1)Main Page

Testing Documentation

Version	Date	Author(s)	Summary of Changes
	April 8 2023	Matthew	Finished validation testing
	April 8 2023	Foster	Finished integration testing
	April 8 2023	Marc	Finished system testing
	April 8 2023	Jane	Finished unit testing

2)Introduction

Overview

University buildings can often be confusing to navigate. Navigating outdoors has become much easier through the use of smart phones, GPS, and mapping services, which allow people to locate buildings with ease. However, many of these services do not do a good job of showing maps of interior spaces, especially with buildings that have multiple floors. Western provides the public with floor plans of all the buildings on campus, but they are only PDFs with no metadata, meaning they are not easy to navigate.

The main purpose of this program is to use the maps provided by Western to allow users to explore the floor layout as well as search for specific points on the map. This includes searching for specific rooms in buildings, locating points of interest in a building, browsing through the available maps, and allowing the user to create their own points of interest. The program will also have an editing tool to allow other developers to create and edit map metadata.

This document outlines all of our test cases to asses the quality of the program, and all of the results. The test plan consists of the following tests: unit testing, integration testing, validation testing, and system testing.

Objectives

The objectives of the project are to:

- Apply principles of software engineering towards a real-world problem
- Work with, interpreting and following a detailed specification provided
- Create models of requirements and design from a specification
- Implement all the requirements and designs in Java, while dealing with decisions that were made earlier in the design process
- Create a user-friendly user-interface
- Write robust and efficient code
- Write well-documented code in Java using best practices
- Reflect on good and bad design decisions that have been made throughout the project

References

- Project Specification
- Requirements Documentation
- Design Documentation

3.1) Unit Testing

List of JUnit tests implemented for the project:

- BuildingsPageTest.java
- BuildingTest.java
- CampusTest.java
- EditPOITest.java
- FloorPageTest.java
- FloorTest.java
- Group41Test.java
- InsertPOITest.java
- LayerTest.java
- LoginTest.java
- POITest.java
- ProfileTest.java
- RegistrationTest.java
- WeatherAPITest.java

BuildingsPageTest.java:

This is a JUnit test is for the class BuildingsPage, which is a GUI component of the project. This test includes setUpClass(), tearDownClass(), setUp(), tearDown(), and testMain().

- testMain(): creates a BuildingsPage object and checks if the class is displayable. The test fails if not.

BuildingTest.java

This is a JUnit test is for class Buildings, which is an object that stores two variables LinkedList<Floor> floorList and int buildingID. This test includes setUpClass(), tearDownClass(), setUp(), tearDown(), testGetBuildingID(), testSetBuildingID(), testGetFloorList(), and testAddFloorList().

- testGetBuildingID(): creates an Building object with an expected BuildingID, variable result saves the return from getBuildingID, and compares if they are equal. The test fails if not.
- testSetBuildingID(): creates an Building object, sets an expected BuildingID, gets BuildingID and compares if they are equal. The test fails if not.
- testGetFloorList(): creates an Building object with an expected LinkedList<Floor>, variable result saves the return from getFloorList, and compares if they are equal. The test fails if not.
- testAddFloorList(): creates an Building object, adds an expected Floor object to LinkedList<Floor>, gets Floor object from the linked list, and compares if they are equal. The test fails if not.

CampusTest.java

This is a JUnit test is for class Campus, which is an object that stores two variables LinkedList<Building> buildingList. This test includes setUpClass(), tearDownClass(), setUp(), tearDown(), testSetBuilding(), and testGetBuildingList().

- testSetBuilding(): creates an Campus object with an expected LinkedList<Building> buildingList, sets the buildingList to campus object, saves the return in variable results, and compare if they are equal. The test fails if not.
- testGetBuildingList(): creates an Campus object with an expected LinkedList<Building> buildingList, saves the return from getBuildingList in variable results, and compare if they are equal. The test fails if not.

EditPOITest.java

This is a JUnit test is for class EditPOI, which is a GUI component of the project. This test includes setUpClass(), tearDownClass(), setUp(), tearDown(), and testMain().

- testMain(): creates an EditPOI object with a new POI object and checks if the class is displayable. The test fails if not.

FloorPageTest.java

This is a JUnit test is for class FloorPage, which is a GUI component of the project. This test includes setUpClass(), tearDownClass(), setUp(), tearDown(), and testFloorChange(), testPaintMarker(), and testMain().

- testFloorChange(): creates a FloorPage object with a compelete Campus object as an input, and change the Floor using floorChange. The test fails if any expection is caught.
- testPaintMarker(): creates a FloorPage object with a compelete Campus object as an input, and calls the method paintMarker. The test fails if any expection is caught.
- testMain(): creates a FloorPage object with a compelete Campus object as an input, and checks if the class is displayable. The test fails if not.

FloorTest.java

This is a JUnit test is for class FloorTest, which is an object that stores three variables int floorID, int buildingID, and LinkedList<Layer> layerList. This test includes setUpClass(), tearDownClass(), setUp(), tearDown(), testGetFloorID(), testSetFloorID(), testGetBuildingID(), testSetBuildingID(), testGetLayerList(), and testAddLayerList().

- testGetFloorID(): creates a Floor object with an expected FloorID, variable result saves the return from getFloorID, and compares if they are equal. The test fails if not.
- testSetFloorID(): creates a Floor object, sets an expected FloorID, gets FloorID, and compares if they are equal. The test fails if not.
- testGetBuildingID(): creates a Floor object with an expected BuildingID, variable result saves the return from getBuildingID, and compares if they are equal. The test fails if not.
- testSetBuildingID(): creates a Floor object, sets an expected BuildingID, gets BuildingID, and compares if they are equal. The test fails if not.
- testGetLayerList(): creates a Floor object with an expected LinkedList<Layer>, variable result saves the return from getLayerList, and compares if they are equal. The test fails if not.
- testAddLayerList(): creates a Floor object, adds an expected Layer object to LinkedList<Layer>, gets Layer object from the linked list, and compares if they are equal. The test fails if not.

Group41Test.java

This is a JUnit test is for class Group41, which is the main class of the project. This test includes setUpClass(), tearDownClass(), setUp(), tearDown(), testSearchPOI(), testSearchProfile(), and testMain(), .

- testSearchPOI(): creates an Group41 object, searches for a POI using method searchPOI, and compares with the expected results. The test fails if not.
- testSearchProfile(): creates an Group41 object, searches for a Profile object using method searchProfile, and compares with the expected results. The test fails if not.
- testMain(): creates an Group41 and checks if any exception occures during the process. The test fails if any expection is caught.

InsertPOITest.java

This is a JUnit test is for class InsertPOI, which is a GUI component of the project. This test includes setUpClass(), tearDownClass(), setUp(), tearDown(), testFloorChange(), testClickToInsert(), and testMain().

- testFloorChange(): creates an InsertPOI object and changes the int FloorID using floorChange. The test fails if any expection is caught.
- testClickToInsert(): creates an InsertPOI object and tries the method clickToInsert. The test fails if any expection is caught.
- testMain(): creates an InsertPOI object and checks if the class is displayable. The test fails if not.

LayerTest.java

This is a JUnit test is for class Layer, which is an object that stores two variables String layerType, LinkedList<POI> POIList. This test includes setUpClass(), tearDownClass(), setUp(), tearDown(), testAddPOI(), testGetPOIList(), testGetLayerType(), and testSetLayerType().

- testAddPOI(): creates a Layer object and a POI object, adds a POI to a Layer object using method addPOI, gets POI the object stored in the LinkedList<POI> of Layer Object, and compares if they are equal. The test fails if not.
- testGetPOIList(): creates a Layer object with an expected LinkedList<POI>, gets LinkedList<POI>, and compares if they are equal. The test fails if not.
- testGetLayerType(): creates a Layer object with an expected LayerType, gets LayerType, and compares if they are equal. The test fails if not.
- testSetLayerType(): creates a Layer object, sets the LayerType, gets LayerType, and compares if they are equal. The test fails if not.

LoginTest.java

This is a JUnit test is for class Login, which is a GUI component of the project. This test includes setUpClass(), tearDownClass(), setUp(), tearDown(), and testMain().

- testMain(): creates a Login object and checks if the class is displayable. The test fails if not.

POITest.java

This is a JUnit test is for class POI, which is an object that stores variables String name, String description, String layerType, int buildingID, String room,int floorID, int[] coordinates, and boolean userCreated. This test includes setUpClass(), tearDownClass(), setUp(), tearDown(), testSetName(), testSetDescription(), testSetCoords(), testIsUserCreated(), testGetLayerType(), testGetName(), testGetRoom(), testGetCoords(), testGetBuildingID(), testGetFloorID(), and testGetDescription().

- testSetName(): creates a POI object, sets name, gets name, and compares if they are equal. The test fails if not.
- testSetDescription(): creates a POI object, sets description, gets description, and compares if they are equal. The test fails if not.
- testSetCoords(): creates a POI object, sets coords, gets coords, and compares if they are equal. The test fails if not.
- testIsUserCreated(): creates an POI object with a boolean variable userCreated=true, calls IsUserCreated. The test fails if false.
- testGetLayerType(): creates a POI object with complete input values, gets LayerType, and compares if they are equal. The test fails if not.
- testGetName(): creates a POI object with complete input values, gets name, and compares if they are equal. The test fails if not.
- testGetRoom(): creates a POI object with complete input values, gets room, and compares if they are equal. The test fails if not.
- testGetCoords(): creates a POI object with complete input values, gets coords, and compares if they are equal. The test fails if not.
- testGetBuildingID(): creates a POI object with complete input values, gets BuildingID, and compares if they are equal. The test fails if not.
- testGetFloorID(): creates a POI object with complete input values, gets FloorID, and compares if they are equal. The test fails if not.
- testGetDescription(): creates a POI object with complete input values, gets description, and compares if they are equal. The test fails if not.

ProfileTest.java

This is a JUnit test is for class Profile, which is an object that stores variables String user, String pass, boolean edit, LinkedList<POI> userPOIsList, and LinkedList<String> userFavsList. This test includes setUpClass(), tearDownClass(), setUp(), tearDown(), testAddUserPOIs(), testAddUserFavs(), testRemoveUserPOIs(), testRemoveUserFavs(), testSearchUserPOIs(), testEditFavName(), testSaveData(), testSetUsername(), testSetPassword(), testSetEditor(), testIsEditor(), testGetUsername(), testGetPassword(), and testGetEditor().

- testAddUserPOIs(): creates a Profile object and adds a POI object. The test fails if any exception is caught.
- testAddUserFavs(): creates a Profile object and adds a UserFav string. The test fails if any exception is caught.
- testRemoveUserPOIs(): creates a Profile object and removes a POI object from the Profile object. The test fails if any exception is caught.
- testRemoveUserFavs(): creates a Profile object, adds expected UserFav, gets UserFav, and compares if they are equal. The test fails if not.
- testSearchUserPOIs(): creates a Profile object, adds expected POI, gets POI, and compares if they are equal. The test fails if not.
- testEditFavName(): creates a Profile object, edit UserFav name, gets UserFav name, and compares if they are equal. The test fails if not.
- testSaveData(): creates a Profile object that includes the call for a method saveData. The test fails if any exception is caught during the process.
- testSetUsername(): creates a Profile object, sets username, gets username, and compares if they are equal. The test fails if not.
- testSetPassword(): creates a Profile object, sets password, gets password, and compares if they are equal. The test fails if not.
- testSetEditor(): creates a Profile object, sets boolean Editor=true, and gets an Editor. The test fails if boolean Editor is false.
- testIsEditor(): creates an Profile object with a boolean variable Editor=true, and calls IsEditor. The test fails if false.
- testGetUsername(): creates a Profile object with an expected username input, gets username, and compares if they are equal. The test fails if not.
- testGetPassword(): creates a Profile object with an expected password input, gets password, and compares if they are equal. The test fails if not.
- testGetEditor(): creates a Profile object with a boolean Editor=false as default, and gets boolean Editor. The test fails if boolean Editor is true.

RegistrationTest.java

This is a JUnit test is for class Registration, which is a GUI component of the project. This test includes setUpClass(), tearDownClass(), setUp(), tearDown(), and testMain().

- testMain(): creates a Registration object and checks for the variavle western that's stored in Registration class. The variable western should exist without any assigned value. The test fails if not.

WeatherAPITest.java

This is a JUnit test is for class WeatherAPI, which is a class that uses openweathermap api to get the current weather at Weatern Campus. This test includes setUpClass(), tearDownClass(), setUp(), tearDown(), testGetIcon(), and testGetWeather().

- testGetIcon(): creates a new WeatherAPI object and saves the returned string from calling a method getIcon() to a string result. The test fails if result is null.
- testGetWeather(): creates a new WeatherAPI object and saves the returned string from calling a method getWeather() to a string result. The test fails if result is null.

3.2) Integration Testing

3.2.1) Brief

We are approaching our integration testing using a **top-down approach** with **depth-first integration**. That is, we will begin at our main function Group41.java and integrate other modules completely as we descend the control hierarchy.

3.2.2) Test Details

Test Case Name:	**Integration testing phase 1 - Main function**
Test Case Description	This test seeks to run the main function (Group41.java) which utilizes our class structure to load a Campus structure. This test resembles a **white-box** or **structural test**.
Test Steps	1. Ensure main is in an environment with access to POIFile.json and our class structure 2. Execute Group41.java 3. Ensure launch is successful, and check contents of Campus structure created (called western)
Pre-Requisites	**Group41.java, POIFile.json, Campus.java, Building.java, Floor.java, Layer.java, POI.java**
Expected Results	Group41.java successfully runs and builds a Campus structure containing all POIs within POIFile.json
Test Category	Integration Test
Requirement	Group41.java must be successful in initializing the information required for the execution of the rest of the project
Automation	Run manually while altering the project environment
Date Run	8 April 2023
Pass/Fail	Pass
Test Results	Campus object was successfully created, and contents were correct
Remarks	Everything in this test works as expected

Test Case Name:	**Integration testing phase 2 - Login and Registration**
Test Case Description	This test seeks to run the main function (Group41.java) which will call and execute Login.java which has access to Registration.java. This test is again a **white-box** or **structural test**.
Test Steps	1. Execute Group41.java which after creating **western** (our Campus structure) will launch Login.java 2. Ensure that Login.java launches, is able to create a Profile object with information from ProfileFile.json 3. Ensure that Registration.java can launch, is able to create a **new** Profile object which is stored in ProfileFile.json
Pre-Requisites	Group41.java, POIFile.json, Campus.java, Building.java, Floor.java, Layer.java, POI.java, **ProfileFile.json, Login.java, Registration.java**
Expected Results	All executed programs must be able to run within the given environment. Login.java and Registration.java are successful in their function in regards to creating a Profile object, reading/writing to ProfileFile.json
Test Category	Integration Test
Requirement	Group41.java, Login.java, and Registration.java must be successful in initializing and creating the information required for the execution of the rest of the project
Automation	Run manually while altering the project environment
Date Run	8 April 2023
Pass/Fail	Pass
Test Results	Campus, Profile objects were successfully read/written and created
Remarks	Everything in this test works as expected

Test Case Name:	**Integration testing phase 3 - BuildingsPage, FloorPage, and WeatherAPI**
Test Case Description	This test seeks to run the main function (Group41.java) which will call and execute Login.java which has access to Registration.java. Once a Profile is loaded, BuildingsPage, FloorPage, and WeatherAPI will be run in succession. This test is again a **white-box** or **structural test**.
Test Steps	1. Execute Group41.java which after creating **western** (our Campus structure) will launch Login.java 2. Ensure that Login.java launches, is able to create a Profile object with information from ProfileFile.json 3. Ensure that Registration.java can launch, is able to create a **new** Profile object which is stored in ProfileFile.json 4. Launch BuildingsPage.java which utilizes information from **western** 5. Launch FloorPage.java, which also utilizes information from **western** 6. Launch WeatherAPI.java from FloorPage.java
Pre-Requisites	Group41.java, POIFile.json, Campus.java, Building.java, Floor.java, Layer.java, POI.java, ProfileFile.json, Login.java, Registration.java, **BuildingsPage.java, FloorPage.java, WeatherAPI.java**
Expected Results	All executed programs must be able to run within the given environment. Login.java and Registration.java are successful in their function in regards to creating a Profile object, reading/writing to ProfileFile.json. BuildingsPage.java, FloorPage.java, and WeatherAPI.java can successfully run within the environment with access to created Campus and Profile objects
Test Category	Integration Test
Requirement	Group41.java, Login.java, and Registration.java must be successful in initializing and creating the information required for the execution of the rest of the project. BuildingsPage.java, FloorPage.java, and WeatherAPI.java must successfully run in this project space
Automation	Run manually while altering the project environment
Date Run	8 April 2023
Pass /Fail	Pass
Test Results	Group41.java, Login.java, and Registration.java were successful in running within the given environment
Remarks	Everything in this test works as expected

Test Case Name:	**Integration testing phase 4 - Complete Integration**
Test Case Description	This test seeks to run the main function (Group41.java) which will call and execute Login.java which has access to Registration.java. Once a Profile is loaded, BuildingsPage, FloorPage, and WeatherAPI will be run in succession. InsertPOI and EditPOI will then be executed. This test is again a **white-box** or **structural test**.
Test Steps	1. Execute Group41.java which after creating **western** (our Campus structure) will launch Login.java 2. Ensure that Login.java launches, is able to create a Profile object with information from ProfileFile.json 3. Ensure that Registration.java can launch, is able to create a **new** Profile object which is stored in ProfileFile.json 4. Launch BuildingsPage.java which utilizes information from **western** 5. Launch FloorPage.java, which also utilizes information from **western** 6. Launch WeatherAPI.java from FloorPage.java 7. Launch InsertPOI.java from FloorPage.java, ensure it can create a POI to be stored in POIFile.json. Return to FloorPage.java 8. Launch EditPOI.java from FloorPage.java, ensure it can edit a POI to be stored in POIFile.json
Pre-Requisites	Group41.java, POIFile.json, Campus.java, Building.java, Floor.java, Layer.java, POI.java, ProfileFile.json, Login.java, Registration.java, BuildingsPage.java, FloorPage.java, WeatherAPI.java, **InsertPOI.java, EditPOI.java**
Expected Results	All executed programs must be able to run within the given environment. Login.java and Registration.java are successful in their function in regards to creating a Profile object, reading/writing to ProfileFile.json. BuildingsPage.java, FloorPage.java, and WeatherAPI.java can successfully run within the environment with access to created Campus and Profile objects. InsertPOI.java and EditPOI.java can successfully run and save to POIFile.json and ProfileFile.json
Test Category	Integration Test

Requirement	Group41.java, Login.java, and Registration.java must be successful in initializing and creating the information required for the execution of the rest of the project. BuildingsPage.java, FloorPage.java, and WeatherAPI.java must successfully run in this project space. InsertPOI.java and EditPOI must successfully run and save to POIFile.json and ProfileFile.json
Automation	Run manually while altering the project environment
Date Run	8 April 2023
Pass /Fail	Pass
Test Results	All files were successfully run within the environment
Remarks	Everything in this test works as expected

3.3) Validation Testing

Test Case Name:	Browsing Maps
Test Case Description:	User should be able to browse through all available maps.
Test Steps:	Select a map, and then select a different map, until all maps have been selected.
Pre-Requisites:	Launch the program and login as any user.
Expected Results:	The currently displayed map successfully changes to the chosen map when selecting a new one.
Test Category:	Validation test
Requirement:	Browsing maps functionality
Automation:	Manually run
Date Run:	April 8, 2023
Pass/Fail:	Pass
Test Results:	Currently displayed map successfully changed when selecting a new one.
Remarks:	Everything in this test worked as expected.

Test Case Name:	Scrolling Maps
Test Case Description:	User should be able to scroll through the selected map if it is too large to fit on the screen.
Test Steps:	Select a map and scroll.
Pre-Requisites:	Launch the program and login as any user.
Expected Results:	The current map should be able to scroll up down left and right.
Test Category:	Validation test
Requirement:	Scrolling maps functionality
Automation:	Manually run
Date Run:	April 8, 2023
Pass/Fail:	Pass
Test Results:	Currently displayed map successfully scrolls in all directions.
Remarks:	Everything in this test worked as expected.

Test Case Name:	Displaying Layers
Test Case Description:	User should be able the select multiple layers to be displayed/hidden
Test Steps:	Select a map, and then select "Display/Hide Layers" and select the layers you want displaued.
Pre-Requisites:	Launch the program and login as any user.
Expected Results:	The selected layers are displayed on the map.
Test Category:	Validation test
Requirement:	Displaying layers functionality
Automation:	Manually run
Date Run:	April 8, 2023
Pass/Fail:	Pass
Test Results:	The selected layer types display on the selected map.
Remarks:	Everything in this test worked as expected.

Test Case Name:	Searching
Test Case Description:	User should be able to search through points of interest.
Test Steps:	Click the search bar and type the name of a point of interest.
Pre-Requisites:	Launch the program and login as any user.
Expected Results:	The points of interest matching the search keywords are displayed.
Test Category:	Validation test
Requirement:	Searching functionality
Automation:	Manually run
Date Run:	April 8, 2023
Pass/Fail:	Pass
Test Results:	A list of the related points of interest based on the keywords are displayed.
Remarks:	Everything in this test worked as expected.

Test Case Name:	Marking POI as favourite
Test Case Description:	User should be able to mark a POI as a favourite and see all favourited POIs.
Test Steps:	Click the "favourite" box when creating a POI.
Pre-Requisites:	Launch the program, login as any user, click "insert POI"
Expected Results:	The POI is correctly added to your favourites.
Test Category:	Validation test
Requirement:	Favourites
Automation:	Manually run
Date Run:	April 8, 2023
Pass/Fail:	Fail
Test Results:	POI is successfully added to favourites, however there is currently no option to display all favourited POIs.
Remarks:	First half of the test case passed, second half failed.

Test Case Name:	Creating POIs
Test Case Description:	User should be able to create a new point of interest.
Test Steps:	Click "Insert POI", enter the required information, and click insert.
Pre-Requisites:	Launch the program and login as any user.
Expected Results:	POI is correctly added.
Test Category:	Validation test
Requirement:	User-created POIs
Automation:	Manually run
Date Run:	April 8, 2023
Pass/Fail:	Pass
Test Results:	POI is successfully created with the information entered by the user.
Remarks:	Everything in this test worked as expected.

Test Case Name:	Saving User Data

Test Case Description:	The program should save data (login info and user-created POIs) so that the user can use the data again when closing and re-opening the program.
Test Steps:	Enter some data (create username and password, create POI), close the application, and re-open.
Pre-Requisites:	Launch the program, create a username and password, create a POI.
Expected Results:	The info is successfully saved in a JSON file and can be used in another session after the user closes the program.
Test Category:	Validation test
Requirement:	Persistent Data
Automation:	Manually run
Date Run:	April 8, 2023
Pass/Fail:	Pass
Test Results:	All data (username, password, user-created POIs) are successfully saved, and can be used when re-launching the program.
Remarks:	Everything in this test worked as expected.

Test Case Name:	**Exit/Close**
Test Case Description:	User should be able to exit the program, and if there is unsaved work, it should warn the user to save before closing the application.
Test Steps:	Close the program by clicking file > save and close or save and logout.
Pre-Requisites:	Launch the program and login as any user.
Expected Results:	Program closes and warns user if there is any unsaved work.
Test Category:	Validation test
Requirement:	Exit/Close functionality
Automation:	Manually run
Date Run:	April 8, 2023
Pass/Fail:	Fail
Test Results:	Program closes properly, but does not warn the user if there are any unsaved changes.
Remarks:	Needs to warn the user about unsaved changes when closing.

Test Case Name:	**User Help**
Test Case Description:	User should be able to view a help screen for useful information about how to use the program.
Test Steps:	Select "Help" at the top of the window.
Pre-Requisites:	Launch the program and login as any user.
Expected Results:	A popup window displays with information on how to use the application.
Test Category:	Validation test
Requirement:	User Help
Automation:	Manually run
Date Run:	April 8, 2023
Pass/Fail:	Pass
Test Results:	A popup window displays when selecting the help button, and it contains useful information on how to use the program.
Remarks:	Everything in this test worked as expected.

Test Case Name:	**Editing Built-In POIs**
Test Case Description:	Admin should be able to edit built-in POIs.

Test Steps:	Select edit POI and change the data.
Pre-Requisites:	Launch the program and login as an admin.
Expected Results:	The new data for the POI is saved.
Test Category:	Validation test
Requirement:	Editing Tool/Mode
Automation:	Manually run
Date Run:	April 8, 2023
Pass/Fail:	Fail
Test Results:	POI information does not update properly.
Remarks:	Editing POIs not currently working.

Test Case Name:	**Displaying Weather**
Test Case Description:	User should be able to see the current weather in London, Ontario.
Test Steps:	Click the weather button on the top of the window
Pre-Requisites:	Launch the program and login as any user.
Expected Results:	The current weather is displayed for the user.
Test Category:	Validation test
Requirement:	Current weather
Automation:	Manually run
Date Run:	April 8, 2023
Pass/Fail:	Pass
Test Results:	Successfully displays the current weather in London, Ontario.
Remarks:	Everything in this test worked as expected.

3.4) System Testing

Test Case Name	Making sure the software runs on Windows 10
Test Case Description	The user should be able to run the software on a Windows 10 system.
Test Steps	1) Launch the program using a Windows 10 system. 2) Check if the menu bar, menu bar items (and their shortcuts), buttons, scrollbars, and text fields visible on each application frame are functioning and useable. 3) Close the program.
Pre-Requisites	Have a Windows 10 system with the software application installed and ready to use.
Expected Results	The user is able to use the menu bar, menu bar items (and their shortcuts), buttons, scrollbars, and text fields visible on each application frame on a Windows 10 system.
Test Category	System Testing.
Requirement	All menu bars, menu bar items (and their shortcuts), buttons, scrollbars, and text fields must be visible on each application within a Windows 10 system.
Automation	Manually run
Date Run	April 8th 2023
Pass/Fail	Pass
Test Results	All the menu bars, menu bar items (and their shortcuts), buttons, scrollbars, and text fields visible on each application frame were functioning and useable on a Windows 10 system.
Remarks	The software worked as expected on a Windows 10 system.

Test Case Name	Making sure the software saves/loads data properly.
Test Case Description	The user should be able to register a new account in the system. Moreover, the user should be able to see if they can insert their own personal POIs and view them later. Moreover, their POIs should be saved onto the program so that when they close and reopen the program, their POI is still visible.
Test Steps	1) Launch the program. 2) Create a new account. 3) Login with that new account. 4) Create a new POI. 5) Close the application. 6) Reopen the application. 7) Try to log in and see if POI was saved.
Pre-Requisites	Have the software installed and ready to use.
Expected Results	The user is able to create a new account, log in with that new account, create a new POI, save that POI, close the application, and return to it with their personal changes saved and loaded.
Test Category	System Testing.
Requirement	Persistent Data
Automation	Manually run
Date Run	April 8th 2023

Pass/Fail	Fail
Test Results	Created a new account, successfully logged in, created a new POI, saved that POI, logged out, logged in again, and saw new POI was saved. However, if the user chooses to close the application instead of logging out, and then reopen the application again, and log in, the new POI won't be visible.
Remarks	The software didn't work as expected.

Test Case Name	Making sure the software allows the user to edit POIs.
Test Case Description	The user should be able to edit their saved POIs.
Test Steps	1) Launch the program. 2) Edit a previous POI. 3) Make changes to the POI. 4) Save those changes. 5) Log out. 6) Login again. 7) See if previous changes to POI were saved.
Pre-Requisites	Have the software installed and ready to use and have personal POIs inserted into the program.
Expected Results	The user is able to access a newly created POI, make changes to it, log out, log in again, and see recent changes to POI were saved.
Test Category	System Testing.
Requirement	Editing Tool
Automation	Manually run
Date Run	April 8th 2023
Pass/Fail	Fail
Test Results	Created a POI, saved it, logged out, logged in again, accessed the newly created POI, made changes through the use of the edit button, saved the changes, and found out the POI had been deleted.
Remarks	The software didn't work as expected. The POI was deleted instead of being updated.

Test Case Name	Making sure the software runs as a whole
Test Case Description	The user should be able to edit their saved POIs.
Test Steps	1) Launch the program. 2) Within each application test all functionality. Click on every button, enter text of different length and no text into every field, and test edge cases in regards to POI and Profile saving/loading
Pre-Requisites	Have the software installed and ready to use and have personal POIs inserted into the program.
Expected Results	The user is able to utilize all functions the application provides without failure.
Test Category	System Testing.
Requirement	A complete program

Automation	Manually run
Date Run	April 8th 2023
Pass/Fail	Fail
Test Results	Issues were found saving/loading POIs, utilizing the favourite feature (incomplete list, failed to scroll to the correct POI coordinates).
Remarks	The software didn't work as expected. A POI was deleted instead of being updated, and the favourites functionality was incomplete.

4)Summary

This document outlined all of the tests performed on the program to asses the quality of the application. The types of tests performed were unit testing, integration testing, validation testing and system testing.

Term	Definition
POI	Point of interest
Unit testing	Testing the source code (components, classes) as outlined in the design documentation
Integration testing	Testing how all of the units come together
Validation testing	Testing all of the functional requirements as outlined in the requirements documentation
System testing	The entire program is tested as a whole