UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Faculty of Engineering, Built Environment and Information Technology

Department of Computer Science

Faculty of Engineering, Built Environment & IT

University of Pretoria

COS301 - Software Engineering

Coviduous

Testing Policy

September 18, 2021

Team Name: **CAPSlock**

Team Members: ** - indicates team leader*

Name	Surname	Student Number
Njabulo*	Skosana*	u18089102*
Rudolf	van Graan	u16040865
Clementime	Mashile	u18139508
Peter	Okumbe	u18052640
Chaoane	Malakoane	u18202374

Contents

1 Continuous Integration

1.1 Testing Tool

When considering a tool for automatic testing, we turned to Github Actions and Travis CI, owing to their popularity and wide range of features.

Advantages of Travis CI:

- Built-in GitHub integration - users do not have to manually connect their GitHub to the tool; once given permission, Travis is automatically integrated with the user's GitHub account and has access to their repositories.
- You can monitor GitHub projects.
- Support for a variety of languages like Android, C, C#, C++, Java, JavaScript (with Node.js), Perl, PHP, Python, R, Ruby, etc.
- Easy Deployment to cloud services.
- Comes with free cloud-based hosting which does not require maintenance or administration.
- Whereas Travis CI offers unlimited builds for their payment plans, they do not however, offer a free tier credit renewal every month.

Advantages of Github Actions:

- GitHub Actions makes it easy to automate all your software workflows, with integrated CI/CD.
- Allows to build, test, and deploy your code right from GitHub. Make code reviews, branch management, and issue triaging work the way you want.
- GitHub Actions supports a wide range of programming languages including: Node.js, Python, Java, Ruby, PHP, Go, Rust, .NET, and more. Build, test, and deploy applications in the language of your choice.
- Github Actions is free for all github public repositories - no credit card or admin necessary.
- Free tier credits are renew/reset each month.

However, on comparison of the two, besides being very similar to each other, using Travis CI does not offer a free-tier renewal every month where as Github Actions does and is already integrated in github and is easy to set up, also taking away the overhead of cost involved for testing and was recommended for all teams to use. Github Actions also allows for much easier viewing of the pipeline status for each build since you do not have to leave GitHub's website to see any logs and/or errors; the tab is right there as part of your repository's UI.

1.2 Procedure

When one pushes to the master branch, the tests are automatically run for the backend API server functions as well as the frontend application screens. These tests are run with the software when it is pushed to the repository. One will then see if the build status is successful or not.

`images/.png`

Figure 1: Github Actions Test Report

2 Front-end testing

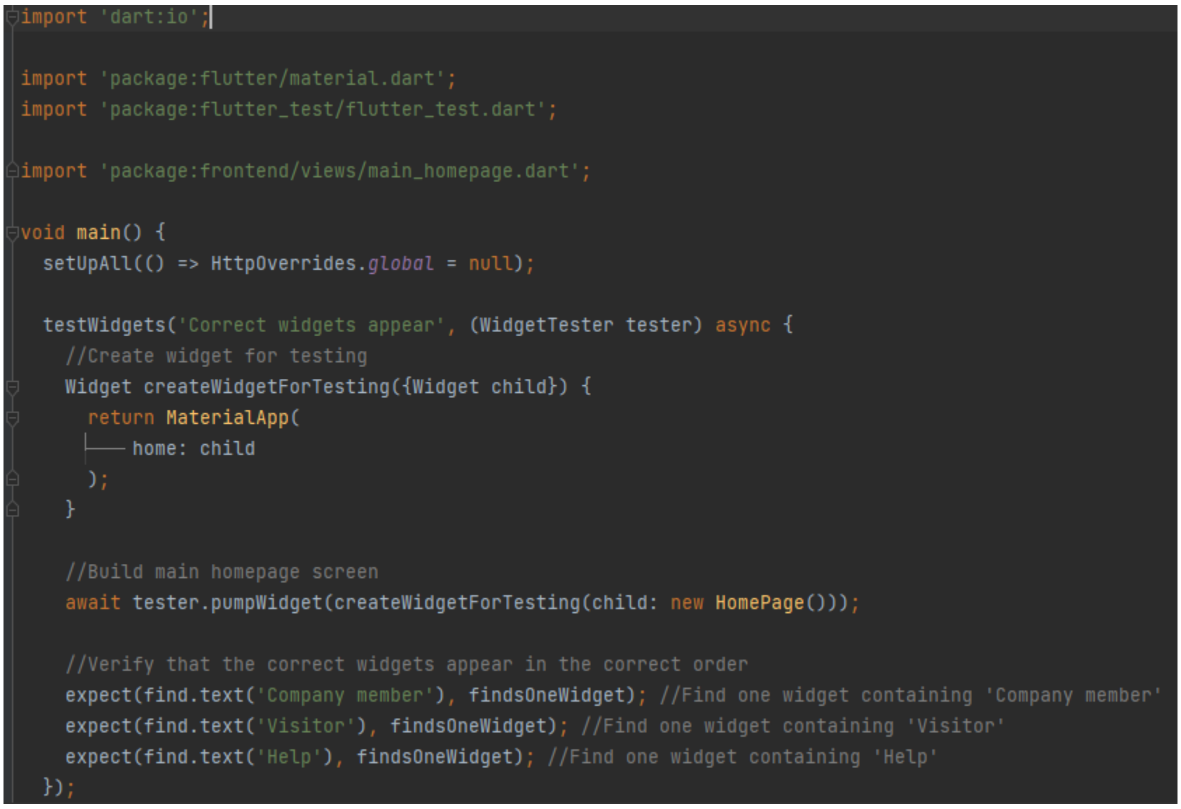
2.1 Testing Tool

We made use of Flutter's build-in testing framework library as our frontend testing tool. It provides two packages - the 'test' package provides the core framework for writing unit tests, and the 'flutter_test' package provides additional utilities for testing widgets.
Given our frontend being developed in the flutter framework and it already having a built-in established tool for testing flutter frameworks, we felt no need to use a different testing tool. This allowed for easily integrating and making use of the built-in testing tool to test our frontend functionality.

2.2 Procedure

2.2.1 Test suites

We divided our tests into different test files for each subsystem using the following naming convention: 'filename_test.dart' - this is the convention used by the test runner when searching for tests. All our test files reside inside a 'test' folder located at the root of our Flutter application. In each test file, the respective subsystem's frontend screens functionalities are tested independently using the 'test' and 'testWidgets' function calls with each test case residing in the body of those functions.

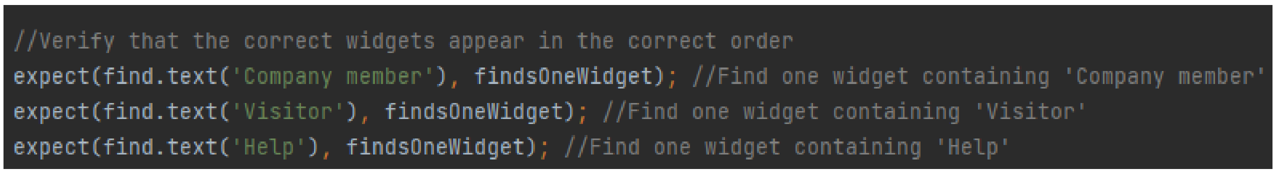


```
import 'dart:io';

import 'package:flutter/material.dart';
import 'package:flutter_test/flutter_test.dart';

import 'package:frontend/views/main_homepage.dart';

void main() {
  setUpAll(() => HttpOverrides.global = null);

  testWidgets('Correct widgets appear', (WidgetTester tester) async {
    //Create widget for testing
    Widget createWidgetForTesting({Widget child}) {
      return MaterialApp(
        home: child
      );
    }

    //Build main homepage screen
    await tester.pumpWidget(createWidgetForTesting(child: new HomePage()));

    //Verify that the correct widgets appear in the correct order
    expect(find.text('Company member'), findsOneWidget); //Find one widget containing 'Company member'
    expect(find.text('Visitor'), findsOneWidget); //Find one widget containing 'Visitor'
    expect(find.text('Help'), findsOneWidget); //Find one widget containing 'Help'
  });
```

Figure 2: Flutter screen test

2.2.2 Assertions

Similar to the syntax of most Javascript testing framework assertion libraries, the flutter framework testing library format constructs and tests assertions at the end of each test case, described by the 'expect' function as seen below:



```
//Verify that the correct widgets appear in the correct order
expect(find.text('Company member'), findsOneWidget); //Find one widget containing 'Company member'
expect(find.text('Visitor'), findsOneWidget); //Find one widget containing 'Visitor'
expect(find.text('Help'), findsOneWidget); //Find one widget containing 'Help'
```

Figure 3: Flutter test assertions

3 Back-end testing

3.1 Testing Tool

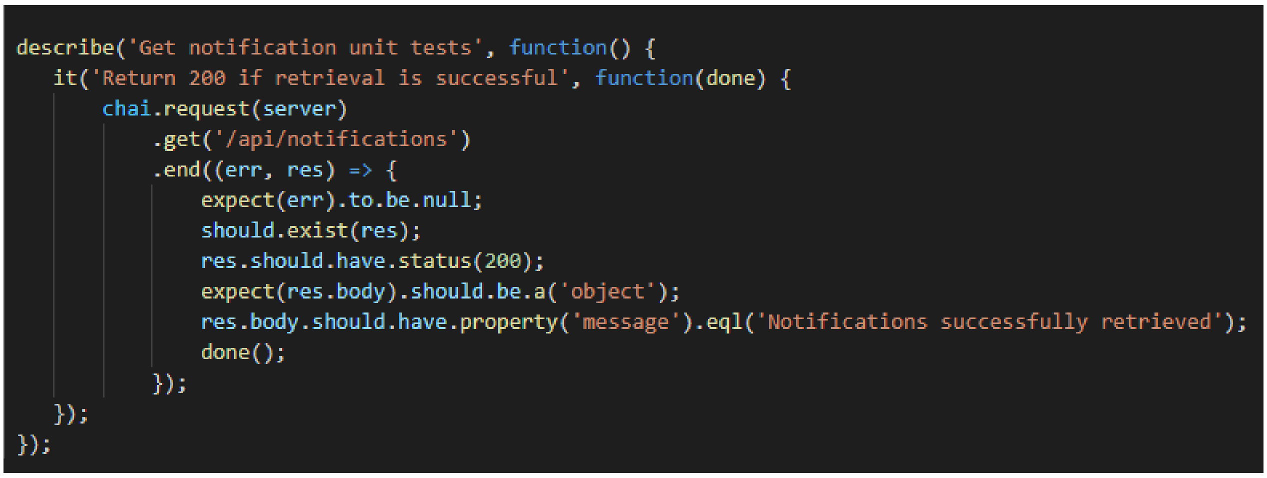
We made use of the Javascript Mocha testing framework and Chai assertion library for our backend testing tool.
Advantages of Mocha and Chai:

- Mocha is a solid open-source testing framework used by JavaScript developers for unit testing. It supports several assertion libraries making it more flexible. We decided to use chai assertion library that is mostly used with Mocha which contains helpful functionality to verify test results.
- Mocha integrates well with NodeJS as the Mocha test framework runs on NodeJS itself. With our backend developed using NodeJS, it allowed for simplicity of testing our system's API.
- Mocha makes it easy to test both synchronous and asynchronous code. Our backend rest API developed using NodeJS has a number of asynchronous function calls and Mocha allows us to test these with support for call-back functions, promises and async/await. We made use of the call-back function "done" to test our asynchronous code.
- Mocha and chai are also the testing frameworks used by Google to test their cloud functions and since we make use of firebase functions for our backend, this was the most notable and viable testing tool approach to use.

3.2 Procedure

3.2.1 Test suites

We divided our tests into different test files for each subsystem using the following the naming convention: 'subsystem_name.test.js'. In each test file, the respective subsystem's test cases were tested independently using Mocha's 'describe' and 'it' function calls with each test case residing in the body of the 'describe' function. The 'describe' function is used for the grouping, describing and separating of tests and the 'it' function specifies the purpose of the test as seen below:

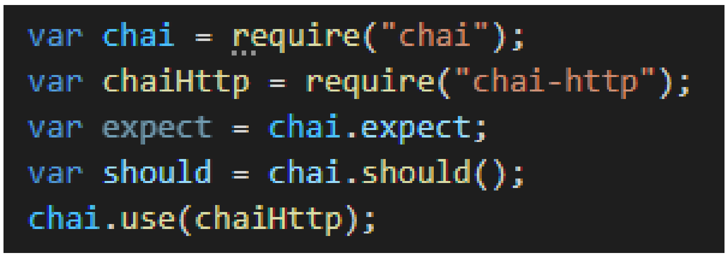


```
describe('Get notification unit tests', function() {
   it('Return 200 if retrieval is successful', function(done) {
       chai.request(server)
           .get('/api/notifications')
           .end((err, res) => {
               expect(err).to.be.null;
               should.exist(res);
               res.should.have.status(200);
               expect(res.body).should.be.a('object');
               res.body.should.have.property('message').eql('Notifications successfully retrieved');
               done();
           });
   });
});
```

Figure 4: Mocha testing using 'describe' and 'it'

3.2.2 Assertions

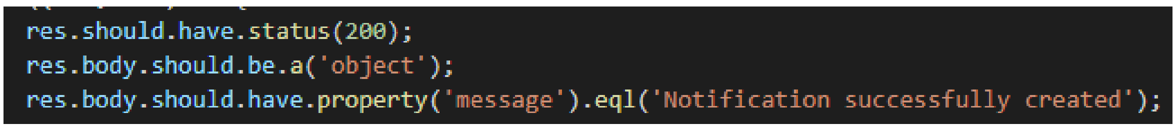
The general set up of the use of the chai assertion library can be seen below: Chai's 'should' and 'expect' functions are used to construct assertions by using language chains to compare and improve readability of assertions.



```
var chai = require("chai");
var chaiHttp = require("chai-http");
var expect = chai.expect;
var should = chai.should();
chai.use(chaiHttp);
```

Figure 5: Chai assertion library variables

The following example shows the format that was used for our assertions using the Chai assertion library. The assertion checks if the returned response body is an object, if it has status code 200, and if the message in the response body is equivalent to the message given as input in the assertion.



```
res.should.have.status(200);
res.body.should.be.a('object');
res.body.should.have.property('message').eql('Notification successfully created');
```

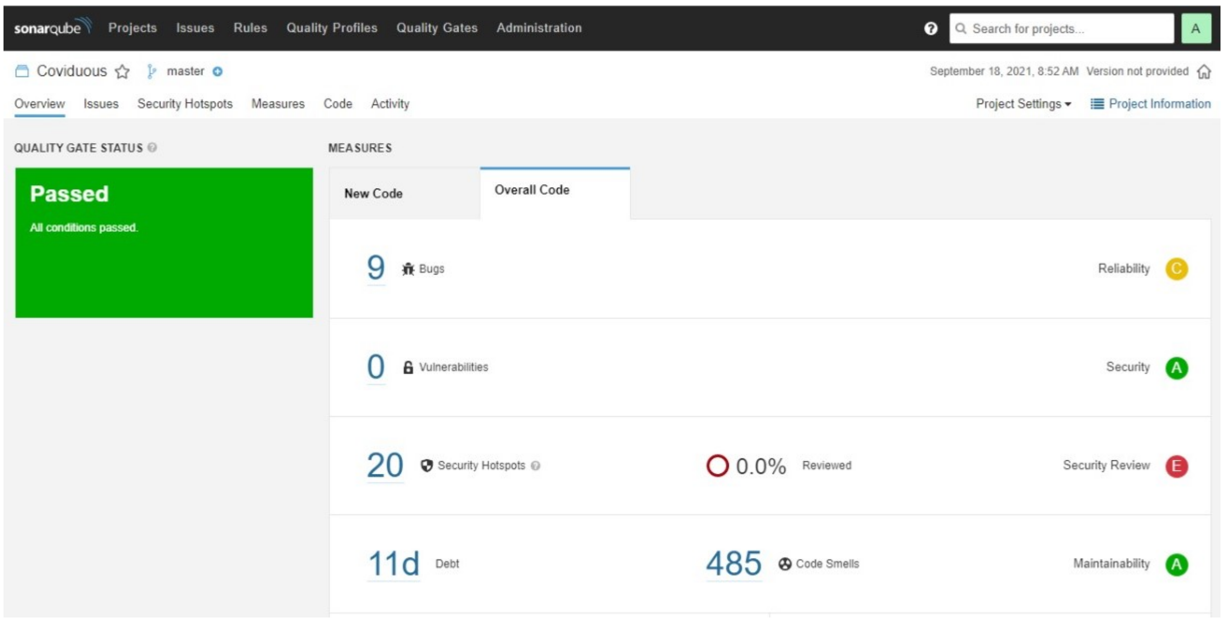
Figure 6: Chai assertions

4 Non-functional requirements testing

4.1 Testing Tool

We made use of the Javascript Mocha testing framework as well as SonarQube testing tool to test and analyze our non-functional requirements. SonarQube is an open-source platform developed by SonarSource for continuous inspection of code quality and security. Sonar does static code analysis, which provides a detailed report of bugs, code smells, vulnerabilities and code duplications.
Advantages of SonarQube:

- SonarQube analyzes branches and Pull Requests so you spot and resolve issues before you merge to main. You can optionally fail your pipeline if the Quality Gate doesn't pass.
- It supports 25+ major programming languages through built-in rulesets and can also be extended with various plugins.
- SonarQube is free for open-source projects and supports CI/CD integration with github to analyze code in repositories.
- SonarQube nicely integrates in GitHub Actions with autodetection of branches and PRs.
- Tests code quality with a pass/fail quality gate review to analyze security, reliability and maintainability of code all done in one place.
- Analyzes code security by use of security hotspots that highlight security-sensitive pieces of code that need review in order to fix vulnerabilities that may compromise code.
- Super-fast analysis with highly precise results. Get code analysis results in minutes without having to wait hours and get an indication when a vulnerability is raised on your code to fix.
- Allows for merging only safe code to repositories with a clear analysis report for your code review.



Coviduous master

September 18, 2021, 8:52 AM Version not provided

QUALITY GATE STATUS

Passed

All conditions passed.

MEASURES

New Code | Overall Code

9 Bugs — Reliability C

0 Vulnerabilities — Security A

20 Security Hotspots — 0.0% Reviewed — Security Review E

11d Debt — 485 Code Smells — Maintainability A

Figure 7: SonarQube test results

Non-functional Requirements:

- Security - The parameter defines how a system is safeguarded against deliberate and sudden attacks from internal and external sources. By using firebase JWT tokens to enforce security in our API function calls, this allowed us to test how unauthorized users would be prevented from making successful requests to the system's API.
- Scalabilty - The term refers to the degree in which any software application can expand its processing capacity to meet an increase in demand. This was tested by analysing the scale using firebase tool analytics when more users made requests to the deployed application.
- Usability - The ease with which the user can learn, operate, prepare inputs and outputs through interaction with a system. This is checked by Usability Testing which was performed to receive feedback from users on the quality and feel of using the application in order to help improve or fix parts of the system to make more user-friendly.
- Reliability - The ease with which the user can learn, operate, prepare inputs and outputs through interaction with a system. Tested with our frontend flutter tests and backend mocha tests ensuring that all frontend screens and backend rest api function calls work correctly. If all test cases pass without failure, along with good usability testing feedback, this results in our system being reliable.
- Integrability - The way in which all services and components of a system integrate with each other. Tested with mocha integration tests to show that our api function calls work correctly by integrating with our database model when api requests are made.
- Testability - Tested by ensuring component parts of our system are testable, such as testing our frontend client layer and backend application server as separate components, and that all types of testing performed in our system work and pass.