University of Regina

ENSE 477: Capstone Project

Testing Strategy

Telport: Sasktel Telecommunications Portal

Authors
Dakota Fisher
200 344 336

Quinn Bast
200 352 973

Supervisor
Dr. Yasser Morgan

Last Modified
February 24, 2019

Revision History

Revision Version	Revision Author	Revision Date
1.0	Dakota Fisher	February 24, 2019

Contents

1 Introduction 1
  1.1 Problem Statement 1

2 Automated testing 2
    2.0.1 TravisCI (In Use) 2
    2.0.2 Mocha (In Use) 3
    2.0.3 Jest (Not In Use) 3
    2.0.4 Chai (In Use) 3
    2.0.5 Sinon (In Use) 3
    2.0.6 Enzyme (In Use) 4
    2.0.7 Selenium (In Use) 4
    2.0.8 Cypress (Not In Use) 4
    2.0.9 NYC Istanbul (In Use) 4

3 User Stories 5
  3.1 Log In 5
  3.2 Log Out 5
  3.3 Check Call Logs 5
  3.4 Turn on Call Forwarding Always 6
  3.5 Turn on Call Forwarding Busy 6
  3.6 Turn on Call Forwarding Selective 6
  3.7 Turn on Call Forwarding No Answer 6
  3.8 Turn on Call Forwarding 6
  3.9 Turn on Do Not Disturb 6
  3.10 View my Profile 7
  3.11 Call a Phone Number 7
  3.12 Start a Call to a Phone Number from my Phone 7
  3.13 View Feature Access Codes 7

4 Tests 8
  4.1 Unit 8
  4.2 User Experience 8
  4.3 Component 8

List of Figures

1 TravisCI Passing Build 2

List of Tables

1 Introduction

This document outlines and defines the Testing plan and libraries utilized to provide a stable, maintainable solution. The initial problem statement and proposed activities are suggestions given by Sasktel to provide a manageable scope. To provide a refresh on the intent of the project, the problem statement has been provided in the following paragraphs. The decisions on infrastructure, aside from the Broadworks API and telephone account access are left up to us to decide.

1.1 Problem Statement

SaskTel requires a communications portal that can interwork with a Telephony Application Server and our core network to present communications and feature capabilities through a browser. This will allow for the exploration of new communications service models.

SaskTel has been pursuing the deployment of a new communications core along with the Cisco/Broadsoft Telephony Application Server branded as Broadworks. One of the drivers is to enable a richer customer experience through a converged architecture that exposes rapid development to enables new capabilities. Broadworks exposes the Application Programming Interface to access service control tools and user information. These tools and information can be used to create new communications applications or add additional value to existing applications.

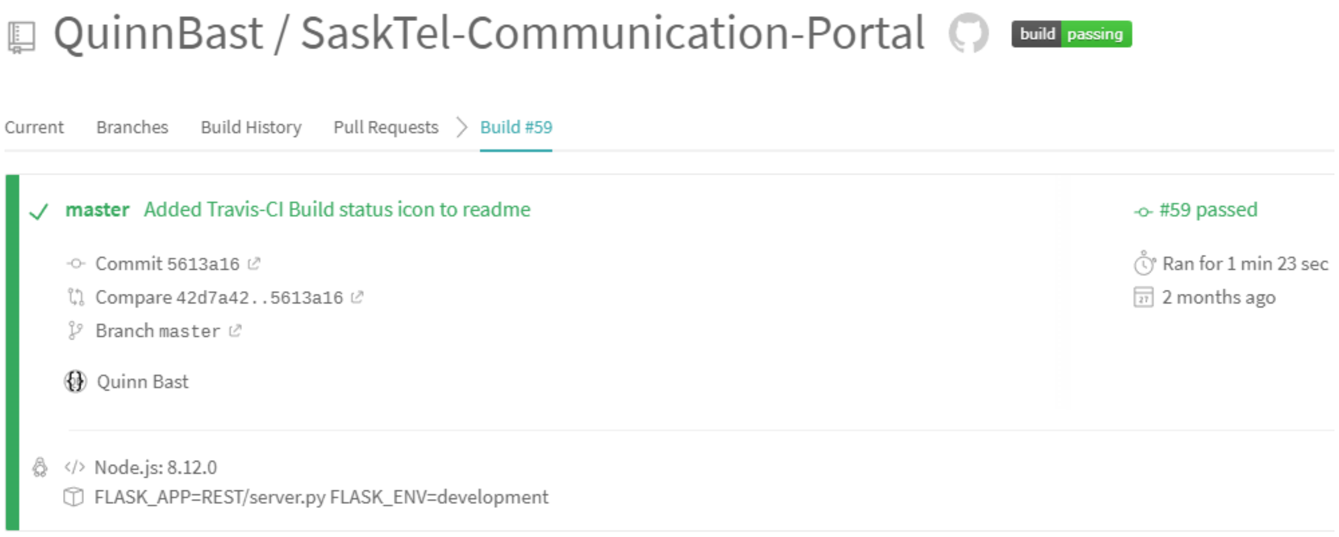
The main objectives for SaskTel is to gain exposure to new and innovative communications service experience for our customers and to promote the potential internally for aligning resources, time and effort in enabling applications.

2 Automated testing

Testing is important for the project, and as such it needs to be done continuously. Plenty of tools exist to provide the frameworks and tools to create easily executed test frameworks. Namely there are tools that provide continuous integration testing, unit testing, and user experience testing.

2.0.1 TravisCI (In Use)

TravisCI, completely referred to as Travis Continuous Integration is a cloud based testing service. It is Configured using a Yaml file to provide a configured Linux environment and run specified tests upon any merge request into the Master branch of the Github project. It provides a report, and if any tests fail it discourages the merge. There are alternatives such as CircleCI and Jenkins, but due to it's ease of use compared to Jenkins, and existing longer than CircleCI it was chosen for this project. The following figure demonstrates a passing test case, providing information on the build and test.



master Added Travis-CI Build status icon to readme

Commit 5613a16

Compare 42d7a42..5613a16

Branch master

Quinn Bast

#59 passed

Ran for 1 min 23 sec

2 months ago

Node.js: 8.12.0

FLASK_APP=REST/server.py FLASK_ENV=development

Figure 1: TravisCI Passing Build

2.0.2 Mocha (In Use)

Mocha is a robust test bench, which provides a harness for running unit tests within a node JavaScript environment. The strengths of Mocha lay within it's ability to provide a framework for any nested javascript test library. Mocha can be used as a standalone solution, or with a medley of other test frameworks. The key combination of mocha test frameworks used in this project are Chai, Enzyme, and Sinon.

2.0.3 Jest (Not In Use)

Jest is a similar solution to Mocha, and provides a very strong test framework foundation for test development. Jest is a slightly younger test library, and has slightly less community support compared to Mocha, and because of it's popularity, Mocha was chosen. Though, as test are developed, if Jest proves to be a stronger framework, it will be implemented accordingly. Jest also provides the ability to test code coverage.

2.0.4 Chai (In Use)

Chai is a strong assertion library. It's vast reliability on asserting conditions and tests is pivotal to good test development. No matter what test framework. Chai is used in combination with Mocha to provide mocking and assertions for Mocha tests.

2.0.5 Sinon (In Use)

Sinon is a spy and stubbing framework in Javascript. The strengths of Sinon for this project are it's ability to mock server API's and endpoints as well as provide controlled bad data to monitor component responses.

2.0.6 Enzyme (In Use)

The test framework for React components, Enzyme, is a powerful rendering tool to test React components at all phases of it's life-cycle. Enzyme can incorporate many aspects of Sinon, and Chai to completely emulate controlled environments for each individual component.

2.0.7 Selenium (In Use)

Selenium is a browser automation testing tool. The benefits of using Selenium tests are that it lets user experience expectations be tested during each release, to make sure that the overall functionality using a mouse and keyboard are uninhibited. Being able to test that solutions are genuinely providing the correct user output is important. Tests can be completed on components, functions and classes, but unless the web page interacts as expected there's always some level of uncertainty in the test.

2.0.8 Cypress (Not In Use)

Cypress is another browser emulation solution, similar to Selenium. The newly developed framework is a promising product, but at the time of creation of this project, it only supports testing in Chromium environments. Compared to Selenium's support for multiple browsers including Chromium, and Firefox.

2.0.9 NYC Istanbul (In Use)

Istanbul, which is known as NYC in newer iterations, is a code coverage framework which spies on test frameworks such as Mocha to monitor and output test coverage. Every file touched by Mocha tests is analysed and reported on.

3 User Stories

The following sections denote common use cases for the program that we brainstormed and consider to be requirements by usability and basic principle. The basic structure for user stories is as follows. "As a <type of user> I would like to < be able to do task> so that I can < metric improvement to life>". User stories are referenced by the number following the chapter value, which means user story 1 will refer to sub-section x.1.

User stories are important for test development as they provide ground work for the requirements and functionalities that tests should be sure to incorporate and provide solutions for. The following user stories are utilized to provide a feature driven development approach to the project, while keeping quantifiable, and testable data present to be monitored during development.

3.1 Log In

As a SaskTel Customer, I would like to be able to Log into the program so that I can use the program and check my information.

3.2 Log Out

As a SaskTel Customer, I would like to be able to Log out of the program so that I can ensure that anyone using my computer can't access my account.

3.3 Check Call Logs

As a SaskTel Customer, I would like to be able to check my call logs so that I can check when I got calls, and see who called me/when.

3.4 Turn on Call Forwarding Always

As a SaskTel Customer, I would like to be able to always forward calls sent to my phone so that I can answer them on my main phone and not worry about my secondary phone.

3.5 Turn on Call Forwarding Busy

As a SaskTel Customer, I would like to be able to forward calls when my phone is busy so that I can feel comfortable that those calling me get sent to someone who can answer.

3.6 Turn on Call Forwarding Selective

As a SaskTel Customer, I would like to be able to forward phone calls to another phone during a scheduled time so that I can so that phone calls can be forwarded when I have planned events in the way of answering my phone.

3.7 Turn on Call Forwarding No Answer

As a SaskTel Customer, I would like to be able to forward calls when I'm away from my phone so that I can answer them on my cell phone when I'm away from my land line.

3.8 Turn on Call Forwarding

As a SaskTel Customer, I would like to be able to forward phone calls to another number so that I can be sure that important calls reach me on my cell phone or work phone.

3.9 Turn on Do Not Disturb

As a SaskTel Customer, I would like to be able to decline all calls to my phone number so that I can have some time without worrying about phone calls interrupting me.

3.10 View my Profile

As a SaskTel Customer, I would like to be able to view my account information so that I can know that I am using the right account.

3.11 Call a Phone Number

As a SaskTel Customer, I would like to be able to call a phone number from my computer so that I can easily make phone calls without using my phone.

3.12 Start a Call to a Phone Number from my Phone

As a SaskTel Customer, I would like to be able to call another phone number from my phone so that I can more easily call phone numbers I find on my computer.

3.13 View Feature Access Codes

As a SaskTel Customer, I would like to be able to check what star codes I can use so that I can understand what I can do with my phone when I can’t log into the application.

4 Tests

As test cases are created, and audited to be successful test cases they're documented in this section. In order to have a record trail of the tested features and tests that are required, during the test development phase of our project, the following sections will begin to be fleshed out in further detail.

4.1 Unit

This section consists of all test cases that test JavaScript functions and classes.

4.2 User Experience

This section consists of all test cases that tests the complete website against simulated user scenarios and tests expected behaviours.

4.3 Component

This section consists of all test cases that test React Components state rendering and life-cycle correctness.