Binary Ninjaz

Harvest

Testing Policy

Letanyan Arumugam	14228123
Sizo Duma	15245579
Teboho Mokoena	14415888
John Ojo	15096794
Kevin Reid	15008739
Shaun Yates	16007493

Stakeholders

SAMAC:	Barry Christie

Contents

List of Figures

1 Testing Process

In this section, the testing process, based on IEEE-829, will illustrated. This process will fulfil the requirements stated in this document.

Three phases within the testing process can be identified, namely:

1. Test specification,
2. Test implementation,
3. Test reporting.

Each phase is now elaborated.

1.0.1 Test Specification

The test specification can be subdivided into 2 parts. On the one hand, there is *planning and verification*, on the other there is *Analysis, design, and implementation.*

Planning and verification may consist of the following activities:

- Specifying the scope and the risks, and defining the objectives of the tests;
- Determining the strategy;
- Taking decisions with regard to what must be tested, what roles apply, how the test activities will take place and how the outcomes of the tests will be implemented;
- Planning of the test design activities;
- Planning of implementation and evaluation.

During the *analysis, design, and implementation* phase, the test objectives will be transformed into specific test conditions and test cases. This may involve the following activities:

- A review of the test base (such as requirements, architecture, design, and interfaces);
- Evaluation of the testability of the test base and the test objects;

- Identification and prioritisation of the test conditions;
- Designing of the test cases;
- Identifying the necessary test data in order to support the test cases;
- Designing the test environment and identifying the infrastructure and tools.

1.0.2 Implementation of Tests

The implementation of tests forms the phase in which the test procedures and scripts are implemented and the results of these (including incidents) are logged. This may involve the following activities:

- Implementation of the tests, manually or by means of a tool;
- Logging the results of the implementation;
- Comparing the results with the result that had been predicted;
- Reporting any setbacks in the form of incidents and analysing these in order to identify the cause;
- Repeating the tests as a result of the incidents identified.

1.0.3 Test Reporting

Test Reporting forms the phase in which the implementation of the tests is compared with the objectives. This may involve the following tasks:

- Checking of the test logs and the list of defects and comparing these to the exit criteria specified in the test plan;
- Investigating whether additional tests are required;
- Writing a summary report.

1.1 Testing Frequency

Since tests are used to validate the continued performance of a module—or the system as a whole—tests should be run after each module is finished being modified or created.

2 Testing Tools

There are many types of tests that can be run, in the case of **Harvest**, Unit and UI tests are run.

2.1 Android

2.1.1 JUnit

JUnit is a unit testing framework for the Java Environment, and as such is used for testing the Android—programmed in Java—implementation of **Harvest**. Android development is carried out using the Android Studio IDE, which facilitates easy use and set up of the JUnit framework. Android Studio also recommends the use of JUnit. For the reasons given, JUnit was selected to run unit tests on Android.

Android unit tests can be found here.

2.1.2 Espresso

Espresso, like JUnit, is integrated into, and recommended by, Android Studio, and therefore is used to facilitate UI testing on Android.

Android UI tests can be found here.

2.1.3 Execution

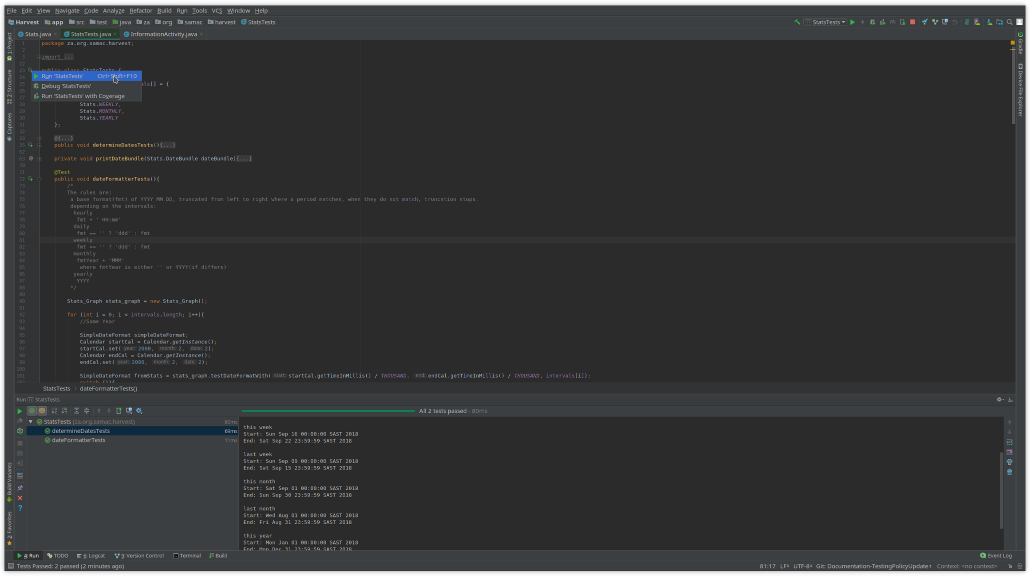
Both UI and Unit tests are executed in the same manner for Android, the primary difference being that a UI test will need to run on a connected device or virtual machine.

In order to run a unit or UI test for Android, one must open the file containing the desired test(s) using Android Studio as in figure 1. Once there, one can right click on the green 'test' button in the margin, then they can select to run the tests, the results of which will be seen at the bottom of the IDE.

2.2 iOS

2.2.1 XCTest

XCTest integrates seamlessly into Xcode's (the iOS IDE) testing workflow, and for that reason is used to facilitate all testing on iOS. XCTest is used for


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Harvest app src test java za org samac harvest StatsTests
Stats.java StatsTests.java InformationActivity.java
Run 'StatsTests' Ctrl+Shift+F10
Debug 'StatsTests'
Run 'StatsTests' with Coverage
StatsTests dateFormatterTests()
Run StatsTests
4: Run TODO 6: Logcat 9: Version Control Terminal Build

```
package za.org.samac.harvest;

import ...

                   als[] = {

            Stats.WEEKLY,
            Stats.MONTHLY,
            Stats.YEARLY
    };

    @{...}
    public void determineDatesTests(){...}

    private void printDateBundle(Stats.DateBundle dateBundle){...}

    @Test
    public void dateFormatterTests(){
        /*
        The rules are:
         a base format(fmt) of YYYY MM DD, truncated from left to right where a period matches, when they do not match, truncation stops.
         depending on the intervals:
          hourly
           fmt + ' HH:mm'
          daily
           fmt == '' ? 'ddd' : fmt
          weekly
           fmt == '' ? 'ddd' : fmt
          monthly
           fmtYear + 'MMM'
            where fmtYear is either '' or YYYY(if differs)
          yearly
           YYYY
         */

        Stats_Graph stats_graph = new Stats_Graph();

        for (int i = 0; i < intervals.length; i++){
            //Same Year

            SimpleDateFormat simpleDateFormat;
            Calendar startCal = Calendar.getInstance();
            startCal.set( year: 2000,  month: 2,  date: 2);
            Calendar endCal = Calendar.getInstance();
            endCal.set( year: 2000,  month: 2,  date: 2);

            SimpleDateFormat fromStats = stats_graph.testDateFormatWith( start: startCal.getTimeInMillis() / THOUSAND,  end: endCal.getTimeInMillis() / THOUSAND, intervals[i]);
```

```
All 2 tests passed - 80ms

StatsTests (za.org.samac.harvest)   80ms
    determineDatesTests             69ms
    dateFormatterTests              11ms
```

```
this week
Start: Sun Sep 16 00:00:00 SAST 2018
End: Sat Sep 22 23:59:59 SAST 2018

last week
Start: Sun Sep 09 00:00:00 SAST 2018
End: Sat Sep 15 23:59:59 SAST 2018

this month
Start: Sat Sep 01 00:00:00 SAST 2018
End: Sun Sep 30 23:59:59 SAST 2018

last month
Start: Wed Aug 01 00:00:00 SAST 2018
End: Fri Aug 31 23:59:59 SAST 2018

this year
Start: Mon Jan 01 00:00:00 SAST 2018
```

```
Tests Passed: 2 passed (2 minutes ago)
```

Figure 1: Android Studio Test Example

both UI and unit testing on iOS.

iOS unit tests can be found here, and UI tests can be found here.

2.2.2 Execution

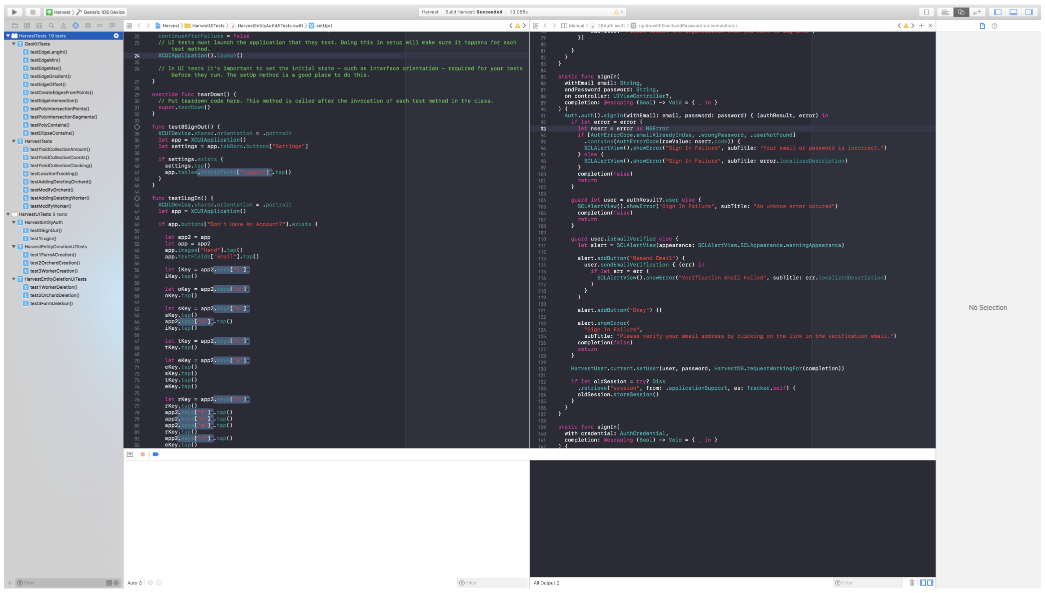
To begin tests in Xcode, click on the play button in the top right, on the line that is highlighted in blue; as seen in figure 2. This will run the tests. The results will look like figure 9.

2.3 Web

2.3.1 QUnit

QUnit is used for unit testing javascript functions. It was chosen since it displays its results in a convenient way—by including them in the web page, and it's very simple to use.

QUnit tests can be found here.

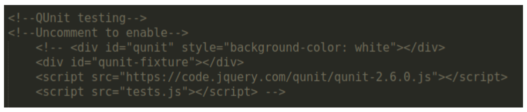

Harvest 〉 Generic iOS Device
Harvest | Build Harvest: Succeeded | 13.088s
Harvest 〉 HarvestUITests 〉 HarvestEntityAuthUITests.swift 〉 setUp()
Manual 〉 DBAuth.swift 〉 signIn(withEmail:andPassword:on:completion:)
No Selection
Auto
All Output

```
HarvestTests 19 tests
  GeoKitTests
    testEdgeLength()
    testEdgeMin()
    testEdgeMax()
    testEdgeGradient()
    testEdgeOffset()
    testCreateEdgesFromPoints()
    testEdgeIntersection()
    testPolyIntersectionPoints()
    testPolyIntersectionSegments()
    testPolyContains()
    testEllipseContains()
  HarvestTests
    testYieldCollectionAmount()
    testYieldCollectionCoords()
    testYieldCollectionClocking()
    testLocationTracking()
    testAddingDeletingOrchard()
    testModifyOrchard()
    testAddingDeletingWorker()
    testModifyWorker()
HarvestUITests 8 tests
  HarvestEntityAuth
    test0SignOut()
    test1LogIn()
  HarvestEntityCreationUITests
    test1FarmACreation()
    test2OrchardCreation()
    test3WorkerCreation()
  HarvestEntityDeletionUITests
    test1WorkerDeletion()
    test2OrchardDeletion()
    test3FarmDeletion()
```

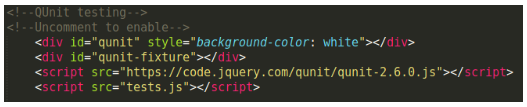
```swift
    continueAfterFailure = false
    // UI tests must launch the application that they test. Doing this in setup will make sure it happens for each
       test method.
    XCUIApplication().launch()

    // In UI tests it's important to set the initial state - such as interface orientation - required for your tests
       before they run. The setUp method is a good place to do this.
  }

  override func tearDown() {
    // Put teardown code here. This method is called after the invocation of each test method in the class.
    super.tearDown()
  }

  func test0SignOut() {
    XCUIDevice.shared.orientation = .portrait
    let app = XCUIApplication()
    let settings = app.tabBars.buttons["Settings"]

    if settings.exists {
      settings.tap()
      app.tables.staticTexts["Logout"].tap()
    }
  }

  func test1LogIn() {
    XCUIDevice.shared.orientation = .portrait
    let app = XCUIApplication()

    if app.buttons["Don't Have An Account?"].exists {

      let app2 = app
      let app = app2
      app.images["Hand"].tap()
      app.textFields["Email"].tap()

      let iKey = app2.keys["i"]
      iKey.tap()

      let oKey = app2.keys["o"]
      oKey.tap()

      let sKey = app2.keys["s"]
      sKey.tap()
      app2.keys["u"].tap()
      iKey.tap()

      let tKey = app2.keys["t"]
      tKey.tap()

      let eKey = app2.keys["e"]
      eKey.tap()
      sKey.tap()
      tKey.tap()
      eKey.tap()

      let rKey = app2.keys["r"]
      rKey.tap()
      app2.keys["@"].tap()
      app2.keys["h"].tap()
      app2.keys["a"].tap()
      rKey.tap()
      app2.keys["v"].tap()
      eKey.tap()
```

```swift
      })

    }
  }
}

static func signIn(
  withEmail email: String,
  andPassword password: String,
  on controller: UIViewController?,
  completion: @escaping (Bool) -> Void = { _ in }
) {
  Auth.auth().signIn(withEmail: email, password: password) { (authResult, error) in
    if let error = error {
      let nserr = error as NSError
      if [AuthErrorCode.emailAlreadyInUse, .wrongPassword, .userNotFound]
        .contains(AuthErrorCode(rawValue: nserr.code)) {
        SCLAlertView().showError("Sign In Failure", subTitle: "Your email or password is incorrect.")
      } else {
        SCLAlertView().showError("Sign In Failure", subTitle: error.localizedDescription)
      }
      completion(false)
      return
    }

    guard let user = authResult?.user else {
      SCLAlertView().showError("Sign In Failure", subTitle: "An unknow error occured")
      completion(false)
      return
    }

    guard user.isEmailVerified else {
      let alert = SCLAlertView(appearance: SCLAlertView.SCLAppearance.warningAppearance)

      alert.addButton("Resend Email") {
        user.sendEmailVerification { (err) in
          if let err = err {
            SCLAlertView().showError("Verification Email Failed", subTitle: err.localizedDescription)
          }
        }
      }

      alert.addButton("Okay") {}

      alert.showError(
        "Sign In Failure",
        subTitle: "Please verify your email address by clicking on the link in the verification email.")
      completion(false)
      return
    }

    HarvestUser.current.setUser(user, password, HarvestDB.requestWorkingFor(completion))

    if let oldSession = try? Disk
      .retrieve("session", from: .applicationSupport, as: Tracker.self) {
      oldSession.storeSession()
    }
  }
}

static func signIn(
  with credential: AuthCredential,
  completion: @escaping (Bool) -> Void = { _ in }
) {
```

Figure 2: iOS Xcode Test Start



```html
<!--QUnit testing-->
<!--Uncomment to enable-->
    <!-- <div id="qunit" style="background-color: white"></div>
    <div id="qunit-fixture"></div>
    <script src="https://code.jquery.com/qunit/qunit-2.6.0.js"></script>
    <script src="tests.js"></script> -->
```



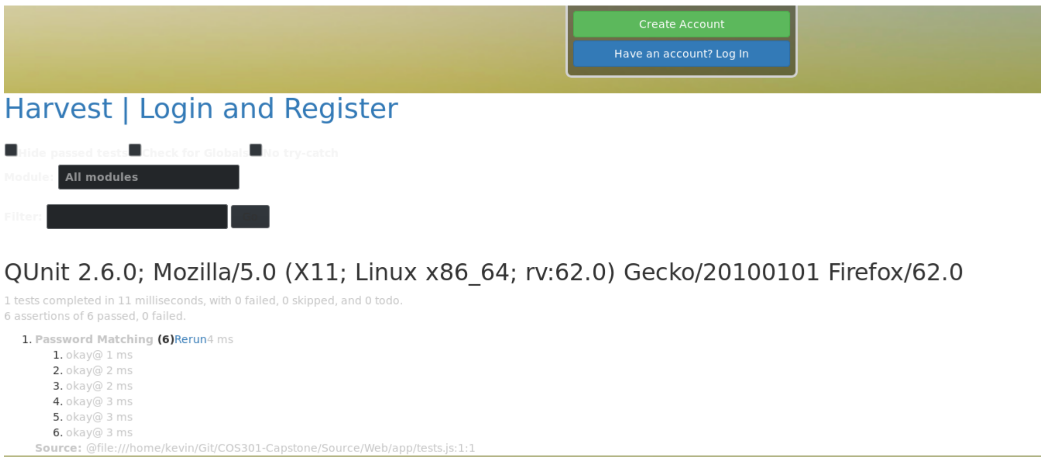
```html
<!--QUnit testing-->
<!--Uncomment to enable-->
    <div id="qunit" style="background-color: white"></div>
    <div id="qunit-fixture"></div>
    <script src="https://code.jquery.com/qunit/qunit-2.6.0.js"></script>
    <script src="tests.js"></script>
```

Figure 3: Before Enabling Tests

Figure 4: After Enabling Tests

2.3.2 Execution

QUnit To execute QUnit tests, go into the html file that uses the functions to be tested. At the bottom, find a segment that mentions `QUnit Testing`, such as in figure 3; then simply uncomment the code, as in figure 4, and refresh the page. The results are now displayed, such as in figure 5.



Harvest | Login and Register

```
QUnit 2.6.0; Mozilla/5.0 (X11; Linux x86_64; rv:62.0) Gecko/20100101 Firefox/62.0
1 tests completed in 11 milliseconds, with 0 failed, 0 skipped, and 0 todo.
6 assertions of 6 passed, 0 failed.
   1. Password Matching (6)Rerun4 ms
        1. okay@ 1 ms
        2. okay@ 2 ms
        3. okay@ 2 ms
        4. okay@ 3 ms
        5. okay@ 3 ms
        6. okay@ 3 ms
      Source: @file:///home/kevin/Git/COS301-Capstone/Source/Web/app/tests.js:1:1
```

Figure 5: Results of QUnit test

3 Test Cases

3.1 Database

The database cannot be tested locally with unit tests, so UI tests are used to perform create, read, update, and delete (CRUD) operations on the database. The tests also confirm that after the operations are completed, the database returns the correct information.

These UI tests can be found here, here, and here for iOS; and here for Android. These tests are all simply performings one or more CRUD operation pertaining to the database, and confirming that the correct results are returned after the operation; before moving onto the next.

3.2 Location Services

Location services are handled by device hardware and cannot be reasonably tested, however, it is possible to test the handling of the results returned by the location services.

Examples of these tests can be found here on iOS, and here for Android.

These tests deal mostly with geo-fencing—testing if a location falls within a given area. These functions serve to test which orchard, if any, a foreman is within when they make use of the mobile application.

The testing is carried out by creating a polygon demarcated by a set of coordinates, then feeding the geo-fencing functions a set of predetermined coordinates to test if the correct results are returned by these functions.

3.3 Cloud Functions

A variety of functions that operate on the database are hosted in the cloud, so that mobile applications can request the results of a function rather than download all the necessary data from the database and perform the calculations locally, this saves on valuable mobile data.

Tests for these functions can be found here.

These functions are tested simply by giving them dummy data and confirming that the correct results are returned.

3.4 Mobile Application and Website

These are umbrella terms that each encompass a variety of smaller subsystems.

3.5 Collection Tracker

The collection tracker (also termed as yield tracker) is the primary mobile activity performed by foremen. It is how they measure the collections of the workers.

On iOS this is tested by unit tests that create fake information and then feeds it into the tracker to confirm that arbitary information can be processed. Simultaneoulsy, it is confirmed that the location and time measured by the tracker is handled correctly. These tests can be found here.

On Android, testing is carried out by means of UI tests which confirm that arrival to the tracker and its operations is carried out correctly as a whole. The tests can be found here.

3.6 Foreman Location Viewer

The foreman location viewer is used to display the live locations of requested foremen, it relies heavily on accurate location reporting from the foreman tracker. Once the information is received, it is simply displayed.

3.7 Foreman Tracker

The foreman tracker is used to report the location of the foreman (the mobile application) to the foreman location viewer. It relies almost exclusively on accurate location reporting, which is received from hardware and written to the database.

3.8 Collection Viewer

The collection viewer shows summaries of previously recorded sessions. It is used solely to report simple previously recorded information to the user. This information is read from the database, tests for which have already been discussed.

3.9 Statistics Viewer

The statistics viewer is used to report information to the user in the form of line graphs. Given the large amounts of data processed, this is handled by cloud functions, testing for which has already been discussed. The complication being the correct reporting of the information received.

Android unit tests, which confirm the correct handling of dates can be found here.

3.10 Information Manager

The information manager is used exclusively to perform basic CRUD operations on the database, no additional processing is performed. Tests for CRUD operations have already been discussed.

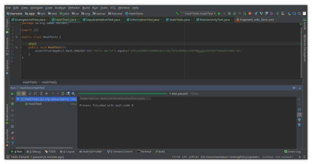
On Android this is tested via a UI test here, which runs through adding, modifying, and deleting information.

4 History

4.1 Android

Android test logs can found here.

Refer to figures 6, 7, and 8 for examples of tests.

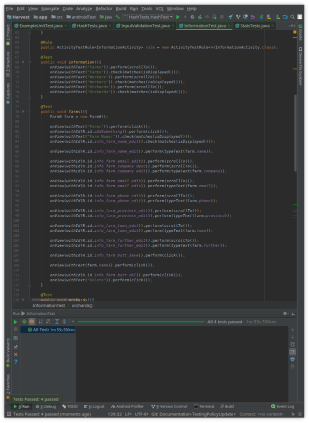


```
package za.org.samac.harvest;

import ...

public class HashTests {

    @Test
    public void HashTest(){
        assertTrue(AppUtil.Hash.SHA256( text: "Hello World").equals("a591a6d40bf420404a011733cfb7b190d62c65bf0bcda32b57b277d9ad9f146e"));
    }
}
```

```
/home/kevin/.Android/Studio/jre/bin/java ...

Process finished with exit code 0
```

1 test passed - 13ms

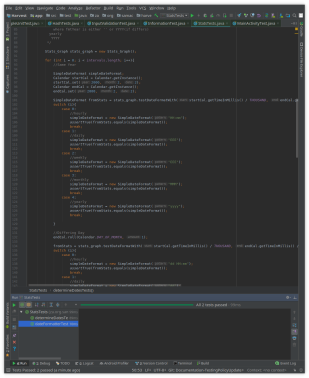
Tests Passed: 1 passed (a minute ago)



```
    }

    @Rule
    public ActivityTestRule<InformationActivity> rule = new ActivityTestRule<>(InformationActivity.class);

    @Test
    public void information(){
        onView(withText("Farms")).perform(scrollTo());
        onView(withText("Farms")).check(matches(isDisplayed()));
        onView(withText("Workers")).perform(scrollTo());
        onView(withText("Workers")).check(matches(isDisplayed()));
        onView(withText("Orchards")).perform(scrollTo());
        onView(withText("Orchards")).check(matches(isDisplayed()));
    }

    @Test
    public void farms(){
        Farm0 farm = new Farm0();

        onView(withText("Farms")).perform(click());
        onView(withId(R.id.addSomething)).perform(click());
        onView(withText("Farm Name:")).check(matches(isDisplayed()));
        onView(withId(R.id.info_farm_name_edit)).check(matches(isDisplayed()));

        onView(withId(R.id.info_farm_name_edit)).perform(typeText(farm.name));

        onView(withId(R.id.info_farm_email_edit)).perform(scrollTo());
        onView(withId(R.id.info_farm_company_desc)).perform(scrollTo());
        onView(withId(R.id.info_farm_company_edit)).perform(typeText(farm.company));

        onView(withId(R.id.info_farm_email_edit)).perform(scrollTo());
        onView(withId(R.id.info_farm_email_edit)).perform(typeText(farm.email));

        onView(withId(R.id.info_farm_phone_edit)).perform(scrollTo());
        onView(withId(R.id.info_farm_phone_edit)).perform(typeText(farm.phone));

        onView(withId(R.id.info_farm_province_edit)).perform(scrollTo());
        onView(withId(R.id.info_farm_province_edit)).perform(typeText(farm.province));

        onView(withId(R.id.info_farm_town_edit)).perform(scrollTo());
        onView(withId(R.id.info_farm_town_edit)).perform(typeText(farm.town));

        onView(withId(R.id.info_farm_further_edit)).perform(scrollTo());
        onView(withId(R.id.info_farm_further_edit)).perform(typeText(farm.further));

        onView(withId(R.id.info_farm_butt_save)).perform(click());

        onView(withText(farm.name)).perform(click());

        onView(withId(R.id.info_farm_butt_del)).perform(click());
        onView(withText("Delete")).perform(click());
    }

    @Test
    public void orchards(){
```

All 4 tests passed - 1m 53s 530ms

Tests Passed: 4 passed (moments ago)



```
         where fmtYear is either '' or YYYY(if differs)
        yearly
         YYYY
       */

        Stats_Graph stats_graph = new Stats_Graph();

        for (int i = 0; i < intervals.length; i++){
            //Same Year

            SimpleDateFormat simpleDateFormat;
            Calendar startCal = Calendar.getInstance();
            startCal.set( year: 2000, month: 2, date: 2);
            Calendar endCal = Calendar.getInstance();
            endCal.set( year: 2000, month: 2, date: 2);

            SimpleDateFormat fromStats = stats_graph.testDateFormatWith( start: startCal.getTimeInMillis() / THOUSAND, end: endCal.ge
            switch (i){
                case 0:
                    //hourly
                    simpleDateFormat = new SimpleDateFormat( pattern: "HH:mm");
                    assertTrue(fromStats.equals(simpleDateFormat));
                    break;
                case 1:
                    //daily
                    simpleDateFormat = new SimpleDateFormat( pattern: "EEE");
                    assertTrue(fromStats.equals(simpleDateFormat));
                    break;
                case 2:
                    //weekly
                    simpleDateFormat = new SimpleDateFormat( pattern: "EEE");
                    assertTrue(fromStats.equals(simpleDateFormat));
                    break;
                case 3:
                    //monthly
                    simpleDateFormat = new SimpleDateFormat( pattern: "MMM");
                    assertTrue(fromStats.equals(simpleDateFormat));
                    break;
                case 4:
                    //yearly
                    simpleDateFormat = new SimpleDateFormat( pattern: "yyyy");
                    assertTrue(fromStats.equals(simpleDateFormat));
                    break;

            }

            //Differing Day
            endCal.roll(Calendar.DAY_OF_MONTH, amount: 1);

            fromStats = stats_graph.testDateFormatWith( start: startCal.getTimeInMillis() / THOUSAND, end: endCal.getTimeInMillis() /
            switch (i){
                case 0:
                    //hourly
                    simpleDateFormat = new SimpleDateFormat( pattern: "dd HH:mm");
                    assertTrue(fromStats.equals(simpleDateFormat));
                    break;
                case 1:
                    //daily
```

All 2 tests passed - 99ms

Tests Passed: 2 passed (a minute ago)

Figure 6: Unit Test to Verify Functioning of Hash Function

Figure 7: UI Test to Verify Functioning of Information Editing

Figure 8: Unit Test to Verify Formatting of Dates

4.2 iOS

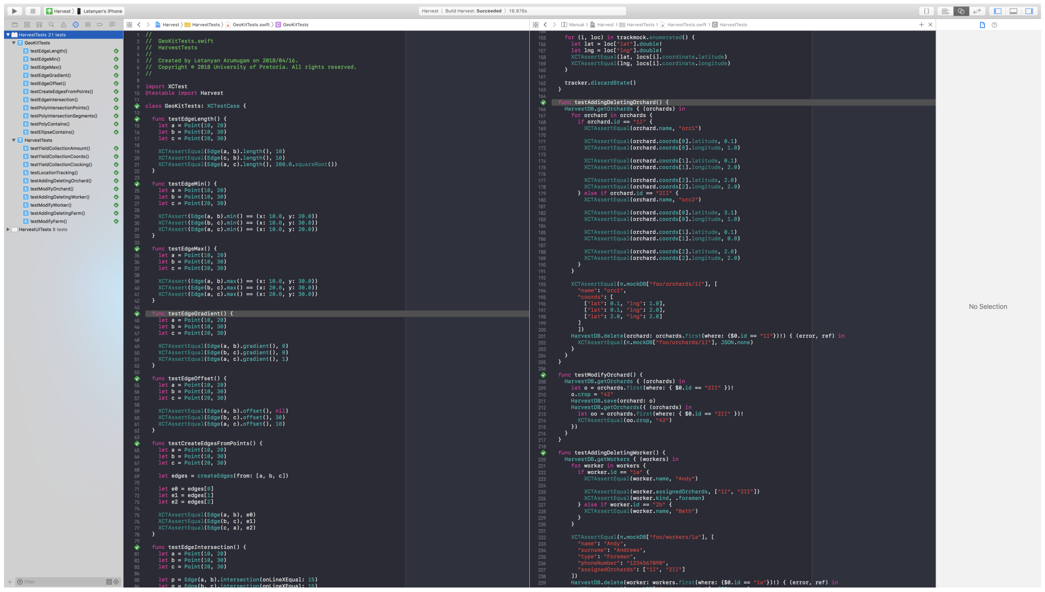
iOS test logs can be found here.

Refer to figure 9 to see iOS tests.

4.3 Web

Web test logs can be found here.

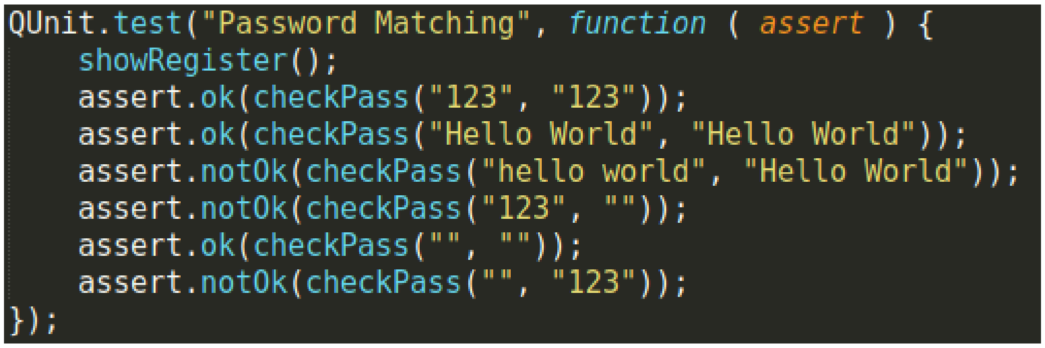
Refer to figure 10 for an example of a QUnit test.


Harvest 〉 Letanyan's iPhone
Harvest | Build Harvest: Succeeded | 16.976s
Harvest 〉 HarvestTests 〉 GeoKitTests.swift 〉 GeoKitTests
Manual 〉 Harvest 〉 HarvestTests 〉 HarvestTests.swift 〉 HarvestTests
HarvestTests 21 tests
GeoKitTests
testEdgeLength()
testEdgeMin()
testEdgeMax()
testEdgeGradient()
testEdgeOffset()
testCreateEdgesFromPoints()
testEdgeIntersection()
testPolyIntersectionPoints()
testPolyIntersectionSegments()
testPolyContains()
testEllipseContains()
HarvestTests
testYieldCollectionAmount()
testYieldCollectionCoords()
testYieldCollectionClocking()
testLocationTracking()
testAddingDeletingOrchard()
testModifyOrchard()
testAddingDeletingWorker()
testModifyWorker()
testAddingDeletingFarm()
testModifyFarm()
HarvestUITests 8 tests
No Selection

```swift
//
//  GeoKitTests.swift
//  HarvestTests
//
//  Created by Letanyan Arumugam on 2018/04/16.
//  Copyright © 2018 University of Pretoria. All rights reserved.
//

import XCTest
@testable import Harvest

class GeoKitTests: XCTestCase {

  func testEdgeLength() {
    let a = Point(10, 20)
    let b = Point(10, 30)
    let c = Point(20, 30)

    XCTAssertEqual(Edge(a, b).length(), 10)
    XCTAssertEqual(Edge(c, b).length(), 10)
    XCTAssertEqual(Edge(a, c).length(), 200.0.squareRoot())
  }

  func testEdgeMin() {
    let a = Point(10, 20)
    let b = Point(10, 30)
    let c = Point(20, 30)

    XCTAssert(Edge(a, b).min() == (x: 10.0, y: 20.0))
    XCTAssert(Edge(b, c).min() == (x: 10.0, y: 30.0))
    XCTAssert(Edge(a, c).min() == (x: 10.0, y: 20.0))
  }

  func testEdgeMax() {
    let a = Point(10, 20)
    let b = Point(10, 30)
    let c = Point(20, 30)

    XCTAssert(Edge(a, b).max() == (x: 10.0, y: 30.0))
    XCTAssert(Edge(b, c).max() == (x: 20.0, y: 30.0))
    XCTAssert(Edge(a, c).max() == (x: 20.0, y: 30.0))
  }

  func testEdgeGradient() {
    let a = Point(10, 20)
    let b = Point(10, 30)
    let c = Point(20, 30)

    XCTAssertEqual(Edge(a, b).gradient(), 0)
    XCTAssertEqual(Edge(b, c).gradient(), 0)
    XCTAssertEqual(Edge(a, c).gradient(), 1)
  }

  func testEdgeOffset() {
    let a = Point(10, 20)
    let b = Point(10, 30)
    let c = Point(20, 30)

    XCTAssertEqual(Edge(a, b).offset(), nil)
    XCTAssertEqual(Edge(b, c).offset(), 30)
    XCTAssertEqual(Edge(a, c).offset(), 10)
  }

  func testCreateEdgesFromPoints() {
    let a = Point(10, 20)
    let b = Point(10, 30)
    let c = Point(20, 30)

    let edges = createEdges(from: [a, b, c])

    let e0 = edges[0]
    let e1 = edges[1]
    let e2 = edges[2]

    XCTAssertEqual(Edge(a, b), e0)
    XCTAssertEqual(Edge(b, c), e1)
    XCTAssertEqual(Edge(c, a), e2)
  }

  func testEdgeIntersection() {
    let a = Point(10, 20)
    let b = Point(10, 30)
    let c = Point(20, 30)

    let p = Edge(a, b).intersection(onLineXEqual: 15)
    let q = Edge(b, c).intersection(onLineXEqual: 15)
```

```swift
    for (i, loc) in trackmock.enumerated() {
      let lat = loc["lat"].double!
      let lng = loc["lng"].double!
      XCTAssertEqual(lat, locs[i].coordinate.latitude)
      XCTAssertEqual(lng, locs[i].coordinate.longitude)
    }

    tracker.discardState()
  }

  func testAddingDeletingOrchard() {
    HarvestDB.getOrchards { (orchards) in
      for orchard in orchards {
        if orchard.id == "1I" {
          XCTAssertEqual(orchard.name, "orc1")

          XCTAssertEqual(orchard.coords[0].latitude, 0.1)
          XCTAssertEqual(orchard.coords[0].longitude, 1.0)

          XCTAssertEqual(orchard.coords[1].latitude, 0.1)
          XCTAssertEqual(orchard.coords[1].longitude, 2.0)

          XCTAssertEqual(orchard.coords[2].latitude, 2.0)
          XCTAssertEqual(orchard.coords[2].longitude, 2.0)
        } else if orchard.id == "2II" {
          XCTAssertEqual(orchard.name, "orc2")

          XCTAssertEqual(orchard.coords[0].latitude, 3.1)
          XCTAssertEqual(orchard.coords[0].longitude, 1.0)

          XCTAssertEqual(orchard.coords[1].latitude, 0.1)
          XCTAssertEqual(orchard.coords[1].longitude, 0.0)

          XCTAssertEqual(orchard.coords[2].latitude, 2.0)
          XCTAssertEqual(orchard.coords[2].longitude, 2.0)
        }
      }

      XCTAssertEqual(π.mockDB["foo/orchards/1I"], [
        "name": "orc1",
        "coords": [
          ["lat": 0.1, "lng": 1.0],
          ["lat": 0.1, "lng": 2.0],
          ["lat": 2.0, "lng": 2.0]
        ]
        ])
      HarvestDB.delete(orchard: orchards.first(where: {$0.id == "1I"})!) { (error, ref) in
        XCTAssertEqual(π.mockDB["foo/orchards/1I"], JSON.none)
      }
    }
  }

  func testModifyOrchard() {
    HarvestDB.getOrchards { (orchards) in
      let o = orchards.first(where: { $0.id == "2II" })!
      o.crop = "42"
      HarvestDB.save(orchard: o)
      HarvestDB.getOrchards({ (orchards) in
        let oo = orchards.first(where: { $0.id == "2II" })!
        XCTAssertEqual(oo.crop, "42")
      })
    }
  }

  func testAddingDeletingWorker() {
    HarvestDB.getWorkers { (workers) in
      for worker in workers {
        if worker.id == "1a" {
          XCTAssertEqual(worker.name, "Andy")

          XCTAssertEqual(worker.assignedOrchards, ["1I", "2II"])
          XCTAssertEqual(worker.kind, .foreman)
        } else if worker.id == "2b" {
          XCTAssertEqual(worker.name, "Beth")
        }
      }

      XCTAssertEqual(π.mockDB["foo/workers/1a"], [
        "name": "Andy",
        "surname": "Andrews",
        "type": "Foreman",
        "phoneNumber": "1234567890",
        "assignedOrchards": ["1I", "2II"]
      ])
      HarvestDB.delete(worker: workers.first(where: {$0.id == "1a"})!) { (error, ref) in
```

Figure 9: iOS Xcode Test Example



```javascript
QUnit.test("Password Matching", function ( assert ) {
    showRegister();
    assert.ok(checkPass("123", "123"));
    assert.ok(checkPass("Hello World", "Hello World"));
    assert.notOk(checkPass("hello world", "Hello World"));
    assert.notOk(checkPass("123", ""));
    assert.ok(checkPass("", ""));
    assert.notOk(checkPass("", "123"));
});
```

Figure 10: QUnit Test to Verify Passwords are Rejected or Approved Appropriately