SE 3XA3: Test Plan
MAC Schedule Importer

Team 12, Team 0C
Cassandra Nicolak, nicolace
Michelle Leung, leungm16
Winnie Liang, liangw15

October 27, 2018

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	Version	Notes
2018-10-25	1.0	Rough Draft
2018-10-26	1.1	Final Draft

1 General Information

1.1 Purpose

The purpose of the test plan is to build confidence in the correctness of the implementation for the project.

1.2 Scope

The scope of the test plan is to provide a basic testing platform for the correctness and functionality of the software. The test plan includes a description of the testing procedures, tools and test cases that will be implemented to verify and validate the software for this project.

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Abbreviations**

Abbreviation	**Definition**
CSV	Comma-separated values file
URL	Uniform Resource Locator/web address
Google API	Googles Application Programming Interface

Table 3: **Table of Definitions**

Term	Definition
Mosaic	McMaster University's online administrative information system.
MacID	A McMaster University student's login account.
Scrapy	A Web scraping Python library.
Crawler	Web crawler/spider that collects information from an html webpage.
XPATH Selector	Uses path expressions to select nodes/node-sets in an XML document.
Google Calendar	Googles online free calendar connected to a gmail account.
Notepad ++	Text editor.

1.4 Overview of Document

This application will be a reimplementation of the open source Chrome extension UMD Google Calendar Schedule Importer. The reimplementation will be modified to allow students from McMaster University to export their class schedule from Mosaic and import it into their Google Calendar.

2 Plan

2.1 Software Description

The software will parse Mosaic for a user's schedule and then export it to an array. This array will then be used to import their schedule into their Google calendar. The implementation will be completed in Python 3.6.

2.2 Test Team

The members of Team 0C are Cassandra Nicolak, Michelle Leung and Winnie Liang are the test team for this project.

2.3 Automated Testing Approach

2.4 Testing Tools

The testing tools that will be used are PyTest and Pycharm. PyTest will be used to automate unit testing and Pycharm will be used for coverage checking and debugging.

2.5 Testing Schedule

See Gantt Chart at the following url: Group12-Gantt03

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 Testing for User Input

File Explorer

1. Test: FUI-01

   Type: Functional, Dynamic, Manual

   Initial State: File explorer that is used to select the html file of the schedule.

   Input: Valid URL or absolute file path.

   Output: The specified location of the html file will be saved at the URL required for the parsing component of the software.

   How test will be performed: The function that acquires the URL input from the user will check if the XPATHs are present.

2. Test: FUI-02

   Type: Functional, Dynamic

   Initial State: File explorer that is used to select the html file of the schedule.

   Input: Valid URL or absolute file path.

Output: Error message that will direct the user to the correct URL.

How test will be performed: Automated test suite will be used to check if error message is thrown correctly.

3.1.2 Testing for Exporting from Mosaic

Parsing

1. Test: FEX-01

   Type: Functional, Dynamic

   Initial State: Parsing an html document.

   Input: Absolute file path.

   Output: Successful execution of crawler.

   How test will be performed: Automated test cases will be used to handle exceptions associated with incorrect paths.

2. Test: FEX-02

   Type: Functional, Dynamic

   Initial State: Parsing an html document.

   Input: Correct XPATH selectors.

   Output: XPATH values assigned to yield variables.

   How test will be performed: Automated test cases will be used to determine if variables are assigned the correct value.

3. Test: FEX-03

   Type: Functional, Dynamic

   Initial State: Assigning parsed information to an array.

   Input: Array containing parsed information

   Output: Array with parsed information to be passed to another function.

   How test will be performed: Automated test cases will be used to determine if the array is being passed correctly.

3.1.3 Testing for Importing to Google Calendar

Input Configuration

1. CV-01
   Test Time Format Type: Functional, Dynamic

   Initial State: None

   Input: Time from CSV

   Output: Time formatted according to RFC3339

   How test will be performed: Time from the format that the parser outputs will be inputted into a function that converts it to RFC3339 format. The output of the function will be compared to the expected output.

2. CV-02
   Test Title Format Type: Functional, Dynamic

   Initial State: None

   Input: From Parser

   Output: Title for Event

   How test will be performed: Data from the parser will be the input for a function that creates the title of the event. The output of the function will be compared to the expected output.

3. CV-03

   Type: Functional, Dynamic

   Initial State: None

   Input: From Parser

   Output: Description for Event

   How test will be performed: Data from the parser will be the input for a function that creates the description of the event. The output of the function will be compared to the expected output.

3.1.4 Testing for Google API Handling

Authentication and Authorization Test The test must show that the class is able to connect to a user's Google account.

1. Test: GC-01

   Type: Functional, Manual

   Initial State: The client is not connected to their Google account.

   Input: Run function.

   Output: Access to a user's account.

   How test will be performed: The function will run for authentication and authorization. Next, the user will follow the given instructions and the user will receive text confirming authentication and authorization.

Information Retrieval and Upload Test The test must show that the class for retrieving and exporting data to a Google calendar is able to do so for:

1. Getting calendars.
2. Getting events in calendars.
3. Creating events in calendars.
4. Removing events in calendars.

1. Test: GC-02

   Type: Functional, Dynamic

   Initial State: Calendars are present in Account

   Input: None Output: List of Calendars

   How test will be performed: The test will run the function and check that a list of dictionaries and calendar ids with their titles are returned.

2. Test: GC-03

   Type: Functional, Dynamic

   Initial State: Events are in specified calendar

   Input: Calendar Id Output: List of events

   How test will be performed: The test will run the function and check that a list of dictionaries, events ids and their properties are returned.

3. Test: GC-04

   Type: Functional, Dynamic

   Initial State: List of calendar events are saved

   Input: Calendar Id, Event to be inserted. Output: New list of events

   How test will be performed: The test will save a current list of events from calendar, insert event into that calendar, get list of events from calendar, and finally compare the new list to the old list to check that the event is inserted.

4. Test: GC-05

   Type: Functional, Dynamic

   Initial State: List of calendar events are saved.

   Input: Calendar Id, event to be deleted. Output: New list of events.

   How test will be performed: The test will save the current list of events from calendar, delete event from that calendar, get list of events from calendar, and compare the new list to the old list to check that the event has been deleted.

3.2 Tests for Nonfunctional Requirements

3.2.1 Look and Feel

Accessibility

1. AC-01

   Type: Dynamic

   Initial State: None

   Input/Condition: Valid URL or absolute file path.

   Output/Result: Mosaic schedule in Google Calendar.

   How test will be performed: The software will be tested on various browsers and operating systems to ensure that the software is accessible to users with different operating systems and browsers. This includes Google Chrome, Internet Explorer, Mozilla FireFox, as well as Windows and Linux.

3.2.2 Usability

1. US-01

   Type: Manual

   Initial State: None

   Input/Condition: None

   Output/Result: User Survey Response

   How test will be performed: Users will be asked to use the software and be given a survey asking about their experience.

3.2.3 Performance

1. PF-01

   Type: Manual

   Initial State: None

   Input/Condition: None

   Output/Result: User response

   How test will be performed: Software will be manually tested and judged to see if the response time is reasonable.

3.2.4 Maintainability

1. MN-01

   Type: Static, Manual

   Initial State: None

   Input/Condition: None

   Output/Result: None

   How test will be performed: Code inspections will be conducted for the software. Additionally, the time taken to diagnose and fix problems, as well as making enhancements and adaptions to the software will be measured to ensure maintainability of the software.

3.2.5 Security

1. SC-01

   Type: Manual, Static

   Initial State: None

   Input/Condition: None

   Output/Result: Qualitative Risk Assessment Table, Impact Matrix and Effectiveness Matrix.

   How test will be performed: Risk assessments and Defect Detection Preventation (DDP) techniques will be used to identify and assess possible risks and provide optimal countermeasures.

3.3 Traceability Between Test Cases and Requirements

3.3.1 Functional Requirements

Requirement #	Description/Summary	Test ID(s)
FR01	Notify user if unable to access the Mosaic schedule.	FUI-02
FR02	Notify user of information that will be imported into Google Calendar prior to proceeding.	
FR03	Multiple uses from user.	
FR04	Simple GUI	
FR05	UI has Help option.	
FR06	Show user changes to existing Google Calendar prior to changing it.	
FR07	Ask for confirmation to confirm that the listed changes are correct.	
FR08	Display a preview of the users timetable before importing.	
FR09	Request permission to access users personal information in their Mosaic and Google account.	GC-01
FR10	Have an option that allows a user to exit.	

**Note that the functional requirements that have yet to be traced are in development.

3.3.2 Non-Functional Requirements

Requirement #	Fit-Criterion/Summary	Test ID(s)
NF01	All information will be visible and not be dependant on colour.	
NF02	Perform the import in less than five steps.	
NF03	Easy to use.	US-01
NF04	Application is easy to install.	
NF05	All information should be at a basic English reading level.	LF-01
NF06	Status indicators.	
NF07	Prompts to guide users step by step.	
NF08	Run and use the application on a desktop computer or laptop.	
NF09	Respond to a users input in a reasonable amount of time (0.5 seconds).	
NF10	Available on reliable site.	
NF11	The programming language is supported on Windows and Linux.	AC-01
NF12	The source code can be accessed by the public.	
NF13	Current developers can be contacted by the public.	MN-01
NF14	The application will not have the ability to store user data.	
NF15	Inoffensive display.	
NF16	Adheres to relevant standards and laws.	
NF17	Able to run the application with no harm to their health and safety.	

**Note that the non-functional requirements that have yet to be traced are in development.

4 Tests for Proof of Concept

4.1 Parser

Scrapy Library

1. Test: POCP-01

   Type: Functional, Dynamic, Manual

   Initial State: Parsing an html document.

   Input: Absolute file path.

   Output: Successful execution of crawler.

   How test will be performed: Visually check to see if a yield is printed in the Scrapy shell.

2. POCP-02

   Type: Functional, Dynamic, Manual

   Initial State: Parsing an html document.

   Input: Correct XPATH selectors.

   Output: XPATH values assigned to yield variables.

   How test will be performed: Visually check to see if the correct values are assigned per line.

3. POCP-03

   Type: Functional, Dynamic, Manual

   Initial State: Exporting parsing contents.

   Input: Yield containing an XPATH value.

   Output: A csv file.

   How test will be performed: Visually check if a csv file is created and contains data.

4. POCP-04

   Type: Functional, Dynamic, Manual

   Initial State: Exporting parsing contents.

   Input: Yield containing schedule information.

Output: A csv file of the yield.

How test will be performed: Visually check if the yield matches the csv file's contents.

4.2 Link to Google Calendar API

Authentication and Authorization Test Must show that the class is able to connect to a user's Google account.

1. Test: GC-01

   Type: Functional, Manual

   Initial State: The client is not connected to their Google account.

   Input: Run function.

   Output: Access to a user's account.

   How test will be performed: Run function for authentication and authorization. User follows instructions. User will receive text confirming authentication and authorization.

Information Retrieval and Upload Test Must show that the class for retrieving and exporting data to a Google calendar is able to do so for: - Getting calendars - Getting events in calendars - Creating events in calendars - Removing events in calendars

1. Test: GC-02

   Type: Functional, Dynamic

   Initial State: Calendars are present in Account

   Input: None Output: List of Calendars

   How test will be performed: Run function. Check that a list of dictionaries; calendar ids with their title are returned.

2. Test: GC-03

   Type: Functional, Dynamic

Initial State: Events are in specified calendar

Input: Calendar Id Output: List of events

How test will be performed: Run function. Check that a list of dictionaries; events ids and their properties are returned.

3. Test: GC-04

   Type: Functional, Dynamic

   Initial State:List of calendar events are saved

   Input: Calendar Id, Event to be inserted. Output: New list of events

   How test will be performed: Save current list of events from calendar, insert event into that calendar, get list of events from calendar, compare the new list to the old list to check that the event is inserted.

4. Test: GC-05

   Type: Functional, Dynamic

   Initial State: List of calendar events are saved.

   Input: Calendar Id, event to be deleted. Output: New list of events.

   How test will be performed: Save current list of events from calendar, delete event from that calendar, get list of events from calendar, compare the new list to the old list to check that the event has been deleted.

5 Comparison to Existing Implementation

Both implementations perform the task of parsing an html document that contains a user's schedule information and then imports that information into a Google calendar. Both implementations are intended for different Universities. The existing implementation is written in JavaScript, whereas the re-implementation is written in Python 3. Another difference is that since the existing implementation is written in JavaScript, it is run as a Chrome extension through a web browser. The re-implementation will be executed as a Desktop application.

6 Unit Testing Plan

Test cases will run on different operating systems, including Windows, and Linux.

6.1 Unit testing of internal functions

Every internal function will be tested with the following cases, where applicable:

1. A case where the function takes input it is expected to handle.
2. All edge cases.
3. Cases for all run-time errors and exceptions. (Whitebox)
4. If the function receives input from a user, a case where it does not receive the expected input.

6.2 Unit testing of output files

Due to the user privacy constraint, no output files will be saved locally on a user's machine. However, the output of the software will be live on a user's Google calendar. The correctness of the output will be tested by getting calendar data of a sample event from a Google calendar and compared it to what it should be.

7 Appendix

More will be added to this section in the next revision of this document.

7.1 Symbolic Parameters

Currently not applicable at this time.

7.2 Usability Survey Questions?

1. How easy were the instructions to follow?
2. Were you confused at how to use the tool? If so, where and why?
3. Are there any other feature that you would like to see?