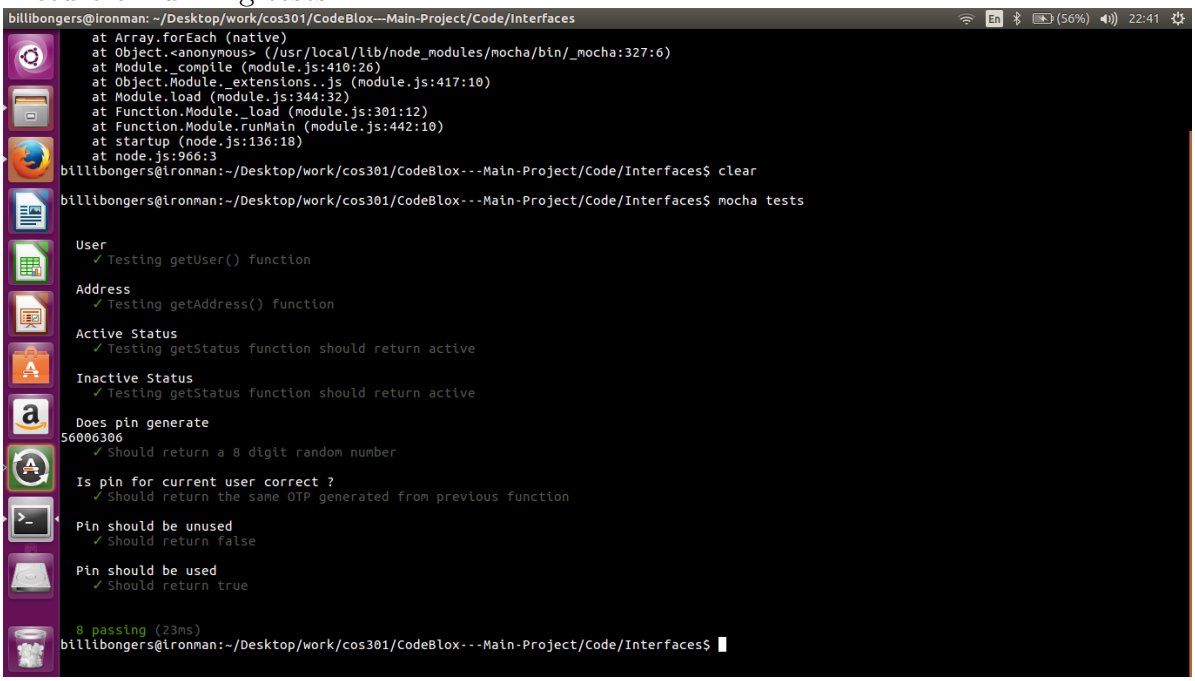
- Random 8-digit pin was created the pin was returned in the correct format
- pin was linked to correct user
- Result: Pass

### 7.2.6 Test Case 6 (4.6)

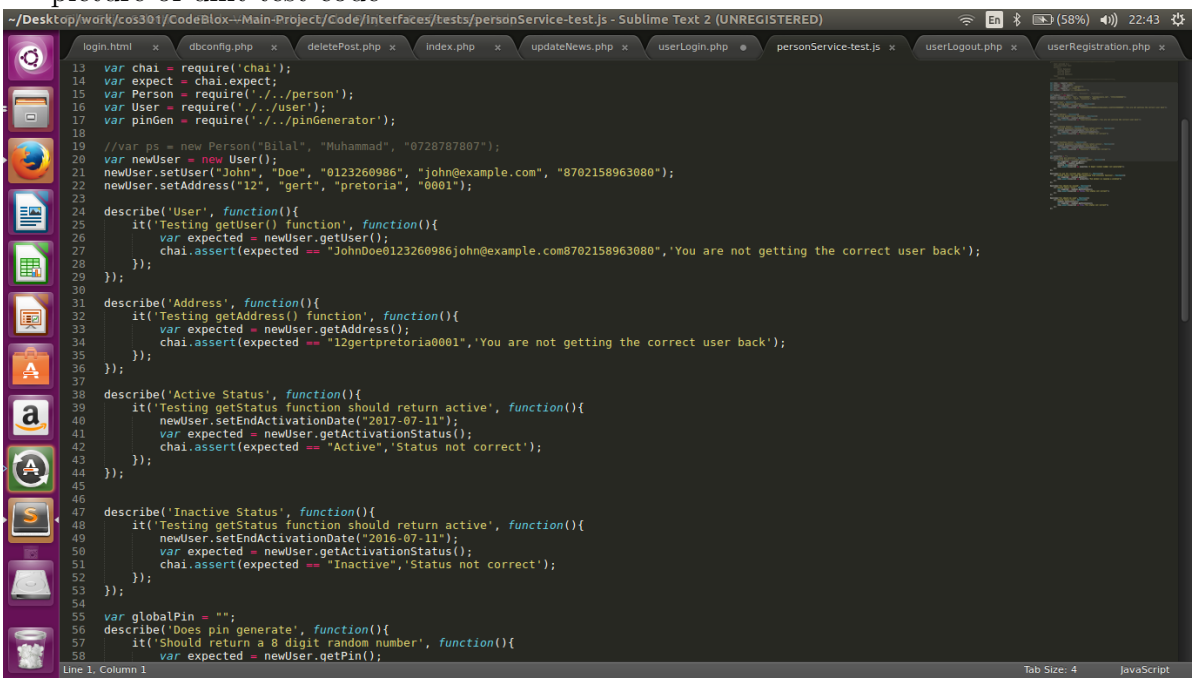
- Condition 1 returned un-used status and condition 2 returned used status
- Result: Pass

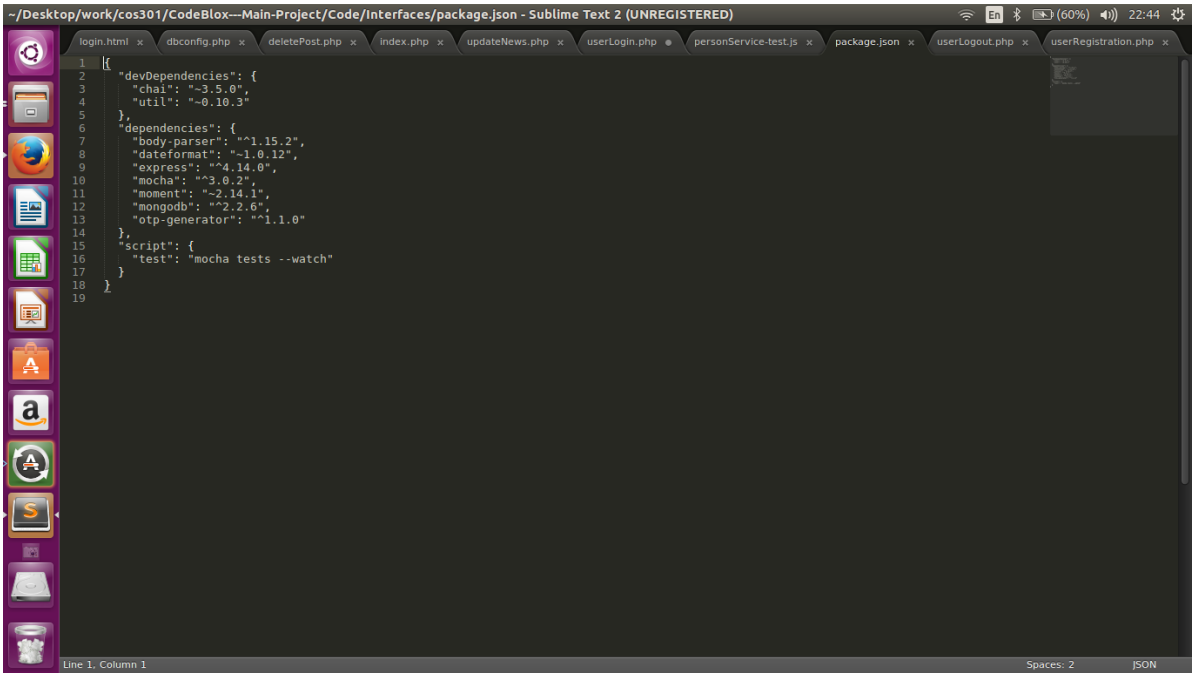
# 8 Other

Picture of running tests



picture of unit test code



# 9 Conclusions and Recommendations

We have gained much insight in terms of testing and its uses. We switched to a testing strategy called Test Driven Development, hereto referred as TTD. With TTD, we are able to write tests without any code written to pass it. This enables the user to write unbiased code because we have no knowledge of the inner code.

We have also learnt to write tests that test a function in depth. This was achieved by using a special library called chai that is used in conjunction with mocha. It has a vast library of asserts that can be used.

Overall, we are happy with our progress as far as testing is concerned. After completion, if time allows, we wish to carry out a usability test.