SE 3XA3: Test Plan: Rev 1
Plagiarism Check

310, MXQ Squad
Wang Xu , wangx147
Mrinal Kumar Tiwari , tiwarim
Qifeng Xu and xuq14

April 7, 2020

Contents

1 General Information **1**
- 1.1 Purpose . . . 1
- 1.2 Scope . . . 1
- 1.3 Acronyms, Abbreviations, and Symbols . . . 1
- 1.4 Overview of Document . . . 2

2 Plan **2**
- 2.1 Software Description . . . 2
- 2.2 Test Team . . . 3
- 2.3 Automated Testing Approach . . . 3
- 2.4 Testing Tools . . . 3
- 2.5 Testing Schedule . . . 3

3 System Test Description **4**
- 3.1 Tests for Functional Requirements . . . 4
  - 3.1.1 User registration . . . 4
  - 3.1.2 User Login . . . 7
  - 3.1.3 Plagiarism check . . . 8
  - 3.1.4 Token refill . . . 9
- 3.2 Tests for Nonfunctional Requirements . . . 10
  - 3.2.1 Usability . . . 11
  - 3.2.2 Performance . . . 11
- 3.3 Traceability Between Test Cases and Requirements . . . 13

4 Tests for Proof of Concept **13**
- 4.1 Token payment method . . . 14

5 Comparison to Existing Implementation **15**

6 Unit Testing Plan **15**
- 6.1 Unit testing of internal functions . . . 15
  - 6.1.1 Registration Testing . . . 15
  - 6.1.2 Registration Fail Testing . . . 16
  - 6.1.3 Detect Testing . . . 17
  - 6.1.4 Detect Invalid user Fail Testing . . . 17
  - 6.1.5 Detect Wrong password Fail Testing . . . 18
  - 6.1.6 Detect No Tokens Fail Testing . . . 18

6.1.7 Refill Testing 19
6.1.8 Refill Invalid user Fail Testing 20
6.1.9 Refill Invalid password Fail Testing 20
6.2 Unit testing of output files 21

7 Appendix **22**
7.1 Symbolic Parameters 22
7.1.1 Status codes 22
7.2 Usability Survey Questions? 22

List of Tables

1 **Revision History** ii
2 **Table of Abbreviations** 1
3 **Table of Definitions** 2
4 **Testing Assignemts** 3
5 **Traceability Table** 13

List of Figures

Table 1: **Revision History**

Date	Version	Notes
Feb 9, 2020	1.0	Added the requirement specification
Feb 28, 2020	1.1	Added the Test Plan
April 6, 2020	1.2	Updated the test plan

1 General Information

1.1 Purpose

The purpose of this Test Plan Document is to build the confidence that the software being implemented for the Plagiarism Check project is tested as per the standard and is correct.

1.2 Scope

The Test Plan presents a basis for testing the functionality and user interface of the implementation of Plagiarism Check. This test plan has a goal to prove that Plagiarism Check has met the requirements specified in the Requirements Document and to attach metrics to those requirements so that adherence to requirements is quantifiable and measurable.
The testing plan acts as a means to arrange testing activities. Test Plan will present what part of the software is to be tested and by what methodology. It will also act as an outline of testing methods and will outline the tools that will be utilized to test the software.

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Abbreviations**

Abbreviation	**Definition**
PoC	Proof of Concept
SRS	Software Requirements Specification
GUI	Graphical User Interface
JS	Java Script

Table 3: **Table of Definitions**

Term	**Definition**
Structural Testing	Testing derived from internal structure of the software
Functional testing	Testing derived from a description of how the program functions
Static testing	Testing that does not involve program execution
Dynamic testing	Testing which includes having test cases run during execution
Automated testing	Testing that is run automatically by software
Register page	Webpage allowing user to register for the API
Detect page	Webpage allowing user to input two texts and check similarity ratio
Refill page	Webpage allowing user to input their username, admin password and refill the number of tokens available to them

1.4 Overview of Document

The Plagiarism Check project will be an additional development to the open source Rest Api acting as its backend. The software will allow user to sign-up for the Api, log into their account, input two texts and check for the plagiarism. These user interactions would be done through a website having a *GUI*.
All of the software requirements are numbered in the Requirements Document.

2 Plan

2.1 Software Description

The software will allow users to check their input text data. The implementation will be completed in Python and JavaScript.

2.2 Test Team

The test team will consist of Qifeng Xu, Wang Xu, Mrinal Tiwari. The three main testers will split the entire testing evenly covering all types of testing.

2.3 Automated Testing Approach

PyUnit will be used to automate the unit testing. We will also be using Mocha,JS based automated testing framework. It is a dynamic framework well known for its ease of use in both frontend and backend testing.

PyTest would be used to automate unit testing of the Backend Api written in Python. PyTest framework for unit testing is easy to use, simple to write and easy to automate the testing. Once all PyTests pass, unit testing using MochaJS would be used to ensure integration of Python Backend and JS frontend integrates as expected. MochaJS is also an easy to use and automate unit testing frameowork for testing Api in JS.

2.4 Testing Tools

The Tool that will be utilized for this project is PyUnit and Mocha testing framework. We would be also using Coverage.py for code coverage metrics.

2.5 Testing Schedule

Table 4: **Testing Assignemts**

Date	**Task**	**Team Member**
March 15th 2020	Mocha testing	Qifeng Xu
March 20th 2020	Registration Testing	Mrinal Kumar Tiwari
March 30th 2020	Unit Testing	Wang Xu
March 30th 2020	Unit Testing	Qifeng Xu
March 30th 2020	Mocha Testing	Mrinal Kumar Tiwari
March 30th 2020	Unit Testing	Mrinal Kumar Tiwari
April 5th 2020	Usability	Qifeng Xu
April 5th 2020	Performance	Wang Xu

See Gantt Chart at the following url:

https://gitlab.cas.mcmaster.ca/wangx147/3xa3-project/-/blob/master/ProjectSchedule/Testing_Schedule.gan

3 System Test Description

3.1 Tests for Functional Requirements

The tests for functional requirements have been written using PyUnit and Mocha unit testing frameworks. Please refer to section 3 and section 6 both to see different tests for the functional requirements.

3.1.1 User registration

also refer to section 6.1.1 - 6.1.2

Username and password input

1. Test Case 1.1.1

   Type: Functional, Dynamic, Manual.

   Initial State: The interface asks user to input a new username and password.

   Input: A valid username and password pair.

   Output: Registration Completed successfully.

   How test will be performed: The function that registers a new user will be run through the interface with certain inputs. After it has been run, check the result of this action to see if the registration is completed successfully or there is an error.

2. Test Case 1.1.2

   Type: Functional, Dynamic, Manual.

   Initial State: The interface asks user to input a new username and password.

   Input: Invalid username (Already been registered) and valid password.

Output: Fail to complete the registration, the username already exists. Error code : 301

How test will be performed: The function that register a new user will be run through the interface with certain inputs. After it has been run, check the result of this action to see if the registration is completed successfully or there is an error.

3. ~~Test Case 1.1.3~~

   ~~Type: Functional, Dynamic, Manual.~~

   ~~Initial State: The interface asks user to input a new username and password.~~

   ~~Input: Invalid username (Invalid characters) and valid password.~~

   ~~Output: Fail to complete the registration, the username contains invalid characters.~~

   ~~How test will be performed: The function that register a new user will be run through the interface with certain inputs. After it has been run, check the result of this action to see if the registration is completed successfully or there is an error.~~

4. ~~Test Case 1.1.4~~

   ~~Type: Functional, Dynamic, Manual.~~

   ~~Initial State: The interface that asked user to input a new username and password.~~

   ~~Input: Valid username and invalid password (Too short).~~

   ~~Output: Fail to complete the registration, the password is too short, minimum of a valid password should contain 8 characters.~~

   ~~How test will be performed: The function that register a new user will be run through the interface with certain inputs. After it has been run, check the result of this action to see if the registration is completed successfully or there is an error.~~

5. Test Case 1.1.5

Type: Functional, Dynamic, Manual.

Initial State: The interface that asked user to input a new username and password.

Input: Valid username and invalid password (Invalid characters).

Output: Fail to complete the registration, the password contains invalid characters. Error code: 302

How test will be performed: The function that register a new user will be run through the interface with certain inputs. After it has been run, check the result of this action to see if the registration is completed successfully or there is an error.

6. ~~Test Case 1.1.6~~

~~Type: Functional, Dynamic, Manual.~~

~~Initial State: The interface that asked user to input a new username and password.~~

~~Input: Invalid username (Invalid characters) and invalid password (Invalid characters).~~

~~Output: Fail to complete the registration, the username contains invalid characters.~~

~~How test will be performed: The function that register a new user will be run through the interface with certain inputs. After it has been run, check the result of this action to see if the registration is completed successfully or there is an error.~~

7. Test Case 1.1.7

Type: Functional, Dynamic, Manual.

Initial State: The interface that asked user to input a new username and password.

Input: Invalid username (Already been registered) and invalid password (Invalid characters).

Output: Fail to complete the registration, the username already exists. Error: 301

How test will be performed: The function that register a new user will be run through the interface with certain inputs. After it has been run, check the result of this action to see if the registration is completed successfully or there is an error.

3.1.2 User Login

Username and password input

1. Test Case 1.2.1

   Type: Functional, Dynamic, Manual.

   Initial State: The interface that asked user to input a new username and password.

   Input: Valid username and password.

   Output: ~~Login~~ Registration successfully. Status Code: 200

   How test will be performed: The function that login with an exist username and a password will be run through the interface with certain inputs. After it has been run, check the result of this action to see if the user log in successfully or there is an error.

2. ~~Test Case 1.2.2~~

   ~~Type: Functional, Dynamic, Manual.~~

   ~~Initial State: The interface that asked user to input a new username and password.~~

   ~~Input: Invalid username (Does not exist) and password.~~

   ~~Output: Fail to login the account, the username does not exist.~~

   ~~How test will be performed: The function that login with an exist username and a password will be run through the interface with certain inputs. After it has been run, check the result of this action to see if the user log in successfully or there is an error.~~

3. Test Case 1.2.3

Type: Functional, Dynamic, Manual.

Initial State: The interface that asked user to input a new username and password.

Input: Valid username and invalid password (Wrong password).

Output: Fail to login the account, the password does not match. Error code: 302

How test will be performed: The function that login with an exist username and a password will be run through the interface with certain inputs. After it has been run, check the result of this action to see if the user log in successfully or there is an error.

3.1.3 Plagiarism check

Also refer to section 6.1.3 - 6.1.6

Text input

1. Test Case 1.3.1

   Type: Functional, Dynamic, Manual.

   Initial State: The interface that asked user to input two pieces of text.

   Input: Valid text 1 and text 2.

   Output: Similarity check complete successfully and outputs the similarity ratio. Status code: 200

   How test will be performed: The function that check for similarity ratio of two texts will be run through the interface with certain inputs. After it has been run, check the result of this action to see if the similarity check is completed successfully or there is an error.

2. ~~Test Case 1.3.2~~

   ~~Type: Functional, Dynamic, Manual.~~

   ~~Initial State: The interface that asked user to input two pieces of text.~~

   ~~Input: Invalid text 1 (Invalid characters) and valid text 2.~~

~~Output: Fail to complete similarity check and output the error message of "Invalid characters in text 1".~~

~~How test will be performed: The function that check for similarity ratio of two texts will be run through the interface with certain inputs. After it has been run, check the result of this action to see if the similarity check is completed successfully or there is an error.~~

3. ~~Test Case 1.3.3~~

   ~~Type: Functional, Dynamic, Manual.~~

   ~~Initial State: The interface that asked user to input two pieces of text.~~

   ~~Input: Valid text 1 and invalid text 2 (Invalid characters).~~

   ~~Output: Fail to complete similarity check and output the error message of "Invalid characters in text 2".~~

   ~~How test will be performed: The function that check for similarity ratio of two texts will be run through the interface with certain inputs. After it has been run, check the result of this action to see if the similarity check is completed successfully or there is an error.~~

4. ~~Test Case 1.3.4~~

   ~~Type: Functional, Dynamic, Manual.~~

   ~~Initial State: The interface that asked user to input two pieces of text.~~

   ~~Input: Invalid text 1 (Invalid characters) and invalid text 2 (Invalid characters).~~

   ~~Output: Fail to complete similarity check and output the error message of "Invalid characters in both text 1 and text 2".~~

   ~~How test will be performed: The function that check for similarity ratio of two texts will be run through the interface with certain inputs. After it has been run, check the result of this action to see if the similarity check is completed successfully or there is an error.~~

3.1.4 Token refill

Also refer to section 6.1.7 - 6.1.9

Amount input

1. Test Case 1.4.1

   Type: Functional, Dynamic, Manual.

   Initial State: The interface that asked user to input the amount of tokens they would like to refill.

   Input: Valid integer.

   Output: Token refill completes successfully, and updates the current amount of tokens that user has. Status code: 200

   How test will be performed: The function that refill tokens will be run through the interface with certain inputs. After it has been run, check the result of this action to see if the token refill is completed successfully and the token amount is updated or there is an error.

2. Test Case 1.4.2

   Type: Functional, Dynamic, Manual.

   Initial State: The interface that asked user to input the amount of tokens they would like to refill.

   Input: Invalid ~~characters~~ admin password .

   Output: Fail to refill the tokens due to invalid ~~amount input~~ admin password, and the current amount of tokens that user has does not change. Error code: 302

   How test will be performed: The function that refill tokens will be run through the interface with certain inputs. After it has been run, check the result of this action to see if the token refill is completed successfully and the token amount is updated or there is an error.

3.2 Tests for Nonfunctional Requirements

The non-functional requirements of the system would be tested using user-testing, and through tracking time needed to call a resource in the Api.

3.2.1 Usability

1. Test Case 2.1.1

   Type: Functional, Dynamic, Manual

   Initial State: The interface ask the user to input two texts with 90 percent similarity.

   Input/Condition: two texts with 90 percent similarity

   Output/Result: 90 percent plagiarism

   How test will be performed: User could input two texts which have 90 percent similarity. The actual result should match the one we expected. This test would also include a user testing technique testing how easy it is for a user to use this website and doing user-acceptance testing.

2. Test Case 2.1.2

   Type: Functional, Dynamic, Manual

   Initial State: The interface ask the user to input two texts which are not English.

   Input: Two Chinese texts

   Output: failed to check the similarity

   How test will be performed: User could input two texts which are not English, for example Chinese, to see if it works. We supposed that the software could only check the English texts.

3.2.2 Performance

To check performance of our website, each Api call would be timed and compared to each implementation thus optimizing the end result.

1. Test Case 2.2.1

   Type: Functional, Dynamic, ~~Manual~~ Automated

   Initial State: The interface ask the user to input two texts to check the similarity

Input/Condition: ~~Totally the same texts input~~ Two duplicate texts

Output/Result: ~~100 percent plagiarism~~ Response time: should be less than 1 second

How test will be performed: User could input two texts which are totally the same to see if ~~the result is correct. If the input data are the same, the expected result should be 100 percent plagiarism.~~ the response time is less than 1 seconds or not.

2. Test Case 2.2.2

   Type: Functional, Dynamic, ~~Manual~~ Automated

   Initial State: The interface ask the user to input two texts to check the similarity

   Input: ~~Totally the different texts input~~ Two different texts.

   Output: ~~0 percent plagiarism~~ Response time should be less than 2 seconds

   How test will be performed: User could input two texts which are totally the different to see if ~~the result is correct. If the input data are different, the expected result should be 0 percent plagiarism.~~ the response time is less than 2 seconds or not.

3. ~~Test Case 2.2.3~~

   ~~Type: Functional, Dynamic, Manual~~

   ~~Initial State: The interface ask the user to input two texts to check the similarity~~

   ~~Input: two texts are partially similar~~

   ~~Output: the output should be a specific number according to the similarity check~~

   ~~How test will be performed: User could input two texts which are partially similar. The text could be simple enough so that the testers could calculate the similarity by themselves, and to see if the result is correct.~~

4. **Test Case 2.2.4**

Type: Functional, Dynamic, Automated

Initial State: The interface ask the user to input username and password to register

Input: username and password

Output: Response time is less than 1 second.

How test will be performed: User could input username and password to see if the response time is less than 2 seconds or not.

5. **Test Case 2.2.5**

Type: Functional, Dynamic, Automated

Initial State: The interface ask the user to input admin password and refill amount to refill available tokens

Input: admin password and refill amount

Output: Response time is less than 1 second.

How test will be performed: User could input admin password and refill amount to see if the response time is less than 2 seconds or not.

3.3 Traceability Between Test Cases and Requirements

Table 5: **Traceability Table**

Requirements	Test cases
FR1: The website shall allow user registration using username and password.	Test case 1.1.1, 1.1.2,1.1.7, ~~1.1.3, 1.14, 1.1.6~~ 6.1.1 - 6.1.2
FR2: The website shall allow user to login using unique username and password.	Test case 1.2.1, 1.2.3, ~~1.2.2~~
FR4: User shall be able to enter the two texts it wants to compare for plagiarism.	Test case 1.3.1 ~~1.1.2- 1.3.4~~ , 6.1.3 - 6.1.6
FR7: The website shall allow a user to refill his tokens whenever required.	Test case 1.4.1 - 1.4.2, 6.1.7 - 6.1.9
NFR2.1: All text displayed by interface shall in English with Canadian spelling.	Test case 2.1.1 - 2.1.2
NFR3: The result shall be displayed immediately after user submits the text.	Test case 2.2.1, ~~2.2.3~~, 2.2.4, 2.2.5

4 Tests for Proof of Concept

Issue: How to get the website up live?
Solution: By buying a domain name and thus giving a placeholder online that could be used later to host our fully functional website. http://plagiarismcheck-com.stackstaging.com

Issue: How to get the Api online that could be called through the website?
Solution: This got resolved by deploying the Api on AWS EC2 instance that is available online round the clock everyday. ec2-18-224-246-224.us-east-2.compute.amazonaws.com:8000/

4.1 Token payment method

Payment method selection and information input

1. Test case 4.1

   Type: Functional, Dynamic, Manual.

   Initial State: The interface that asked user to select payment method.

   Input: The selection of one payment method in the list.

   Output: Return a interface with selected payment method and the information it requires.

   How test will be performed: The function that select a payment method will be run through the interface with certain inputs. After it has been run, check the result of this action to see if the payment information it asked is corresponding to the selected one. (eg. When user select credit card as the payment method, the interface will ask for the credit card information instead of Paypal account.)

2. Test case 4.2

   Type: Functional, Dynamic, Manual.

   Initial State: The interface that asked user to enter payment information.

   Input: The correct information of a credit card.

   Output: The payment is completed successfully, the tokens are successfully purchased and added into user's account.

   How test will be performed: The function that input payment information will be run through the interface with certain inputs. After it

has been run, check the result of this action to see if the payment is completed successfully and if the current amount of tokens is updated.

5 Comparison to Existing Implementation

n/a

6 Unit Testing Plan

The PyUnit and Mocha framework will be used to do unit testing for the project's backend.

6.1 Unit testing of internal functions

In order to create unit tests for the internal functions, all the methods that return value can be tested. This will involve giving required inputs to the methods and checking the output.Based on the input and the expected output from the method, a number of unit tests can be created.
Unit tests will include tests that include normal case input as well as boundary and edge cases that supposedly would throw exceptions. We would not be importing any stubs or drivers for testing. Anything that needs to be imported to either implement a functionality or test a function would be already imported in the individual classes.
We would be using coverage metrics to determine how much of our code has been covered. Our goal remains to cover maximum portion of the code(more than 80%) to ensure the adequate tests for all functions.

6.1.1 Registration Testing

This will be a unit test using PyUnit (Mocha) to validate correctness of register.

Test Factors Involved: Correctness

Initial State: User is on Register page of the website

Inputs: username, password

Outputs: status code: 200 OK
message: User has been successfully registered

Schedule: This test regards the registration of the user and therefore will be necessary for the Final Demonstration 1

Methodology: This test will be conducted by checking the status code and also if the user is registered for the Api after calling the Register resource in the Restful Backend.

Tests For: This test validates section **FR1**, of SRS document.

6.1.2 Registration Fail Testing

This will be a unit test using PyUnit (Mocha) to validate correctness of register.

Test Factors Involved: Correctness

Initial State: User is on Register page of the website

Inputs: username, password

Outputs: status code: 301 User exists
message: A user already exists with this username

Schedule: This test regards the registration of the user and therefore will be necessary for the Final Demonstration 1

Methodology: This test will be conducted by checking the status code and message after calling the Register resource in the Restful Backend.

Tests For: This test validates section **FR1**, of SRS document.

6.1.3 Detect Testing

This will be a unit test using PyUnit (Mocha) to validate correctness of Detect resource.

Test Factors Involved: Correctness

Initial State: User is on *Detect* page of the website

Inputs: username, password, two string texts

Outputs: status code: 200 OK
message: Similarity ratio calculated
similarity ratio: a decimal number

Schedule: This test regards the calculation of the similarity between the two texts and therefore will be necessary for the Final Demonstration 1

Methodology: This test will be conducted by checking the status code and the similarity ratio after calling the Detect resource in the Restful Backend.

Tests For: This test validates section **FR4**, of SRS document.

6.1.4 Detect Invalid user Fail Testing

This will be a unit test using PyUnit (Mocha) to validate correctness of Detect resource.

Test Factors Involved: Correctness

Initial State: User is on *Detect* page of the website

Inputs: username, password, two string texts

Outputs: status code: 301
message: User does not exist

Schedule: This test regards the calculation of the similarity between the

two texts and therefore will be necessary for the Final Demonstration 1

Methodology: This test will be conducted by checking the status code and the message after calling the Detect resource in the Restful Backend.

Tests For: This test validates section **FR4**, of SRS document.

6.1.5 Detect Wrong password Fail Testing

This will be a unit test using PyUnit (Mocha) to validate correctness of Detect resource.

Test Factors Involved: Correctness

Initial State: User is on *Detect* page of the website

Inputs: username, password, two string texts

Outputs: status code: 302
message: Invalid password

Schedule: This test regards the calculation of the similarity between the two texts and therefore will be necessary for the Final Demonstration 1

Methodology: This test will be conducted by checking the status code and the message after calling the Detect resource in the Restful Backend.

Tests For: This test validates section **FR4**, of SRS document.

6.1.6 Detect No Tokens Fail Testing

This will be a unit test using PyUnit (Mocha) to validate correctness of Detect resource.

Test Factors Involved: Correctness

Initial State: User is on *Detect* page of the website

Inputs: username, password, two string texts

Outputs: status code: 303
message: Out of tokens, please refill

Schedule: This test regards the calculation of the similarity between the two texts and therefore will be necessary for the Final Demonstration 1

Methodology: This test will be conducted by checking the status code and the message after calling the Detect resource in the Restful Backend.

Tests For: This test validates section **FR4**, of SRS document.

6.1.7 Refill Testing

This will be a unit test using PyUnit (Mocha) to validate correctness of Refill resource.

Test Factors Involved: Correctness

Initial State: User is on *Refill* page of the website

Inputs: username, password, refill amount

Outputs: status code: 200 OK
message: Tokens refilled successfully

Schedule: This test regards the refill of tokens which is required to use the Api and therefore will be necessary for the Final Demonstration 1

Methodology: This test will be conducted by checking the status code and the message after calling the Refill resource in the Restful Backend.

Tests For: This test validates section **FR7**, of SRS document.

6.1.8 Refill Invalid user Fail Testing

This will be a unit test using PyUnit (Mocha) to validate correctness of Detect resource.

Test Factors Involved: Correctness

Initial State: User is on *Refill* page of the website

Inputs: username, password, refill amount

Outputs: status code: 301
message: User does not exist

Schedule: This test regards the refill of tokens which is required to use the Api and therefore will be necessary for the Final Demonstration 1

Methodology: This test will be conducted by checking the status code and the message after calling the Refill resource in the Restful Backend.

Tests For: This test validates section **FR7**, of SRS document.

6.1.9 Refill Invalid password Fail Testing

This will be a unit test using PyUnit (Mocha) to validate correctness of Detect resource.

Test Factors Involved: Correctness

Initial State: User is on *Refill* page of the website

Inputs: username, password, refill amount

Outputs: status code: 302
message: Invalid Admin password

Schedule: This test regards the refill of tokens which is required to use the Api and therefore will be necessary for the Final Demonstration 1

Methodology: This test will be conducted by checking the status code and the message after calling the Refill resource in the Restful Backend.

Tests For: This test validates section **FR7**, of SRS document.

6.2 Unit testing of output files

Our project does not generate any output files and thus this section is not applicable to our project.

7 Appendix

7.1 Symbolic Parameters

7.1.1 Status codes

200 : This status code is raised if the API call is successful and Api performs as desired
301 : This status code is raised when registering a user twice or if the username does not match
302 : This status code is raised when the input password is invalid/wrong
303 : This status code is raised when user runs out of tokens needed to call the Api

7.2 Usability Survey Questions?

When a user has finished using the website and clicks the close button, the website would prompt a survey to be filled to gain feedback on the loose ends of the website and improve on it.
Questions :

1. How likely are you to recommend this website to a friend?
2. Are you having trouble finding anything on this website?
3. Is there anything missing on the page?
4. How did you experience compared to your expectation?
5. How easy was it to use this website? Did you have any problems?
6. What is the feature you wish our website had?
7. Is there any features you do not find helpful?