SE 3XA3: Test Plan
MAC Schedule Importer

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

December 6, 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
2018-10-26	1.1	Rev 1

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, MAC Schedule Importer. The test plan will provide an overview of the potential test cases for the product. As well, the plan will have a guideline on the testing approaches and methods used along with the testing tools that will be used to implement the tests. Additionally, the document shall have a brief summary of the tests to verify and validate that the functional and non-functional requirements are met.

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. Each functional and non-functional requirement that is listed in the Software Specifications Requirements document will be provided a test case to ensure the requirement is met for the final version of the application. The test case will include the type of testing (manual, automated, black box, white box, etc.) as well as the input and output results expected for the test case. Furthermore, a brief description on how the test will be performed will be given for teach test case.

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
UMD	University of Maryland
XPATH	Extensible Markup Language (XML) Path Language
HTML	Hypertext Markup Language

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.
User	A person who will be using the final version of the application.
Schedule	A user's academic timetable accessed through Mosaic.
PyTest	A software test framework tailored for the Python Programming language
PyCharm	integrated development environment used in computer programming, specifically for the Python language

1.4 Overview of Document

This application will be a reimplementation of the open source Chrome extension UMD Google Calendar Schedule Importer, which imports the classschedule for students at the University of Maryland into Google Calendar. 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 through a desktop application.

2 Plan

2.1 Software Description

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

The application will first require the student to download a html fileof their schedule which they can retrieve through the applications link toMosaic. Next, the program will request the user to upload the file to theapplication where the application will display schedule information in textformat. The user will then confirm that the schedule is correct and theapplication will direct the student to log in to their Google account. Oncethe program acquires the users permission to access their Google accountand their Google Calendar, the application will import the schedule intoGoogle Calendars.

2.2 Test Team

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

However, in the future development of this project, the test team will be separate from the design and implementation team to ensure no conflict of interest when testing. Thus, different teams will be used to test the product for future implementations of the application.

2.3 Automated Testing Approach

2.4 Testing Tools

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

In the future revision of the product, other testing tools will be considered. Some testing tools that may be considered are doctest, testify, testlib and PyMock. The usage of the multiple testing tools will build confidence in the correctness of the MAC Schedule Importer application.

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: test_convert_url

   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 convert it to the proper absolute path format.

3.1.2 Testing for Exporting from Mosaic

Parsing

1. Test: test_parsed_output
   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: Automated test cases will be used to determine if variables are assigned the correct value. This can be checked manually by writing the output to a file.

2. Test: test_print_sched_err
   Type: Functional, Dynamic

   Initial State: Assigning parsed information to an list.

   Input: List containing parsed information

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

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

3. Test: test_print_sched
   Type: Functional, Dynamic

   Initial State: List containing parsed information.

   Input: None

   Output: String converted from list with parsed information.

   How test will be performed: Automated test cases will be used to determine if the list is being converted into a string properly.

4. Test: test_fetch
   Type: Functional, Dynamic

   Initial State: A flag variable set to False.

Input: Url to location of file to be parsed.

Output: String converted from list with parsed information.

How test will be performed: Automated test cases will be used to determine if the system can handle this function being executed more than once by the user.

5. Test: test_set_list
   Type: Functional, Dynamic

   Initial State: An empty list.

   Input: A list containing parsed information.

   Output: String converted from list with parsed information.

   How test will be performed: Automated test cases will be used to determine if the input list was properly assigned to the global list.

3.1.3 Testing for User Interface

Buttons

1. Test: test_fetch_button
   Type: Functional, Manual

   Initial State: Button pressed.

   Input: -

   Output: Status message in textbox.

   How test will be performed: Manual tests will be performed to test that the application prompts the user with a pop-up asking if they have selected their schedule.

2. Test: test_fetch_popup
   Type: Functional, Manual

   Initial State: Fetch Schedule button pressed.

   Input: User response (Yes/No).

Output: No - Close and do nothing. Yes - parse user's schedule.

How test will be performed: Manual tests will be performed to test that the application responds correctly depending on whether Yes or No is clicked.

3. Test: test_login_button
   Type: Functional, Manual

   Initial State: Button pressed.

   Input: -

   Output: Status message in textbox.

   How test will be performed: Manual tests will be performed to test that a user can open the Google login authorization in a browser.

4. Test: test_import_button
   Type: Functional, Manual

   Initial State: Button pressed.

   Input: -

   Output: Status message in textbox.

   How test will be performed: Manual tests will be performed to test that a user can successfully import a calendar.

5. Test: test_browse_button
   Type: Functional, Manual

   Initial State: Button pressed.

   Input: Windows file path.

   Output: Converted filepath.

   How test will be performed: Manual tests will be performed to test that a user can select a file from their computer and update the text box field.

3.1.4 Testing for Importing to Google Calendar

Input Configuration

1. test_rfc_output
   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.

3.1.5 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: test_check_perms, test_login
   Type: Functional, Manual

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

   Input: - Output: Boolean indicating that the program now has access to a user's account.

   How test will be performed: check_perms() and login() functions will be run Next, the user will follow the given instructions and the user will receive text confirming authentication and authorization. A boolean value indicating success or failure will be returned

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 the number events in calendars.
2. Creating a calendar
3. Creating events in the calendars.
4. Removing events in the calendars.

1. Test: test_create$_calType$: $Functional, Dynamic$

   Initial State: User is logged in

   Input: create$_calfunction$

Output: Google Calendars will have a new calendar named Mac Schedule after refreshing the webpage

How test will be performed:

2. Test: test_insert_events, Type: Functional, Dynamic
   Initial State: None
   Input:
   Output:
   How test will be performed:
3. Test: test_get_num_events Type: Functional, Dynamic
   Initial State: None
   Input:
   Output:
   How test will be performed:
4. Test: test_check_insertion Type: Functional, Dynamic
   Initial State: None
   Input:
   Output:
   How test will be performed:
5. Test: test_remove_new$_c alType : Functional, Dynamic$
   Initial State: None
   Input:
   Output:
   How test will be performed:
6. Test: test_push_to_schedule Type: Functional, Dynamic
   Initial State: None
   Input:
   Output:
   How test will be performed:

3.1.6 Testing for Conversion from Python program to Executable application

Due to the nature of this module, all testing will be done manually. The setup.py file contains the secret of the conversion from python to executable file. The code contains importing the packages, libraries, and modules. The simple cx_freeze command to convert the selected python program into a executable is also included in the setup.py module.

1. Test: CPE-01

   Type: Manual
   Initial State: Application is installed and ready to open
   Input: Mouse click to open the application
   Output: Application is opened.
   How the test will be performed: The tester shall open the application via a mouse click. After double clicking on the application, the application shall be opened with the main page displayed.

2. Test: CPE-02

   Type: Manual
   Initial State: Application is opened
   Input: Browse option is pressed
   Output: File explorer of the user's computer displayed.
   How the test will be performed: The tester shall press the browse button via a mouse click. After clicking on the application, the application shall be open the file explorer.

3. Test: CPE-03

   Type: Manual
   Initial State: Application is opened
   Input: Fetch option is pressed
   Output: The schedule information will be displayed in the text box on the application.

How the test will be performed: The tester shall press the fetch button via a mouse click. After clicking on the button, the application shall display all the content of the Mosaic schedule.

4. Test: CPE-04

   Type: Manual
   Initial State: Application is opened
   Input: Login option is pressed
   Output: A browser will be opened to Google's sign in page where the user can enter their credentials and give permission for the application to access their account.
   How the test will be performed: The tester shall press the login button via a mouse click. After clicking on the button, the application shall open the web browser to the Google sign-in page.

5. Test: CPE-05

   Type: Manual
   Initial State: Application is opened
   Input: Import button is pressed
   Output: Import successful will be displayed and the schedule is imported to the user's Google Calendar.
   How the test will be performed: The tester shall press the import button via a mouse click. After clicking on the button, the application shall upload the mosaic schedule information into Google Calendar.

6. Test: CPE-06

   Type: Manual
   Initial State: Application is opened
   Input: Help option is pressed
   Output: A list of the user manual and about information option is displayed.
   How the test will be performed: The tester shall press the help button

via a mouse click. After clicking on the button, the application shall display of help options.

7. Test: CPE-07

   Type: Manual
   Initial State: Application is opened
   Input: Exit button is pressed
   Output: Application is closed and the user is logged out of their Google account.
   How the test will be performed: The tester shall press the exit button via a mouse click. After clicking on the button, the application shall shut down.

3.2 Tests for Nonfunctional Requirements

3.2.1 Look and Feel

Accessibility

1. app_exe
   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.

3.2.2 Usability

1. general_ui
   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)
FR02	Notify user of information that will be imported into Google Calendar prior to proceeding.	CPE-04
FR03	Multiple uses from user.	CPE-01
FR04	Simple GUI	general_ui
FR05	UI has Help option.	CPE-06
FR06	This requirement is removed from the final version of the SRS	N/A
FR07	Ask for confirmation to confirm that the listed changes are correct.	CPE-03
FR08	This function is incorporated into FR02	N/A
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.	CPE-07
FR11	The application must confirm with the user that they can only fetch their Mosaic calendar once.	CPE-03
FR12	The application must indicate that the application has successfully imported the schedule into Google Calendar.	CPE-06
FR13	The application must indicate that the application has successfully accessed the user's Google account.	CPE-05

**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.	general_ui
NF02	Perform the import in less than five steps.	
NF03	Easy to use.	general_ui
NF04	Application is easy to install.	CPE-01
NF05	All information should be at a basic English reading level.	LF-01
NF06	Status indicators.	CPE-04
NF07	Guide users step by step.	CPE-06
NF08	Run and use the application on a desktop computer or laptop.	CPE-01
NF09	Respond to a users input in a reasonable amount of time (0.5 seconds).	PF-01
NF10	Available on reliable site.	general_ui
NF11	The programming language is supported on Windows.	CPE-01
NF12	The source code can be accessed by the public.	MN-01
NF13	Current developers can be contacted by the public.	MN-01
NF14	The application will not have the ability to store user data.	CPE-07
NF15	Inoffensive display.	general_ui
NF16	Adheres to relevant standards and laws.	general_ui
NF17	Able to run the application with no harm to their health and safety.	general_ui

**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 Windows.

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 manually compared it to what it should be.

7 Appendix

Not applicable. There are no added resources that are included in this documentation.

7.1 Symbolic Parameters

Not applicable. The product does not have symbolic parameters.

7.2 Usability Survey Questions?

1. Did you successfully use the software to import your Mosaic Schedule into Google Calendars? [Y/N + Comments]
2. Was the software easy to use? [Y/N + Comments]
3. Were the instructions clear? Did they guide you step by step? [Y/N + Comments]
4. Were the response times to your actions acceptable? [Y/N]
5. Did you run into any difficulty/problems/errors during use? [Comments]
6. Given the color scheme, was the text easy to read? [Y/N]
7. Other feedback or comments?