CS2212 Group Project Specification

Version 3.0

Winter Session 2023

---

1. Overview

Navigating a university can be a daunting task, with buildings often spread over a large sprawling campus. Moving around outdoors has been made much easier through smart phones, GPS, and mapping services like Google Maps that can at least help people locate buildings without much difficulty. Such services, however, often don't do an adequate job of interior spaces where map data is incomplete or difficult to work with, as buildings are typically composed of multiple floors with different layouts that are hard to represent in a single, flat, 2D map.

At Western, to assist people with navigating interior spaces, the university has made the floor plans of all of its buildings available to the public. While the primary use case is to identify accessible features of the campus and its buildings to those in need, the maps available through this service can be useful to anyone wanting to locate a particular room in a particular building. Unfortunately, the maps are simply provided as PDF files with no metadata that makes them readily searchable or easy to use. Furthermore, while a layer of accessibility is baked into each map, there are opportunities for other points of interest and other useful data to be layered onto the maps that are not explored.

The main purpose of this project is to create an application that leverages the maps made available by Western to allow users to search and explore its interior spaces. At a high level, your application will allow users to search for rooms in buildings, locate points of interest in a building, browse through the maps provided, and create and save their own personal points of interest. Your application must also have an accompanying editing tool to facilitate the creation and editing of map metadata by developers for the application.

This document outlines the general specifications for this project. Separate documents will be posted to the OWL assignments tab to provide particular specifications for each stage and milestone of the project.

2. Objectives

This project is designed to give you experience in:

- applying the principles of software engineering towards a real-world problem
- working with, interpreting, and following a detailed specification provided to you
- creating models of requirements and design from such a specification
- implementing your design in Java and having to deal with decisions made earlier in the design process
- creating graphical, user-facing content and applications
- writing robust and efficient code
- write good, clean, well-documented Java code that adheres to best practices
- reflecting on good/bad design decisions made over the course of the project

The project is intended to give you some freedom in design and programming to explore the subject matter, while still providing solid direction towards reaching a specified goal.

3. Working with the Maps

The maps provided by Western are made available in a PDF format. There are a number of ways of displaying PDF files in Java, and doing so should allow you to keep the maps as vector data that scale nicely to your display space (this allows for smooth zooming in and out of the PDF maps and the drawings and text remain crisp and clean). That said, Java might handle things more readily if you were to convert the PDF maps into a collection of image files in JPEG or PNG formats. As bitmaps, they would lose their scaling properties, but they might be more flexible and easier to work with in some cases. If you would like to do the conversions, there is a wide variety of tools and packages that can do this for you. The choice of format for the map data is up to you and should be an early part of your team's planning. **Converting the maps to PNG or JPEG files is recommended to avoid using a third-party library.**

You will see that the maps have accessibility data baked into them. In a way, that means that the maps have an "accessibility layer" that is always on. While it might be nicer to have that as separate data so the "accessibility layer" can be toggled on and off as points of interest in the map, that is fine for your application. That said, I know that some of you would likely rather have it as a separate layer. It is possible to edit the PDF files and remove the accessibility data, if that is your preference. I have found that programs such as Inkscape (free software) Adobe Illustrator (installed in some undergrad computer science labs) do a good job of editing these PDF files (there are likely other tools that can work for this as well). You can also "paint" the data out with a variety of packages, but that is likely to be more time consuming. Again, doing this is completely optional but possible.

As Western has a few dozen buildings, there are a lot of maps available for us to use. As it would be a lengthy task to create a full set of metadata for every map for every building, you only need to concern yourself with **maps for Middlesex College** and **two other buildings of your choice**. You can always produce metadata for more buildings too, and it should be much easier once you have your editing tool complete, but it is completely optional.

You may wish to put some thought into the choice of buildings as this will impact the number of points of interest (POIs) you will be required to create.

An early role for your team to assign might be to have someone start working on cleaning the maps you wish to use and creating a list of points of interest (this should not be their only role in the team, however). Another would be researching ways to import and use the maps in Java.

4. Requirements

Your project will need to adhere to a collection of functional and non-functional requirements. In essence, the functional requirements outline what your application will need to do, while the non-functional requirements specify how you're supposed to go about doing things.

4.1 Functional Requirements

Functional requirements include required functionality, as discussed in the sections below. While this functionality is required, you still have a fair number of design choices to make along the way, as well as opportunities to exercise creativity.

4.1.1 Required Functionality

You must implement all of the required functionality for your project to be considered complete. How the below functionality is delivered is for the most part up to you. While we will not be grading visual appeal or aesthetics directly, if things slide to the point where your application is unintuitive, difficult to use, or unreadable, then this could impact your overall grade.

4.1.1.1 Browsing Maps

Your application needs to provide an easy way for the user to browse through all of the maps for all of the buildings supported by your application (again, as noted above, you only need to support Middlesex College and two other buildings at a minimum but are free to do more). The user should be able to simply select a building from a list and then flip through all the maps

(floors) for the selected building. Switching between buildings to browse should also be fairly seamless, intuitive, and easy to do. Floors should be correctly labelled with a floor number that matches the floor plan map.

The functionality provided for this feature should be similar to what is already available through [Western's map site](), but feel free to explore and experiment with alternate layouts and navigation mechanisms (like a drop-down box for buildings instead of an always-on-display list).

4.1.1.2 Scrolling Maps

When viewing a map in the application, if the application's display is not big enough to accommodate a map in its entirety, then you must provide a way of scrolling the map, for example with scroll bars. Scrolling the map should work correctly with any POIs or layers currently displayed on the map.

At least one map in your application must be large enough to require scrolling.

4.1.1.3 Displaying Layers

Each map should also have the ability to display and hide **layers**. **Layers** are defined as a collection of common POIs for the current map that fall under the same category (e.g. classroom, lab, restaurant, washroom, etc.). Hiding a layer should hide all POIs that are grouped in that layer, displaying a layer should show all of the POIs that are grouped in that layer. It must be possible to display multiple layers at the same time. How this is done is up to you. For example, you could use toggle buttons to turn the layers on and off through your main interface, you could embed such toggles in a menu, or you could have a pop-up screen for this.

At a minimum, you should have a layer for accessibility (this is built into the PDFs), a layer for washrooms (also built into the PDFs), and a layer for each type of point of interest (e.g. classrooms, restaurants, labs, user defined points of interest, favorited points of interest, etc.). Please note that accessibility and washroom information is baked into the maps by default and cannot be turned off, unless you edit the maps as noted in Section 3. This is fine, but in such a case you should not let the user be able to toggle that information on and off as they cannot do so; rather, this option should be shown, but the user cannot interact with it. You are free to add additional layers that add more information than POIs if you think they would be useful, for example, you may wish to have a layer that shows/hides the map legend, one that adds additional text to the map, or one that colour codes different rooms in the building.

4.1.1.4 Searching Maps

Your application also needs to provide an easy way for the user to search for particular Points of Interest (POI) in particular buildings (both built-in POIs and user defined ones). This search should be text based where a user input a string to search for that is matched against the POIs metadata (e.g. room number, description, name, etc.). Once the user initiates a search for a POI, the map containing the POI is displayed, the POI is highlighted in some fashion on the map, and the map is scrolled to the correct location (assuming the map has scroll bars).

In addition to highlighting the POI, a short description should be displayed that includes the room number, title/name of the POI, and a description. This can be the same highlighting and display that is used when the user clicks on a POI.

How you design your interface to perform the search and how you highlight the target POI on the map is up to you, but it must be some sort of text entry with proper error handling (e.g. if the POI can not be found). For highlighting on the map, you could drop a pin at the location or layer some transparent colour over the location (the latter is harder because rooms can be all different shapes, sizes, and orientations on the maps). To show the description for the POI, you might have a separate element in your user interface for this, or you can have a text box pop-up when you hover-over a dropped pin on the map, for example.

4.1.1.5 POI Discovery

In addition to being able to search for POIs with a text-based input, your application should allow the user to discover what POIs are available on each map. At a minimum this should be a list of POIs available on the currently displayed map. This list can be subdivided by POI category or organized in another way so long as it is possible for the user to find every POI on the current map (including user defined POIs).

Clicking on a POI in this list should display the POI on the current map, highlight it, scroll the map to it's location (if required), and display a short description of the POI including the POI's name/title, room number, and a short description.

You must carefully consider how this will interact with the Displaying Layers (4.1.1.3) requirement. For example, will selecting a POI from the list hide other layers? Will it force the layer the POI is on to be shown? What happens when a layer is hidden/displayed when a POI is highlighted.

4.1.1.6 Built-In Points of Interest (POI)

Your application must come with various types of points of interest (POI) built-in that can be separately layered over the map when browsing or searching. The user should be able to choose

which POI are being shown by displaying and hiding layers as per 4.1.1.3 Displaying Layers (i.e. there should be a layer for each type of POI).

You need not include every room or location listed on the map. At a minimum, your application needs to have POI for classrooms (only classrooms listed in the [Western Classroom Directory](#)), navigation (important stairwells and elevators), washrooms, building entry/exit points (only the main/common ones), GenLabs ([see the GenLabs site for a list](#)), restaurants, and Computer Science specific POIs for Middlesex College (e.g. computer labs, collaborative rooms, etc.). The department website lists some of these at the following links:

- [Middlesex College Floor Plans](#)
- [Computer Science Labs](#)
- [Computer Science Classrooms](#)
- [Computer Science Collaborative Spaces](#)

You may add other types of POI if you think they would be useful. You will need to define metadata for each POI. How you store these is again up to you, so long as it is organized and usable. This metadata will need to include, for each POI, a name/title label that is searchable, a room number (also searchable), its position on the map (as simple as an x,y coordinate pair, for example), a short description for the location (searchable), and the POI type/category (e.g. classroom, lab, restaurant, etc.).

You can optionally include additional information as well. For example, suppose you were doing a map of the North Campus Building (NCB). The Tim Horton's on the main floor has a room number, and you could allow a user to search for it by room number. You might also want a second metadata entry with "Tim Horton's" as the name, as practically no one would ever look for it by room number. For both metadata entries, in this case, you could include a summary of the menu and hours of operations in the description for the POI.

It is up to you to determine how to display a POI when the corresponding layers are turned on (again, you can drop pins, add highlighting, etc.). Be creative, informative, and do something that stands out from the rest of the map. You must make sure that your points of interest track properly if the user has to scale or scroll your user interface.

4.1.1.7 Clicking on POIs

Clicking on a POI currently displayed on a map should highlight that POI to clearly indicate that it has been selected and display information about this POI. At a minimum the POI name/title, room number, and a short description should be displayed somewhere in your interface or as a popup. The option to favourite the currently selected POI should also be displayed (see Section 4.1.1.8).

Deselecting or unhighlighting the POI should cause the information to be hidden.

4.1.1.8 Favourites

Your application must allow users to mark and unmark POIs as favourites for quick access. For example, once a POI is highlighted, a favourite toggle can be presented to the user at the same time for such marking purposes.

POI marked as favourites must be accessible through some sort of menu or list that is accessible from any map; when selected, the corresponding location is shown on the appropriate map, just as if the user had searched for the location manually. This may require switching the map, scrolling the map, and changing the POI layers currently displayed.

This would allow users to store the locations of all of their classrooms, for example, for easy access. POI marked as favourites must be remembered in future sessions with the application (See 4.1.1.10 Persistent Data).

The user must be able to store at least 10 POIs as favourites, although you can store more if you choose to do so (or even have no limit). You can also provide ways to allow the user to organize their favourites (into folders, with tags or labels, etc.), but again this is up to you.

4.1.1.9 User Created Points of Interest (POI)

In addition to built-in POI (see 4.1.1.6 Built-In Points of Interest (POI)) users of your application should be able to create their own POI. The interface to enable users to create these POI is up to you, however, it must allow users to designate a point on the map and provide a name, room number, and description at a minimum.

This user created POI should be given their own layer and be editable and removable by the user (built in POIs should not be removeable by the user). This includes both changing the text-based metadata as well as the points location on the map. Changing the location of a point should be done visually (graphically) and not by entering numbers. As with favourites (see 4.1.1.8 Favourites), these user created POI should be accessible through some kind of menu or list that allows the user to select the point to show on the map, edit the point, or remove (delete) the point permanently.

The user must be able to store at least 5 user created POI, although you can store more if you choose to do so (or even have no limit). User created POI must be remembered in future sessions with the application (See 4.1.1.10 Persistent Data).

4.1.1.10 Persistent Data

All data, including the built in POIs, user created POIs, and user favorites most be stored persistently. This way, this information is saved between sessions and available for use once the application is restarted. This data can be stored in one or more local files accessible to the application. I imagine it would be easier to organize metadata as a collection of files, like one to list buildings, and one each per building for building metadata, but how you do so is up to you and your group. As long as it is stored persistently and is available between user sessions, that is the main thing.

Some suggested methods of storing this data are JSON, CSV, TSV, or XML formats. However, the exact method is up to you. Using a database is allowed (if it is stored locally) but not recommended. Ideally only one method should be used for simplicity.

4.1.1.11 Exit/Close and Navigation

User must be able to exit your application cleanly from any current window/state of your application. This must not result in any data loss or corruption. If the user is currently performing a task that has not been completed or saved (e.g. entering data) a warning should be displayed to prevent the user from losing any unsaved work.

The user must always be able to easily navigate between screens/windows in your application. There should always be a "back" or "cancel" button to return to the previous screen/window.

4.1.1.12 User Help

Users must be able to access help of some kind if they get stuck (e.g. a help menu with a user guide). This help guide must be detailed enough to cover all features of your application and clearly explain how to perform all possible tasks. How you choose to make this available to the user is up to you, it can be built into the application or a separate PDF or html page your application opens. However, it must not require an internet connection (i.e. the user guide must be stored locally).

Your application must also have a means to accessing an "About" screen that displays the name of your application, the version, the release date, and a list of your team members names. It should also include a way of contacting your team with questions (e.g. a valid e-mail address for one or more of your team members).

4.1.1.13 Editing Tool/Mode

Provide a special mode or separate tool to allow developers to easily (graphically) edit the metadata for the built-in POI (implemented well, this could help you produce the data you need instead of creating it by hand). It is up to you to determine how you want the developer to access this functionality. It could be a special account, accessible through a password protected menu in the application, a separate tool that edits the same metadata file, or perhaps they run the application with a special command line option (like "-edit") to access this functionality. How you do this is up to you, but it should prevent normal users from accidentally accessing this functionality.

The editing mode will let you work with your POI metadata. You need to be able to add, edit, and remove points of interest. They must be added and edited graphically by placing them on a map, although you also must be able to specify the other metadata for the POI, including name, room number, description, and type/category so that it can be toggled on and off with the right layer.

How you design the interface for your editing mode is again up to you, as long as you provide the required functionality, and it is accessible and easy to use for the developer.

4.1.2 Extra Functionality

The functional requirements in this section are "nice to haves" but not part of the core functionality of your application. **For a full grade on the project your team must include at least one extra feature from this section.** More than one extra feature can be added if desired and a small amount of bonus marks[1] will be awarded for including and fully implementing more than one extra feature from this section.

4.1.2.1 Multi-user System

If you pick this extra feature, your application now needs to support multiple users. This should be done through an account system of some kind which requires the user to sign into the application using a username and password. These accounts should be local to the system the application is installed on (you are not required to connect to any kind of remote server or database). The account information can be stored in a file, database, or any other way so long as the application still functions without an internet connection.

---

[1] Bonus mark can not make overall course grade go over 100%. Bonus mark can not make project grade go over 115%. Any bonus mark awarded is at the sole discretion of the course instructor and requires the extra feature to be implemented fully, without error, and not at the cost of a core feature. Not all extra features are equal, extra features that require more work will be granted more bonus marks.

Users should not be able to view or edit other users' favorites or user defined POIs within the application. However, you are not required to protect against users accessing the raw files.

4.1.2.2 Secure the User Data

The requirements in this document do not have any specifications for security or privacy of user data such as user created POIs or favorites. For this extra feature, you must secure the files used by your program that contain user data such that they are protected even if someone is directly viewing the raw files your application uses.

To accomplish this, you may need to use encryption. The exact method is up to your group to decide but should be secure and use best practices. You can (and should) use pre-existing encryption and security libraries and should not attempt to create your own encryption algorithm.

This may require that the user enters a key or password to decrypt their data or you may find other alternatives (e.g. a key file on a usb stick, etc.).

4.1.2.3 Current Weather

Your application must now also display the current weather on campus (or at least in London) by connecting to a weather service such as the API provided by WeatherAPI or OpenWeather or another provider. The provider is up to you but note that some of these services are pay to use but have free plans for a limited number of requests per day.

The weather data for the current day should be displayed in a visually appealing and easy to understand way in your application. Having a icon that changes based on current weather conditions would be ideal as well as the current temperature would be ideal, but you are free to add more detail.

4.1.2.4 Campus Map

The requirements in section 4.1 only specifies that maps of select building's floors are shown. In addition to this also present an overall map of Western's main campus with all of the buildings shown (you can find PDF of maps like this at https://www.uwo.ca/about/visit/maps.html). Users should be able to clearly see what buildings have floor maps available in your application and clicking on one of these buildings should bring up the floor plan for that building.

Additionally, the overall campus map should have its own POI layer with a POI for at least 10 buildings (you can have more if you like). These building POIs should be searchable.

4.1.2.5 Extra Classroom Metadata

The Western Classroom Directory contains detailed information about each classroom including seating capacity, seating type, projection screen type, information about A/V equipment in each room, pictures of the class room, and in some cases exam seating charts. Add **all** of this extra data (including pictures and exam seating charts) to your classroom POI for the buildings you have selected.

Make sure that the information is presented in a reasonable and easily understandable way to the user. Not all of the information must be shown at once (for example, you could have the user click a picture icon to see a picture of the classroom).

4.1.2.6 Class Find

You application should provide a service similar to ClassFind.com (https://www.classfind.com/Western). That is, users should be able to select a classroom in one of the buildings you provide maps for and a series of written steps and pictures should be shown that guide the user to that classroom. These steps do not have to be from the user's current position but from any entrance to the building of your choice (just like ClassFind.com does). The steps can be hardcoded or predefined for each classroom, you do not need to display a dynamic routes.

4.1.2.7 Edit Buildings and Floors

For this extra feature, your editing tool (from Section 4.1.1.13) must now also provide tools for adding, editing, and removing new buildings as well as floors for each building. You should carefully consider what happens to the POIs in a building or on a floor if it is deleted.

Developers should be able to select images to use for the floor map from the file system. It is ok if you restrict this to a list of available maps from a set folder so long as this list is updated dynamically if a new map file is added to the folder.

4.1.2.8 Your Team's Idea Here

You may suggest an extra feature of your own to add that will count for the extra feature, but it must be a significant feature (not trivial to implement) and first approved by the course instructor in writing (via e-mail).

4.2 Non-Functional Requirements

Your application will need to adhere to the following requirements, and these requirements will be taken into consideration in the assessment of your project.

1. The application must be developed in Java 19 (the current version of Java) and be a desktop application.

2. The application will need to use a Java graphical user interface of some kind using either Swing (recommended) or JavaFX; the choice of framework is up to you, though you will need to standardize across your group. Swing and JavaFX will not be taught in this course so it will be up to your group to research and learn how to use these toolkits. This is meant to simulate real-life where you are expected to learn some new frameworks, APIs, and tools on the job.

3. The application must store its map metadata locally and not require an internet connection. A good option is JSON. To use JSON, you can either use standard Java framework methods and your own code for working with this, or you can use a third party library for working with the JSON data. While a third party library might make some things easier or more efficient, they can also be complex and difficult to use, costing you more time in the end than you saved. The choice is up to you and you can also pick options such as XML, CSV, TSV, etc. but choose carefully. A database can only be used if you are storing the data locally and no internet connection is required (using a database is not recommended for this project but allowed).

4. If you are keeping the maps for your application in their original PDF format (not recommended), you may also need to use a third party PDF library. Again, keep in mind that some of these options might pose challenges in using and integrating them with your code, and storing the maps as images is both allowed and recommended.

5. The application should not need to use any other libraries. If you do wish to use another library, it must be easy to obtain and install (or ideally included with your application). The TA marking your project will need to be able to run it to test it, so you will be responsible for providing instructions on compiling and running your application including any required libraries.

6. Your program must be developed using the NetBeans IDE and packaged as a Maven Project. Your included pom.xml file should correctly include any required dependencies or plugins.

7. All code and files for your project must be stored in the Bitbucket Git repository created for your team (details on this will be announced once available). Your team must actively use this repository and not simply commit the files at the end of the project.

8. All design work and diagrams for your project must be stored and developed on Confluence (details on this will be announced once available).

9. All tasks and issues related to your project must be tracked on Jira (details on this will be announced once available) and be updated as the project progresses.

10. All code in the application must be commented using Javadoc.

11. The majority of your code should be unit tested with a sufficient number of JUnit 5 tests.

12. You may choose as a team the coding conventions and styles you wish to adopt in your code (for naming things, indentation, etc.). However, you must remain consistent in applying those conventions and styles across all files in the application.

13. The application must be executable on a Windows 10 system with a standard Java installation (Java 19), and each team member must be able to compile it and run it from a development environment they have access to.

14. The application must be well self-contained and not create, modify, or delete files outside of the directory in which the application is installed, and subdirectories of this directory.

15. The application must present a visible response to every user action. Erroneous actions or actions that could not succeed for some reason must be met with a useful, professional error message.

16. The file size of the project as a whole should be under 1 gigabyte.

17. The application should run efficiently and not use unnecessary computing resources.

18. The application's GUI must follow best practices for user interface and user experience design.

19. Your application must take accessibility into account when designing the user interface. Allowing tasks to be completed with a keyboard or mouse would be ideal, as well as ensuring UI elements have a logical tab order. Colour use in the UI should also be carefully considered.

20. The application must be designed with sound software engineering principles in mind.