SE 3XA3: Test Plan
The Resume Shotgun

Team 5, Proper Mars Tribe
Gavin Jameson, jamesong
Jeremy Langner, langnerj
Sam Gorman, gormans

March 11, 2022

Contents

List of Tables

List of Figures

1 General Information

1.1 Purpose

Testing is crucial for our product, as it deals with user information and parties other than the user. We need to ensure that the user's information is not misused through unintentional release of the information, or incorrect representation of the user. Furthermore, errors will waste the time of employers as they will have to deal with any data sent to them, however briefly, correct or not.

1.2 Scope

Since the output of our product is not self-contained, manual testing will need to be done frequently for processes occurring nearer to the end. There are also limited tests we can do for robustness, since as mentioned in previous documents, there are ethical concerns with submitting incorrect or incomplete applications.

1.3 Acronyms, Abbreviations, and Symbols

Table 1: **Table of Definitions**

Term	**Definition**
Glassdoor	Online job board. One of the sites from which links are pulled and applied to.
Indeed	Online job board. One of the sites from which links are pulled and applied to.

1.4 Overview of Document

This document outlines the test cases and procedures, as well as their rationale, that will be used for the Resume Shotgun. This document also serves as a way to trace requirements, how they are being implemented, and what modifications have been made to them after the product has been partially implemented.

2 Plan

2.1 Software Description

The Resume Shotgun is a tool to automatically aggregate job postings relevant to the user, fill in required fields with user data, and submit applications.

2.2 Test Team

As there is a lot of manual testing to be done, which is very time consuming, the development team is also the testing team. This testing team consists of Gavin Jameson, Jeremy Langner, and Sam Gorman.

2.3 Automated Testing Approach

We will be using Python's unittest module for automatic white box testing of the internal modules.

2.4 Testing Tools

As mentioned above, we will use unittest for automated testing. If for whatever reason there are tests that unittest can not handle, pytest will be used as a supplement.

2.5 Testing Schedule

See Gantt Chart.

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 Profile Information

Note: The userProfile module being tested assumes inputs are of the correct type. The user never directly inputs values to the functions being tested (they are validated first by the menu, which is also tested), so we do not need to worry about the user inputting incorrect types.

Setters/Getters

1. Test-PS-1: Set Keywords

   Type: Functional, Dynamic

   Initial State: keywords == []

   Input: ["pointless"]

   Final State: ["pointless"]

   How test will be performed: Default value is [], use setter with input, getter for ensuring final state.

2. Test-PS-2: Add/Remove Keywords

   Type: Functional, Dynamic

   Initial State: keywords == ["pointless"]

   Input: ["pointless", "redundant"]

   Final State: ["redundant"]

   How test will be performed: Use Test-PS-1 to set state, use setter with input and toggle flag set to true, getter for ensuring final state.

3. Test-PS-3: Set Site

   Type: Functional, Dynamic

   Initial State: site = "glassdoor"

   Input: 1

   Output: True

   Final State: site = "indeed"

   How test will be performed: Default value is "glassdoor," use setter with input, getter for ensuring final state. Ensure output from setter is True.

4. Test-PS-4: Set Site Invalid

Type: Functional, Dynamic

Initial State: site = "glassdoor"

Input: 5173

Output: False

Final State: site = "glassdoor"

How test will be performed: Default value is "glassdoor," use setter with input, getter for ensuring final state. Ensure output from setter is False.

5. Test-PS-5: Set First Name

   Type: Functional, Dynamic

   Initial State: firstName = ""

   Input: "The"

   Final State: firstName = "The"

   How test will be performed: Default value is "", use setter with input, getter for ensuring final state.

6. Test-PS-6: Set Last Name

   Type: Functional, Dynamic

   Initial State: lastName = ""

   Input: "User"

   Final State: lastName = "User"

   How test will be performed: Default value is "", use setter with input, getter for ensuring final state.

7. Test-PS-7: Set Email

   Type: Functional, Dynamic

   Initial State: email = ""

   Input: "email@domain.web"

Output: True

Final State: email = "email@domain.web"

How test will be performed: Default value is "", use setter with input, getter for ensuring final state. Ensure output from setter is True.

8. Test-PS-8: Set Email Invalid

   Type: Functional, Dynamic

   Initial State: email = ""

   Input: "oopsies..."

   Output: False

   Final State: email = ""

   How test will be performed: Default value is "", use setter with input, getter for ensuring final state. Ensure output from setter is False.

9. Test-PS-9: Set Phone

   Type: Functional, Dynamic

   Initial State: phone = ""

   Input: 1234567890

   Output: True

   Final State: phone = 1234567890

   How test will be performed: Default value is "", use setter with input, getter for ensuring final state. Ensure output from setter is True.

10. Test-PS-10: Set Phone Invalid

    Type: Functional, Dynamic

    Initial State: phone = ""

    Input: 1234

    Output: False

    Final State: phone = ""

How test will be performed: Default value is "", use setter with input, getter for ensuring final state. Ensure output from setter is False.

11. Test-PS-11: Set Organisation

    Type: Functional, Dynamic

    Initial State: organisation = ""

    Input: "3XA3"

    Final State: organisation = "3XA3"

    How test will be performed: Default value is "", use setter with input, getter for ensuring final state.

12. Test-PS-12: Set Resume Path

    Type: Functional, Dynamic

    Initial State: resumePath = ""

    Input: "resume.pdf" (assuming it is in the same directory as testing file)

    Output: True

    Final State: resumePath = "PATH/TO/FILE/resume.pdf"

    How test will be performed: Default value is "", use setter with input, getter for ensuring final state. Ensure output from setter is False.

13. Test-PS-13: Set Resume Path Invalid

    Type: Functional, Dynamic

    Initial State: resumePath = ""

    Input: "thisdoesnotexist.pdf"

    Output: False

    Final State: resumePath = ""

    How test will be performed: Default value is "", use setter with input, getter for ensuring final state. Ensure output from setter is False.

14. Test-PS-14: Set Socials

    Type: Functional, Dynamic

    Initial State: socials = []

    Input: ["account"]

    Final State: socials = ["account"]

    How test will be performed: Default value is [], use setter with input, getter for ensuring final state.

15. Test-PS-15: Add/Remove Socials

    Type: Functional, Dynamic

    Initial State: socials = ["linkedin"]

    Input: ["linkedin", "github"]

    Final State: ["github"]

    How test will be performed: Use Test-PS-14 to set state, use setter with input and toggle flag set to true, getter for ensuring final state.

16. Test-PS-16: Set Location

    Type: Functional, Dynamic

    Initial State: location = []

    Input: ["Hamilton", Ontario", "Canada"]

    Output: True

    Final State: location = ["Hamilton", Ontario", "Canada"]

    How test will be performed: Default value is [], use setter with input, getter for ensuring final state. Ensure output from setter is True.

17. Test-PS-17: Set Location Invalid

    Type: Functional, Dynamic

    Initial State: location = []

Input: ["earth"]

Output: False

Final State: location = []

How test will be performed: Default value is [], use setter with input, getter for ensuring final state. Ensure output from setter is False.

18. Test-PS-18: Set Grad

    Type: Functional, Dynamic

    Initial State: grad = []

    Input: [5, 2024]

    Output: True

    Final State: grad = [5, 2024]

    How test will be performed: Default value is [], use setter with input, getter for ensuring final state. Ensure output from setter is True.

19. Test-PS-19: Set Grad Invalid

    Type: Functional, Dynamic

    Initial State: grad = []

    Input: [13, 2024]

    Output: False

    Final State: grad = []

    How test will be performed: Default value is [], use setter with input, getter for ensuring final state. Ensure output from setter is False.

20. Test-PS-20: Set University

    Type: Functional, Dynamic

    Initial State: university = ""

    Input: "S-cool"

    Final State: university = "S-cool"

How test will be performed: Default value is [], use setter with input, getter for ensuring final state.

21. Test-PS-21: Get Dictionary

    Type: Functional, Dynamic

    Initial State: Default userProfile

    Input: N/A

    Output: Dictionary containing every variable in profile

    How test will be performed: Default values set on initialisation, use getter for output and ensure each variable is contained in the dictionary with the value expected.

Saving/Loading

1. Test-PF-1: Save Profile

   Type: Functional, Dynamic, Manual

   Initial State: Every variable something other than default

   Input: N/A

   Output: Variables in testprofile.yaml

   How test will be performed: Use PS Tests to set state, use saveProfile method to get values from file, Test-PS-21 to ensure final state.

2. Test-PF-2: Load Profile

   Type: Functional, Dynamic

   Initial State: Default userProfile

   Input: test2profile.yaml (existing file)

   Final State: Variables from test2profile.yaml

   How test will be performed: Default values set on initialisation, use loadProfile method to get values from file, Test-PS-21 to ensure final state.

3. Test-PF-3: Load Profile Broken

   Type: Functional, Dynamic

   Initial State: Default userProfile

   Input: test3profile.yaml (existing file, missing some profile values)

   Final State: Variables from test3profile.yaml, otherwise default value

   How test will be performed: Default values set on initialisation, use loadProfile method to get values from file, Test-PS-21 to ensure final state.

4. Test-PF-4: Load Profile Missing

   Type: Functional, Dynamic

   Initial State: Default userProfile

   Input: testnoprofile.yaml (not a file)

   Final State: Default userProfile

   How test will be performed: Default values set on initialisation, use loadProfile method, Test-PS-21 to ensure final state.

5. Test-PF-5: Load Profile Unreadable

   Type: Functional, Dynamic

   Initial State: Default userProfile

   Input: test0profile.yaml (empty file)

   Final State: Default userProfile

   How test will be performed: Default values set on initialisation, use loadProfile method, Test-PS-21 to ensure final state.

3.1.2 User Interactions

Note: In this section, "Output" is what is verified manually.

Main Menu

1. Test-MM-1: To Resume

   Type: Functional, Dynamic, Manual

   Initial State: menu = 0

   Input: 1

   Output: Visual for Menu/Resume

   Final State: menu = 1

   How test will be performed: Main menu opens by default, send input using keyboard in application window, visually check menu has updated.

2. Test-MM-2: To Personal Info

   Type: Functional, Dynamic, Manual

   Initial State: menu = 0

   Input: 2

   Output: Visual for Menu/Info

   Final State: menu = 2

   How test will be performed: Main menu opens by default, send input using keyboard in application window, visually check menu has updated.

3. Test-MM-3: To Keywords

   Type: Functional, Dynamic, Manual

   Initial State: menu = 0

   Input: 3

   Output: Visual for Menu/Keywords

   Final State: menu = 3

   How test will be performed: Main menu opens by default, send input using keyboard in application window, visually check menu has updated.

4. Test-MM-4: To Job Preferences

   Type: Functional, Dynamic, Manual

   Initial State: menu = 0

   Input: 4

   Output: Visual for Menu/Jobs

   Final State: menu = 4

   How test will be performed: Main menu opens by default, send input using keyboard in application window, visually check menu has updated.

5. Test-MM-5: To Sites

   Type: Functional, Dynamic, Manual

   Initial State: menu = 0

   Input: 5

   Output: Visual for Menu/Sites

   Final State: menu = 5

   How test will be performed: Main menu opens by default, send input using keyboard in application window, visually check menu has updated.

6. Test-MM-6: Exit

   Type: Functional, Dynamic, Manual

   Initial State: menu = 0

   Input: 0

   Output: Menu closed

   How test will be performed: Main menu opens by default, send input using keyboard in application window, visually check menu has updated.

7. Test-MM-7: Exit

   Type: Functional, Dynamic, Manual

   Initial State: menu = 0

   Input: "string"

   Output: Same menu with error message

   How test will be performed: Main menu opens by default, send input using keyboard in application window, visually check menu has not changed, but sent an error message.

Resume Menu

1. Test-MR-1: New File

   Type: Functional, Dynamic, Manual

   Initial State: menu = 1

   How test will be performed: Use Test-MM-1 to set menu state, perform Test-PS-12 through keyboard in application window, visually check menu has updated.

2. Test-MR-2: New File Invalid

   Type: Functional, Dynamic, Manual

   Initial State: menu = 1

   How test will be performed: Use Test-MM-1 to set menu state, perform Test-PS-13 through keyboard in application window, visually check menu has not changed, but sent an error message.

3. Test-MR-3: Go Back

   Type: Functional, Dynamic, Manual

   Initial State: menu = 1

   Input: 0

   Output: Visual for Menu (main)

How test will be performed: Use Test-MM-1 to set menu state, send input using keyboard in application window, visually check menu has updated.

Personal Info, Keywords, Job Preferences, and Sites Menus

Each sub menu (and their sub menus, if applicable) listed above can be tested nearly identically to either MM Tests or MR Tests, depending on if it has sub menus or not, respectively. Slight variations on inputs and initial states to the tests may be required depending on what inputs are requested from the sub menus or setter it is using (as tested with PS Tests).

If a string is not a wanted input for sub menus with access to setters (i.e. input consisting of something other than 0-9), Test-MM-7 shall be used in addition to MR Tests.

3.1.3 Website Interactions

Indeed

1. Test-SKL1: Search Keyword and Location with Indeed

   Type: Unit

   Initial State: User profile is updated and menu is able to select to run Indeed module.

   Input: Two strings, first one representing keyword for job, second represents the city location

   Output: Indeed URL page with results.

   How test will be performed: The tester will manually search selected keyword and location, record the Indeed URL page based on search results and compare that URL string with the actually output URL string.

2. Test-GJ1: Get Jobs

   Type: Unit

   Initial State: None.

   Input: Indeed URL with keyword/location search results.

Output: Integer representing the amount of pages to be searched based on the search results.

How test will be performed: The tester will manually find an Indeed URL with at least one page result. They will then count the number of pages displayed at the top or bottom of the webpage and compare such count to the output of the getPages() function.

3. Test-GP1: Get Pages

   Type: Unit

   Initial State: An Indeed URL search result page with ¿ 0 jobs results is available.

   Input: Indeed URL with keyword/location search results.

   Output: List of Tuples that contain the job title, company and link to indeed job posting.

   How test will be performed: The tester will manually find an Indeed URL with at least one job result. They will then create a list of tuples that contain job title, company, and link. Then the tester will run the getJobs() function and assert that every item in their manual list is in the actual output from the function.

3.2 Tests for Nonfunctional Requirements

3.2.1 Look and Feel

1. Test-NF-LF-1

   Type: Manual Dynamic

   Initial State: None

   Input/Condition: None

   Output/Result: Menu displayed through command line is determined to be easily readable by tester.

   How test will be performed: The application will be launched from the preferred shell of the tester. Success will be determined by if text is properly formatted and easy to understand.

3.2.2 Performance

1. Test-NF-PF-1

   Type: Non-Functional Dynamic

   Initial State: On menu with all user information input correctly.

   Input/Condition: None.

   Output/Result: A correct list of at least 100 links aggregated in less than 5 minutes.

   How test will be performed: The "link scraper" will be run with while tracking how many links it has pulled. Once it has collected 100, the program will be terminated and execution time will be compared.

3.2.3 Operational and Environmental

Operating Systems

1. Test-NF-OEO-1

   Type: Manual Dynamic

   Initial State: None

   Input/Condition: None

   Output/Result: Program Executes Successfully

   How test will be performed: The program will be executed on a Windows machine. A test will be considered successful if the program starts and menus are able to be navigated correctly.

2. Test-NF-OEO-2

   Type: Manual Dynamic

   Initial State: None

   Input/Condition: None

   Output/Result: Program Executes Successfully

How test will be performed: The program will be executed on a IOS machine. A test will be considered successful if the program starts and menus are able to be navigated correctly.

3. Test-NF-OEO-3

   Type: Manual Dynamic

   Initial State: None

   Input/Condition: None

   Output/Result: Program Executes Successfully

   How test will be performed: The program will be executed on a Linux machine. A test will be considered successful if the program starts and menus are able to be navigated correctly.

Browser

1. Test-NF-OEB-1

   Type: Manual Dynamic

   Initial State: On menu with all relevant user information input.

   Input/Condition: Desired browser: Chrome.

   Output/Result: Correct list of links from desired site.

   How test will be performed: The "link scraping" portion of the program will be run with a Chrome driver to launch the desired sites from a Google Chrome tab.

2. Test-NF-OEB-2

   Type: Manual Dynamic

   Initial State: On menu with all relevant user information input.

   Input/Condition: Desired browser: Safari.

   Output/Result: Correct list of links from desired site.

How test will be performed: The "link scraping" portion of the program will be run with a Safari driver to launch the desired sites from a Safari tab.

3. Test-NF-OEB-3

   Type: Manual Dynamic

   Initial State: On menu with all relevant user information input.

   Input/Condition: Desired browser: Firefox.

   Output/Result: Correct list of links from desired site.

   How test will be performed: The "link scraping" portion of the program will be run with a Firefox driver to launch the desired sites from a Firefox tab.

3.2.4 Security

1. Test-NF-SC-1

   Type: Manual Static

   Initial State: Program has been executed at least once and a password input by user.

   Input/Condition: None.

   Output/Result: Password is omitted from saved user information file.

   How test will be performed: The file which stores user information will be checked after program execution to ensure that the password was not stored permanently.

2. Test-NF-SC-2

   Type: Dynamic

   Initial State: Program is running and on menu.

   Input/Condition: User has input password.

   Output/Result: The file where the password is stored has an encrypted string rather than the plain text password.

How test will be performed: To prevent any password being exposed during run time, it will be encrypted during use. The save file will be checked to make sure the correct encrypted string is present.

3.3 Traceability Between Test Cases and Requirements

Functional Requirements:

- FR1 tested through Personal Info Menu from MO Tests using PS Tests
- FR2 tested through MR Tests
- FR3 tested through PF Tests
- FR4 tested through Sites Menu from MO Tests using Test-PS-3, Test-PS-4
- FR5 tested through Keywords Menu from MO Tests using Test-PS-1, Test-PS-2
- FR6 tested through Test-MM-6, Test-PS-21
- FR7 tested through Test-GJ1,
- FR8 tested through Test-GJ1,

Non-Functional Requirements:

- LF1 tested through Test-NF-LF-1
- PR1 tested through Test-NF-PF-1
- OE1 tested through Test-NF-OEO-X
- OE2 tested through Test-NF-OEB-X
- SR1 tested through Test-NF-SC-X

Note: Certain non-functional requirements, such as Humanity, Maintainability, Cultural, and Legal were not suited to have tests designed for them, and as such are excluded here.

4 Tests for Proof of Concept

For the proof of concept tests, manual versions of tests covering saving and loading profiles, and menu interactions with resumes, sites and keywords had been conducted. From section 3.1, that corresponds to:

1. Test-PS-1, Test-PS-2, Test-PS-3, Test-PS-4, Test-PS-12, Test-PS-13, Test-PS-21
2. PF Tests
3. MM Tests
4. MR Tests
5. Keywords and Sites Menus from MO Tests

5 Comparison to Existing Implementation

The existing implementation did not feature a testing plan, execution or report available to the public. There was a single functional call that was commented out that was meant to run getURLs() which returns the URLs based on search results. There was no strategy, framework or design plan however for such test thus is was a dynamic manual test.

6 Unit Testing Plan

6.1 Unit testing of internal functions

Unit testing is a useful testing strategy for individually testing "units" which are typically functions or methods within certain modules. It is important to test these methods individually the ensure they work and to gain confidence in testing the larger interactions and systems that use such functions/methods to operate. Unit tests will be conducted by inputting expected inputs, boundary cases and invalid cases thus ensuring proper testing coverage is achieved. Based on these inputs we have to manually determine the expected output and make a comparison to the actual output vs to the expected output using the testing framework. Unittest, a common python

framework, will be used to test such internal functions since it has plenty of functionality, different options to explore comparison methods, and plenty of online support.

6.2 Unit testing of output files

The only output file that will require to be tested is a .yaml file that contains the user profile information and outcomes of running the application. This will be tested by comparing the yaml file contents with the expected output. This would be done by creating several different profiles then running the application that produces the yaml file and testing it outcomes.

7 Appendix

7.1 Symbolic Parameters

N/A

7.2 Usability Survey Questions?

N/A

8 Revision History

Table 2: **Revision History**

Date	**Version**	**Notes**
Mar 8	0.1	Planning format and tests that had not yet been decided
Mar 10	0.2	Mostly filled in
Mar 11	1.0	Finished filling in