Design Documentation

WesternU GIS

This is a design documentation for our map software for Western University.

Version	Date	Author(s)	Summary of Changes
0.1	Feb 20	Parm	Made document
0.2	Feb 26	Mattias	Development Environment
0.3	Mar 6	Parm	Class Diagrams
0.4	Mar 6	Paul	User Interface Design
0.5	Mar 6	Paul	Reviewed and edited each section
0.6	Mar 7	Parm	Added summary, definitions table, table of contents heading
1.0	Mar 7	Parm	Edited the documentation and added more details

Table of Contents

Summary

This design documentation will be used to begin the implementation of our software. With the class diagrams, we can start programming the classes in visual studio code and uploading them individually to the Bitbucket repositories. Through the user interface mockup, we can use JavaFX to begin the creation of the final user interface. The CSV file format will be used to store data that our app will be accessing and modifying. We will, along the way, continue to reference the various UX/UI/OOP patterns available online to avoid reinventing the wheel.

Definitions

Term	Definition
UI	In computer science, UI stands for User Interface. It refers to the visual and interactive components of a computer program or system that allow a user to interact with and control the software. A user interface can take many forms, such as a graphical user interface (GUI), a command-line interface (CLI), or a voice-based interface. The primary goal of a UI is to provide a user-friendly way for users to interact with the software, allowing them to complete tasks and achieve their goals efficiently and effectively. A well-designed user interface should be intuitive, easy to navigate and provide feedback to the user about the actions they are performing. Good UI design can significantly enhance the user experience and increase satisfaction and productivity.
UX	In computer science, UX stands for User Experience. UX is a user's overall experience when interacting with a software application, system, or product. It encompasses the entire user journey, from the initial discovery of the product to the final use and outcome. A good user experience design considers the user's needs, goals, and preferences. It involves designing the software or product to make it easy, enjoyable, visually appealing, and efficient. It also consists in anticipating and addressing potential user pain points and providing appropriate feedback and guidance throughout the user journey. UX design aims to create a positive and memorable user experience that meets the user's needs and expectations. A positive user experience can increase user satisfaction, loyalty, and engagement with the software or product.

Bitbucket	Bitbucket is a web-based hosting service for version control repositories, primarily for source code and development projects. It is similar to other popular version control services like GitHub and GitLab. Bitbucket is owned by Atlassian and was initially designed for Git, but it also supports Mercurial version control systems. Bitbucket allows users to store and manage their code repositories, collaborate with other developers on projects, and track changes made to the codebase over time. In addition, it offers features like pull requests, issue tracking, code reviews, and continuous integration and delivery (CI/CD) integrations with various tools like Jenkins, Travis CI, and Bamboo.
JavaFX	JavaFX is a Java library that provides a set of graphics and media packages for building rich, cross-platform graphical user interfaces (GUIs) and multimedia applications. Oracle introduced it as a successor to the aging Swing GUI toolkit. JavaFX offers a comprehensive set of UI components, including buttons, labels, text fields, tables, charts, and more advanced features like 3D graphics and animation. In addition, it supports desktop and mobile platforms, providing a consistent look and feels across different operating systems.

Design Introduction

Introduction

Welcome to the Geographical Information System design documentation for the University of Western Ontario Campus project! This project aims to create a comprehensive GIS application for the university campus, including various maps and visualizations that campus administrators can use for students, faculty, and visitors. The goal of this project is to provide a better understanding of the campus and its surroundings, identify areas that may need improvement, and guide future development and planning efforts.

Overview

The University Campus GIS design is a software solution that addresses the need for a more detailed understanding of the campus and its interior spaces. This project involves collecting, organizing, and analyzing various data sets related to the campus, such as locations of multiple resources, facilities, and other points of interest. The GIS design will help us create maps that provide valuable insights into the campus and its surroundings, enabling its users to navigate within the building.

Objectives

The general objectives of this design document and project are as follows:

- To create a comprehensive GIS design of the university campus and its surroundings.
- To create an interactive mockup of the user interference for our software.
- To provide a valuable tool for understanding the campus and its indoor spaces.
- To increase knowledge of the layout of Western's underground tunnel system to better navigate between buildings.
- To deliver a high-quality and user-friendly software solution.

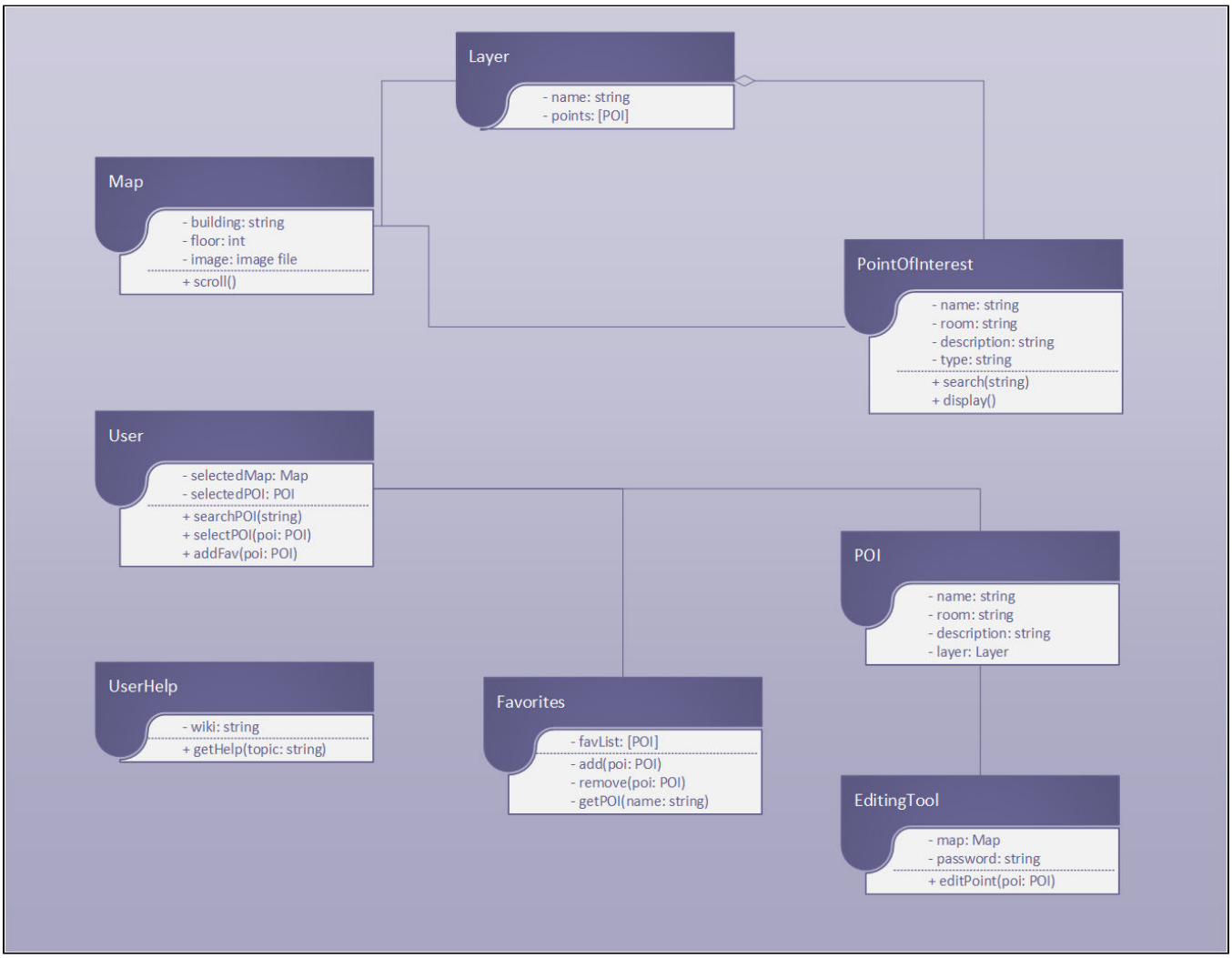
References

Servos, D. (2023). UWO. CS2212 Group Project Specification. https://owl.uwo.ca/access/content/group/f2f3488e-ca75-4b34-978c-0737c707927e/Team%20Project/Project%20Specification/CS2212%20Group%20Project%20Specification%20Winter%202023.pdf Accessed Feb 28, 2023.

Accessibility Floor Plans. (2014). Faculties Management, UWO. https://wufloorplans.uwo.ca/ Accessed Feb 28, 2023.

Site Plan of Pedestrian Enclosed Walkways. (2009). Faculties Engineering, UWO. https://floorplans.uwo.ca/maps/PedestrianTunnels.pdf Accessed Feb 28, 2023.

Class Diagrams



Multiplicity

- `Map` has an association with `PointOfInterest` with a multiplicity of `*` on the `PointOfInterest` end and `1` on the `Map` end, which means that a `Map` can have multiple `PointOfInterest` objects, but a `PointOfInterest` can only belong to one `Map`.
- `Map` has an association with `Layer` with a multiplicity of `*` on the `Layer` end and `1` on the `Map` end, which means that a `Map` can have multiple `Layer` objects, but a `Layer` can only belong to one `Map`.
- `User` has an association with `POI` with a multiplicity of `*` on both ends, which means that a `User` can have multiple `POI` objects, and a `POI` can belong to multiple `User` objects.
- `User` has an association with `Favorites` with a multiplicity of `1` on the `Favorites` end and `1..*` on the `User` end, which means that a `User` can have one `Favorites` object, but a `Favorites` object can belong to multiple `User` objects.
- `Layer` has an aggregation relationship with `PointOfInterest` with a multiplicity of `*` on the `PointOfInterest` end and `1` on the `Layer` end, which means that a `Layer` can contain multiple `PointOfInterest` objects, but a `PointOfInterest` object can only belong to one `Layer`.
- `EditingTool` has an association with `POI` with a multiplicity of `*` on the `POI` end and `1` on the `EditingTool` end, which means that an `EditingTool` can access multiple `POI` objects, but a `POI` object can only be accessed by one `EditingTool`.

CRC

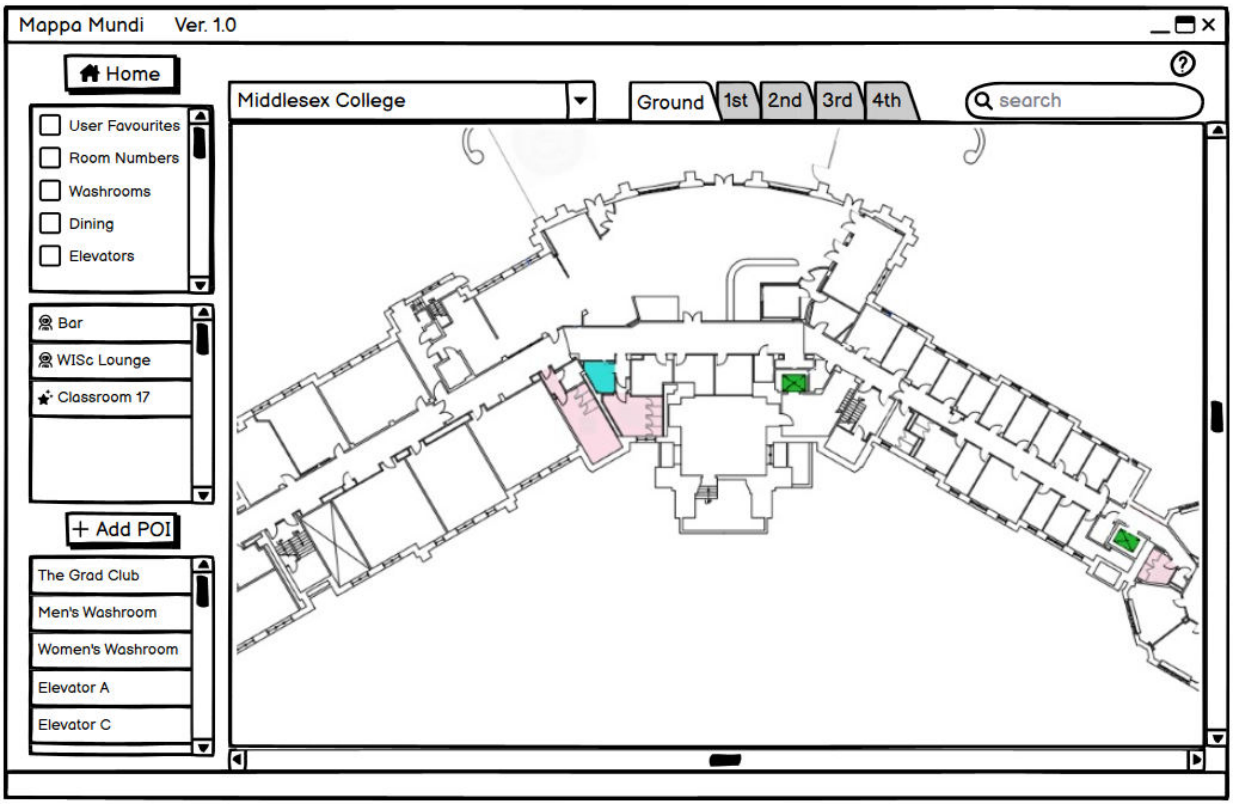
Class Name	Description

Map	Responsibilities: - Display a building or location map - Allow scrolling or panning of the map - Enable users to browse through maps and floors - Enable users to display and hide map layers - Enable users to search for points of interest - Highlight and display information about selected points of interest - Provide a legend for points of interest on the map Collaborators: - PointOfInterest - Layer - User
PointOfInterest	Responsibilities: - Represent a specific location on a map - Store information about the location, including name, room number, and description - Be searchable and displayable on a map - Allow editing and removal of user-created points of interest Collaborators: - Map - User - Layer
Layer	Responsibilities: - Group points of interest together based on category - Allow users to display and hide categories of points of interest on the map Collaborators: - Map - PointOfInterest
User	Responsibilities: - Search for points of interest on maps - Select and highlight points of interest on maps - Add and remove points of interest from their favorites list - Create their own points of interest on maps Collaborators: - Map - PointOfInterest - Favorites
POI	Responsibilities: - Represent a user-created point of interest - Store information about the location, including name, room number, and description - Allow editing and removal of user-created points of interest Collaborators: - User - Map
Favorites	Responsibilities: - Store a list of user-selected points of interest - Allow users to add and remove points of interest from their list Collaborators: - User - POI
EditingTools	Responsibilities: - Provide a tool for developers to edit metadata for built-in points of interest - Allow adding, editing, and removing points of interest - Allow editing of point of interest metadata, including name, room number, description, and category Collaborators: - POI
UserHelp	Responsibilities: - Provide a help section for users to consult - Cover all features and possible tasks of the application - Include an About screen that displays application information and team member names and contact information Collaborators: - None (this class does not interact with other classes directly)

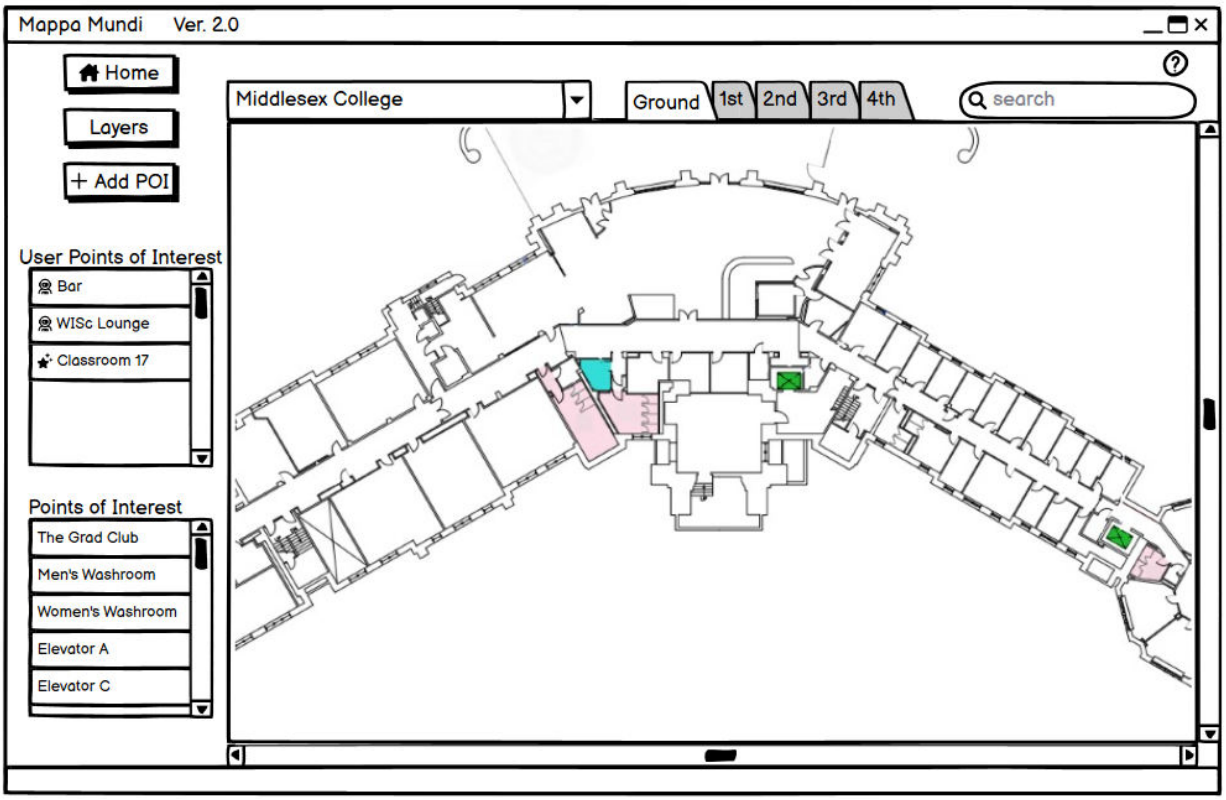
User Interface Mockup

Design Prototypes

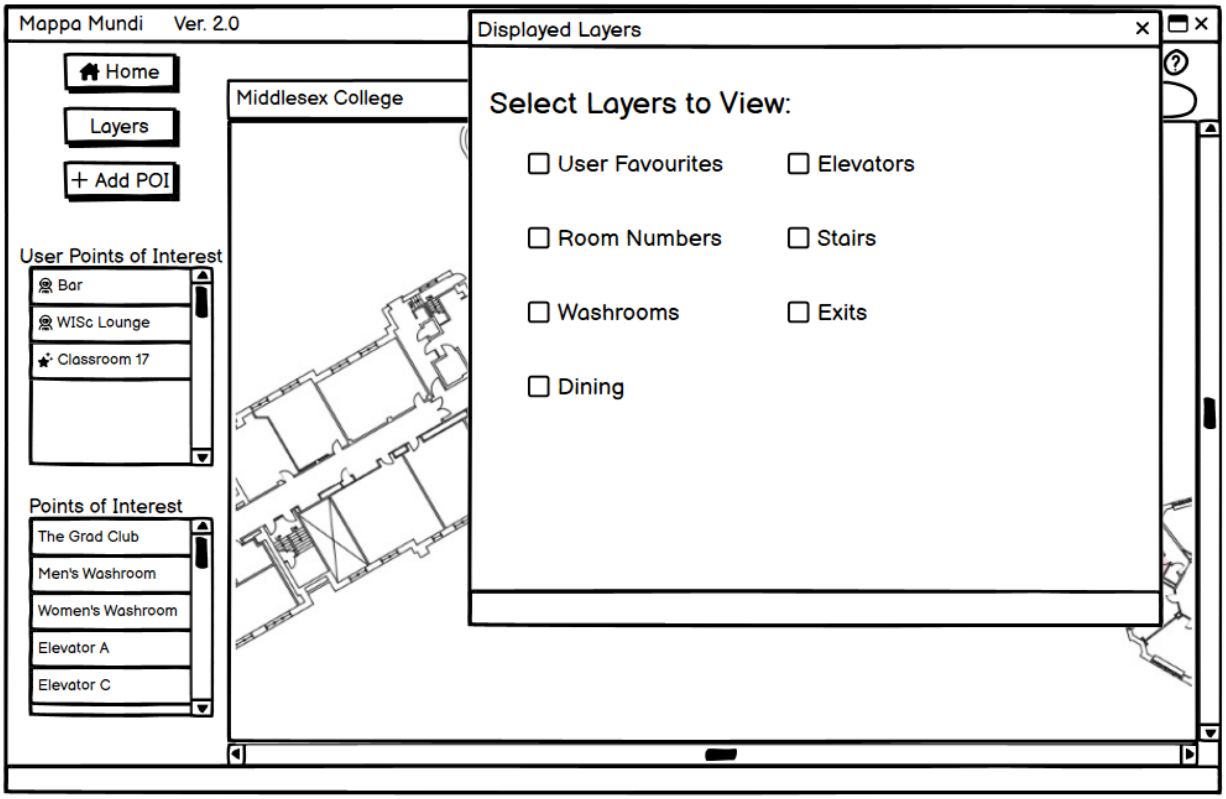
We are considering three options for our GIS design of the University Campus project, numbered versions 1-3.



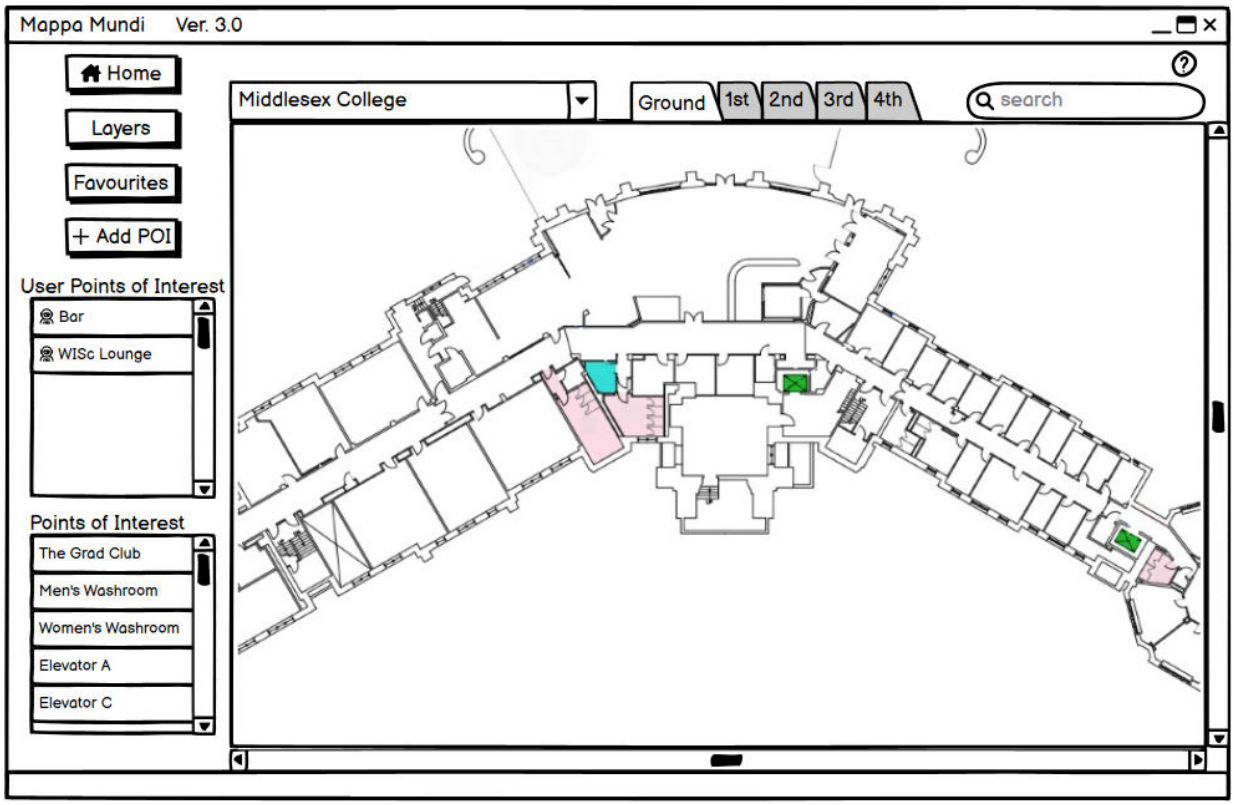
Version 1 (pictured above) is the first edition created. We will showcase most of our functionality through version 1, as the newer editions don't differ much. The map graphic dominates our design as it is the most important visual feature. To create a more intuitive design, we provided navigation features grouped across the top and informational features grouped across the left side of the window.



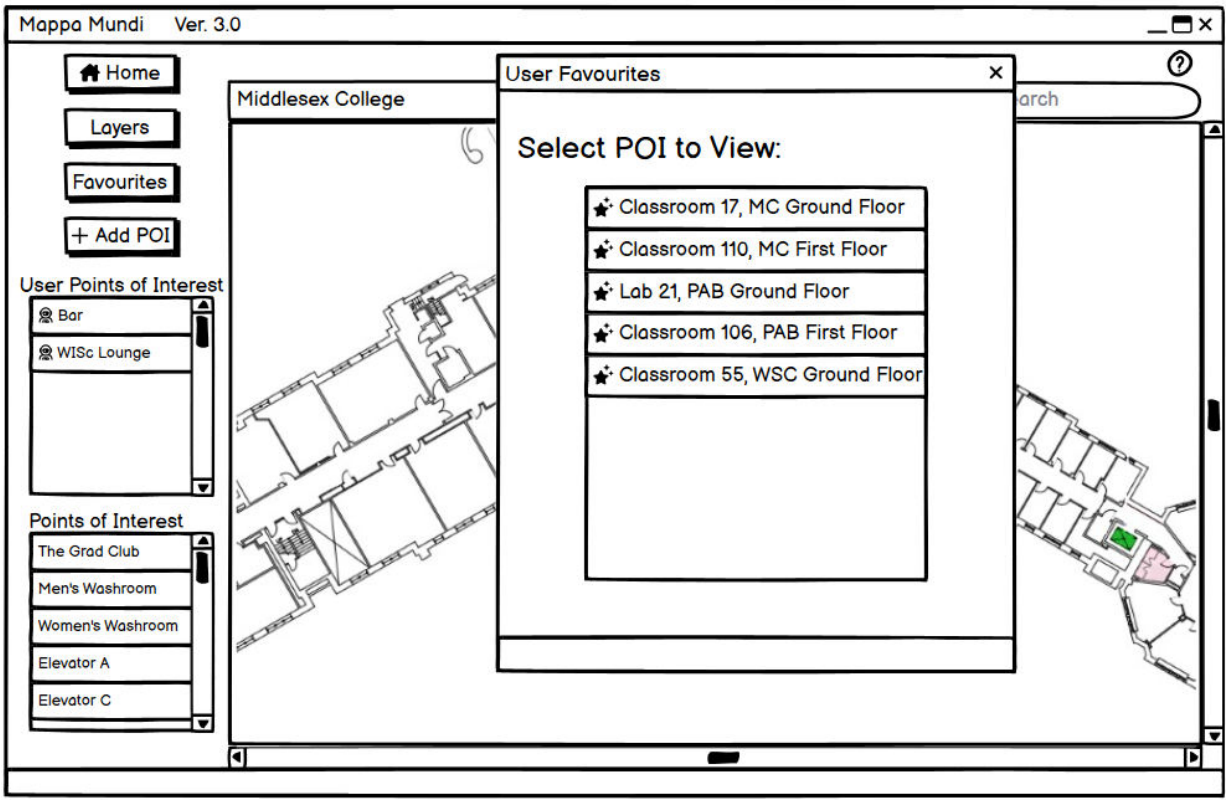
Version 2 (pictured above) was created to reduce the clutter of Version 1. For example, rather than having available map layers spread out in a scrollable box, they are now accessed through the *Layers* button.



Upon toggling the ***Layers* button**, a window with the selectable layers of the map is displayed (pictured above). The layers are displayed or hidden on the map as the User toggles the layer boxes. The User can then close the layers window.



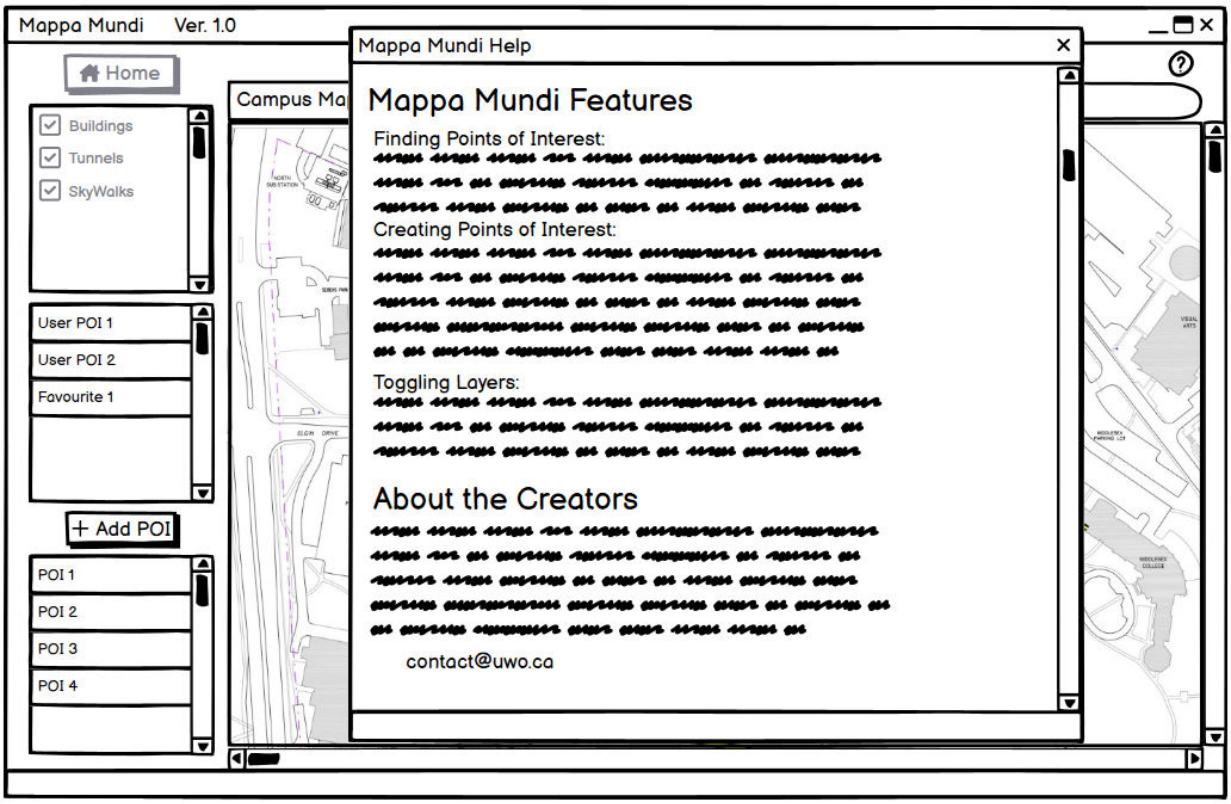
Version 3 (pictured above) was created to provide more accessible, consistent access to the User's favourite Points of Interest. The User can now access all their stored favourited POIs across all the maps through the *Favourites* button. This change also further reduces the clutter seen in Version 1.



Upon toggling the *Favourites* button, a window with all of the User's favourited POIs is opened (pictured above). The User can select the desired point of interest from here, and the map window will change to the chosen map and scroll to display the Point of Interest and its information. The Favourites window is automatically closed upon selection.

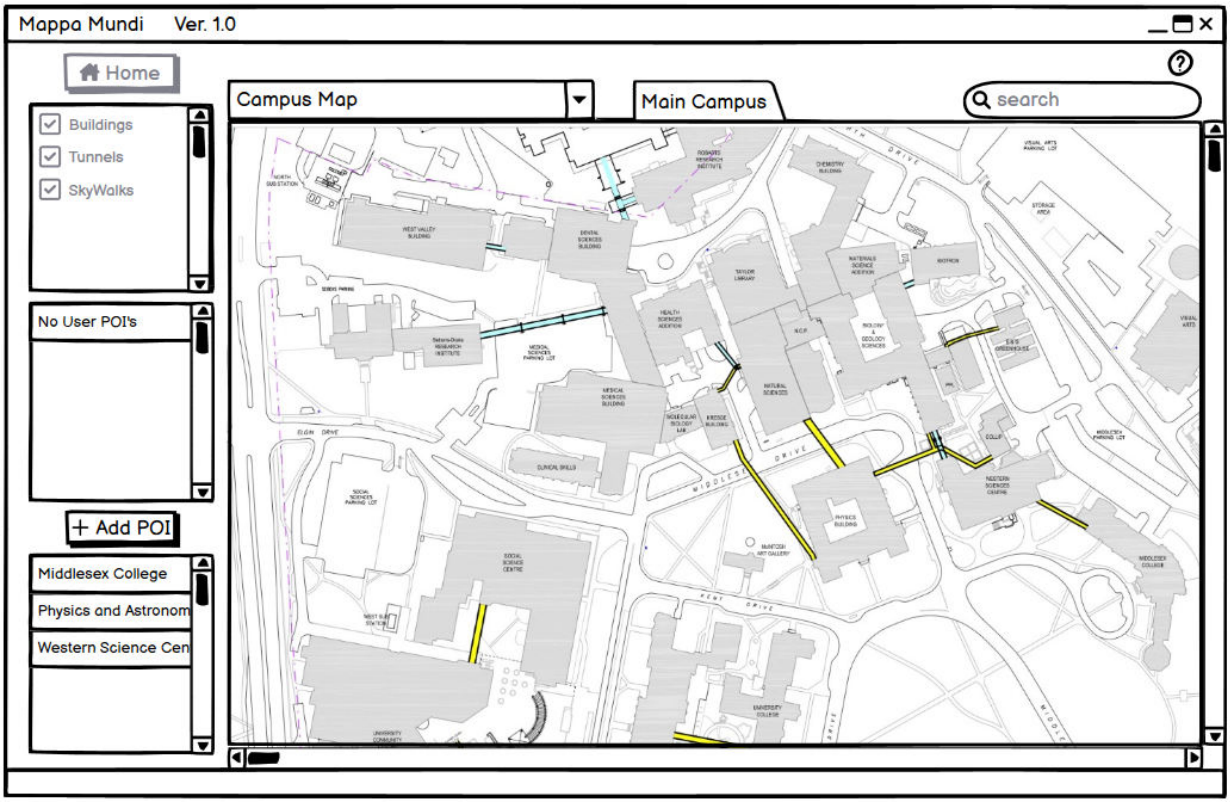
Information and Homepage

All three versions follow the same base design principle of navigation across the top of the window and map information across the left side of the window. An exception to this is the *Question Mark* icon in the top right corner. This is the toggle for the help screen (pictured below), with information about how to use the application and the creators. We decided to put this in the top right corner because we believe this is the most intuitive place to look through the conventions of other applications.

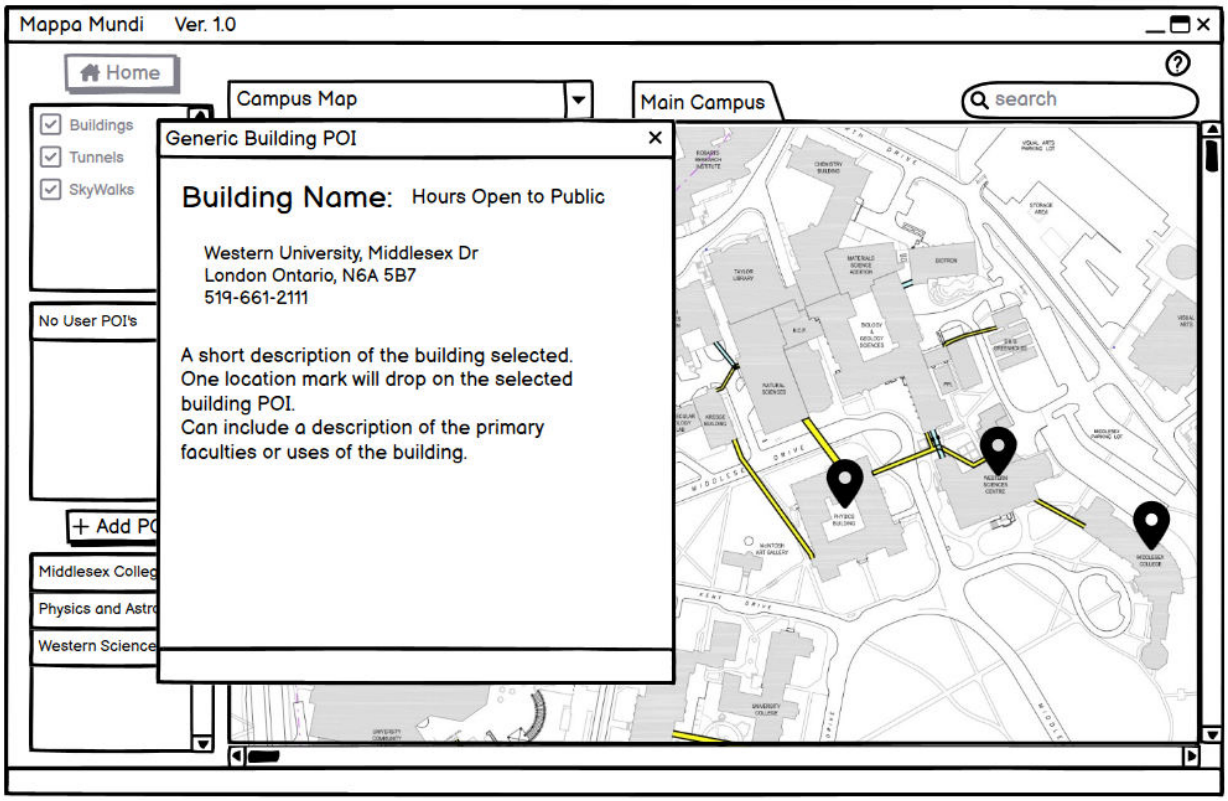


The User can close the Help window (pictured above) when they are finished and open it again whenever needed.

Opposite the *Question Mark* icon on the left side of the screen is our *Home* button. This returns the User to our home page (pictured below) from whichever map the User is on.

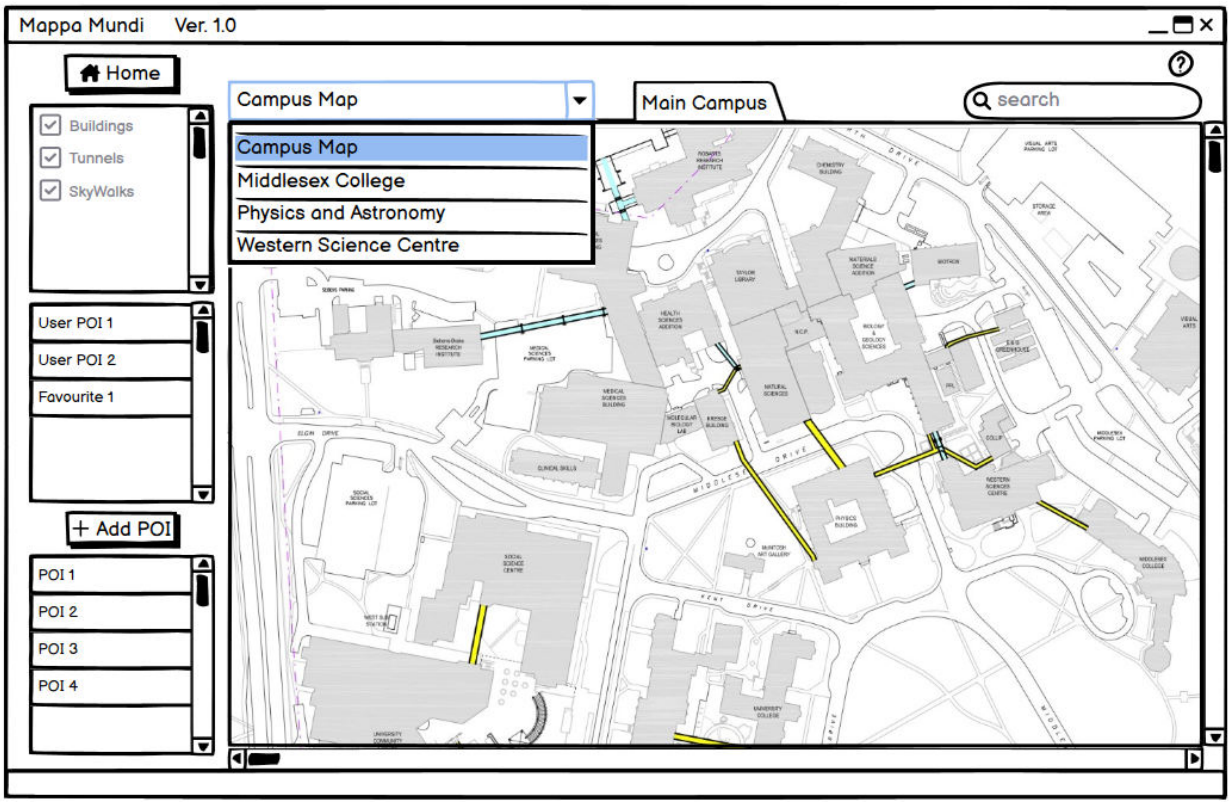


Our Home page (pictured above) featured a map of the Main Campus, including underground tunnels and skywalks connecting campus buildings. This provides the User with the ability to find various facilities on Campus.

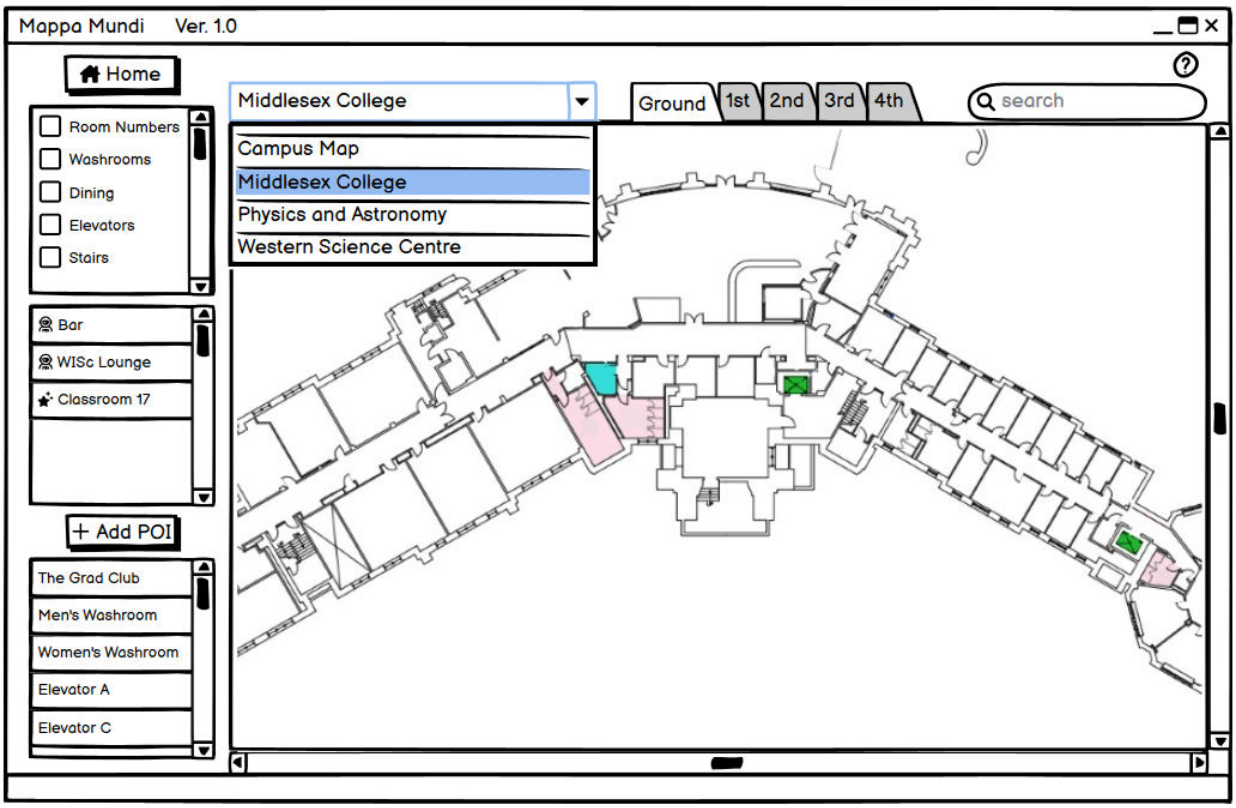


In place of built-in Points of Interest are Building Points of Interest for the buildings available for navigation in our application. Upon clicking the building name in the point of interest list, a Point of Interest window containing information on the building is displayed. In addition, the building is highlighted on the map graphic (pictured above). The User can close this window when finished.

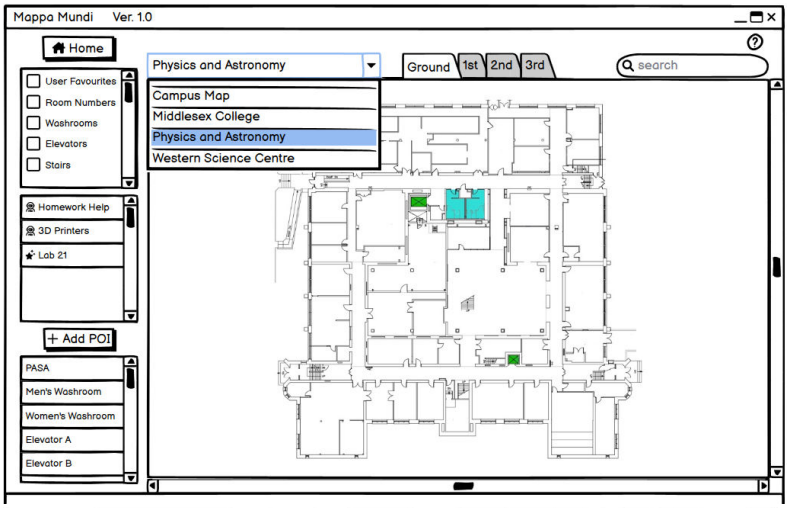
Navigating Maps

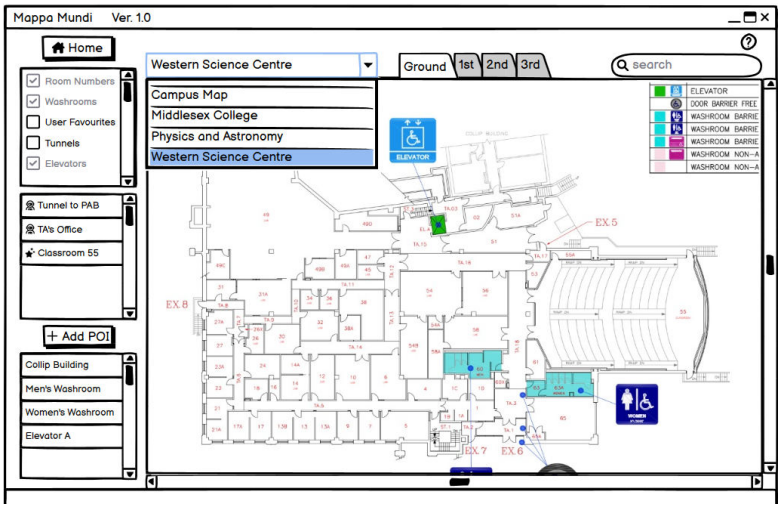


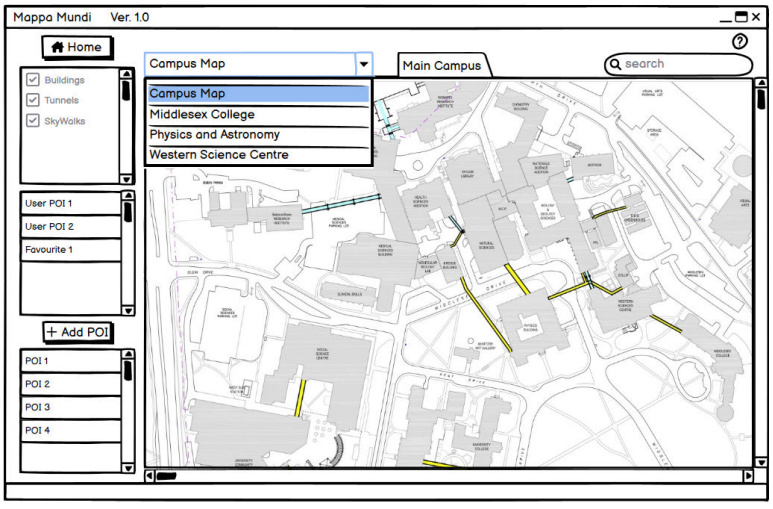
To navigate to maps of a building, the User selects the drop-down menu (pictured above) and selects the desired building.



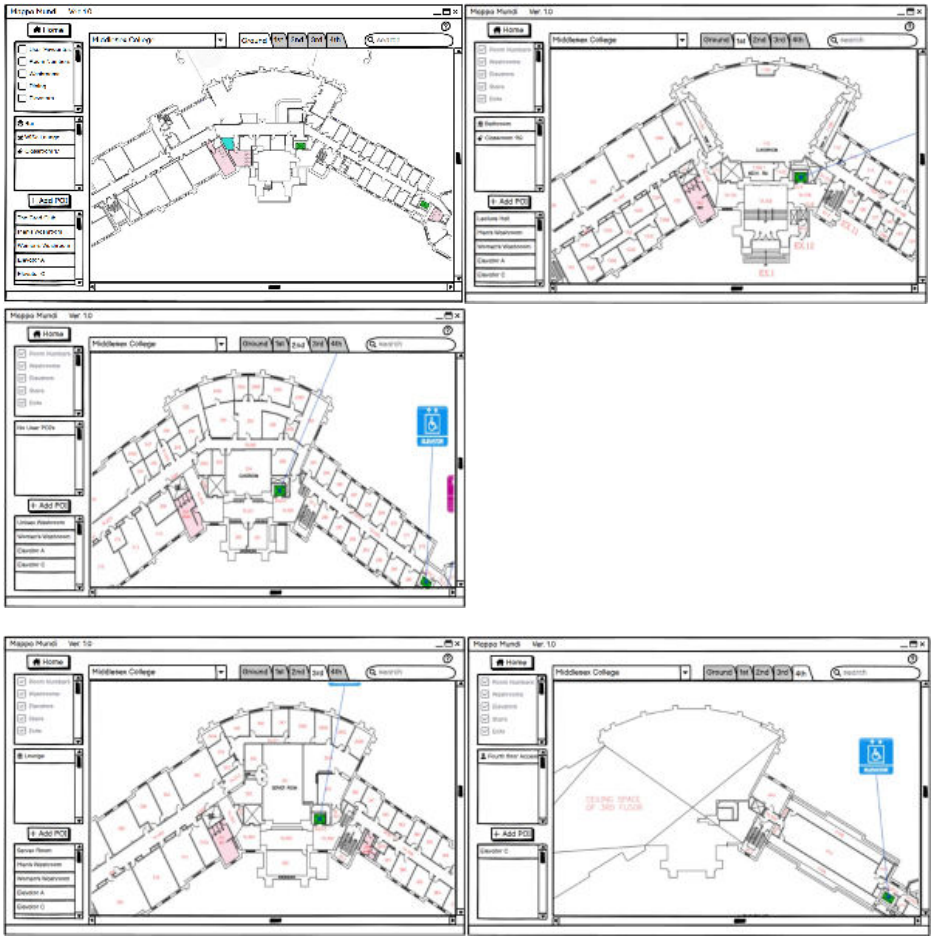
Upon selection, the map graphic is changed to display the desired map (pictured above). Then, through the drop-down menu, the User can switch between maps of various buildings (shown below) or back to the Home page featuring the Campus Map.



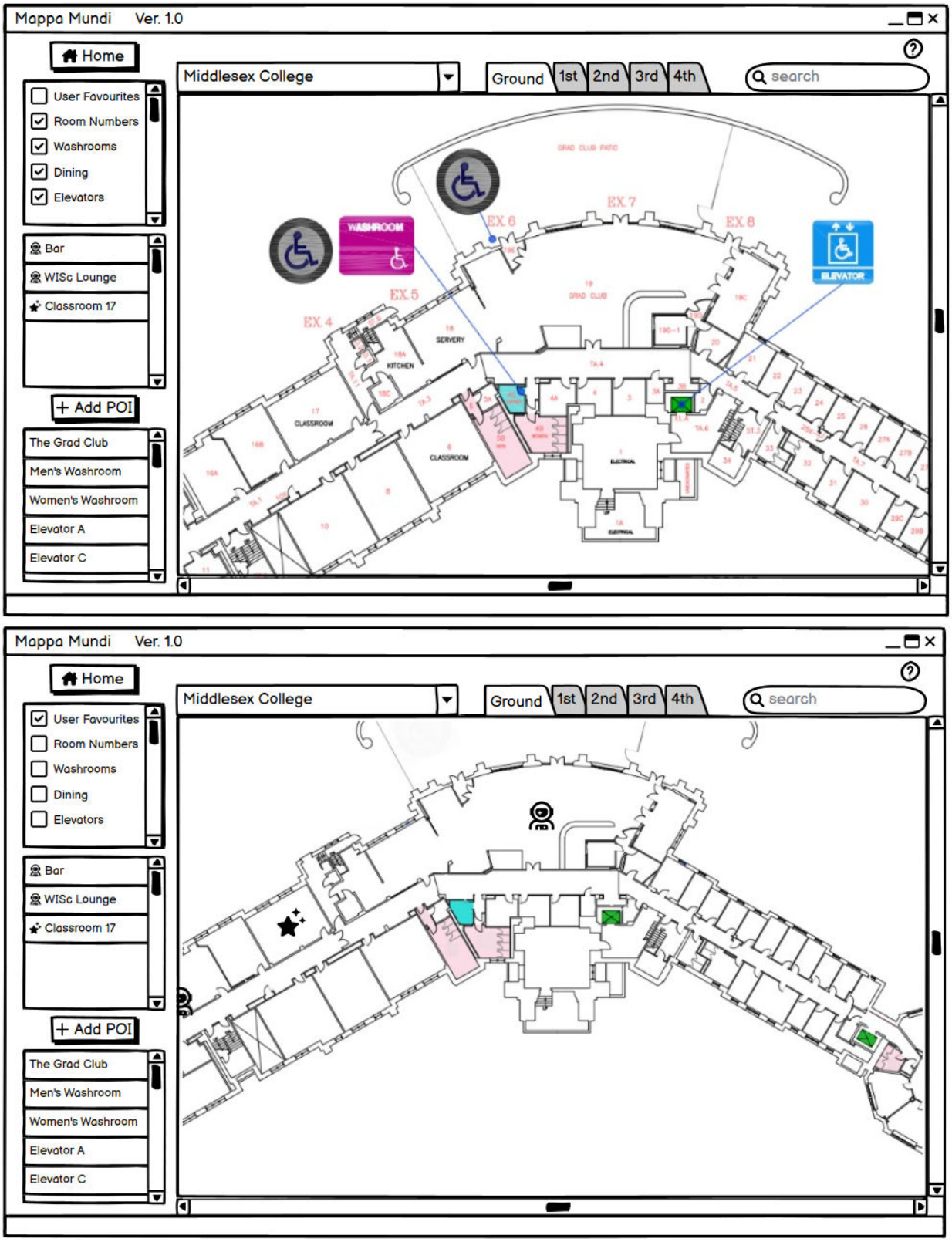




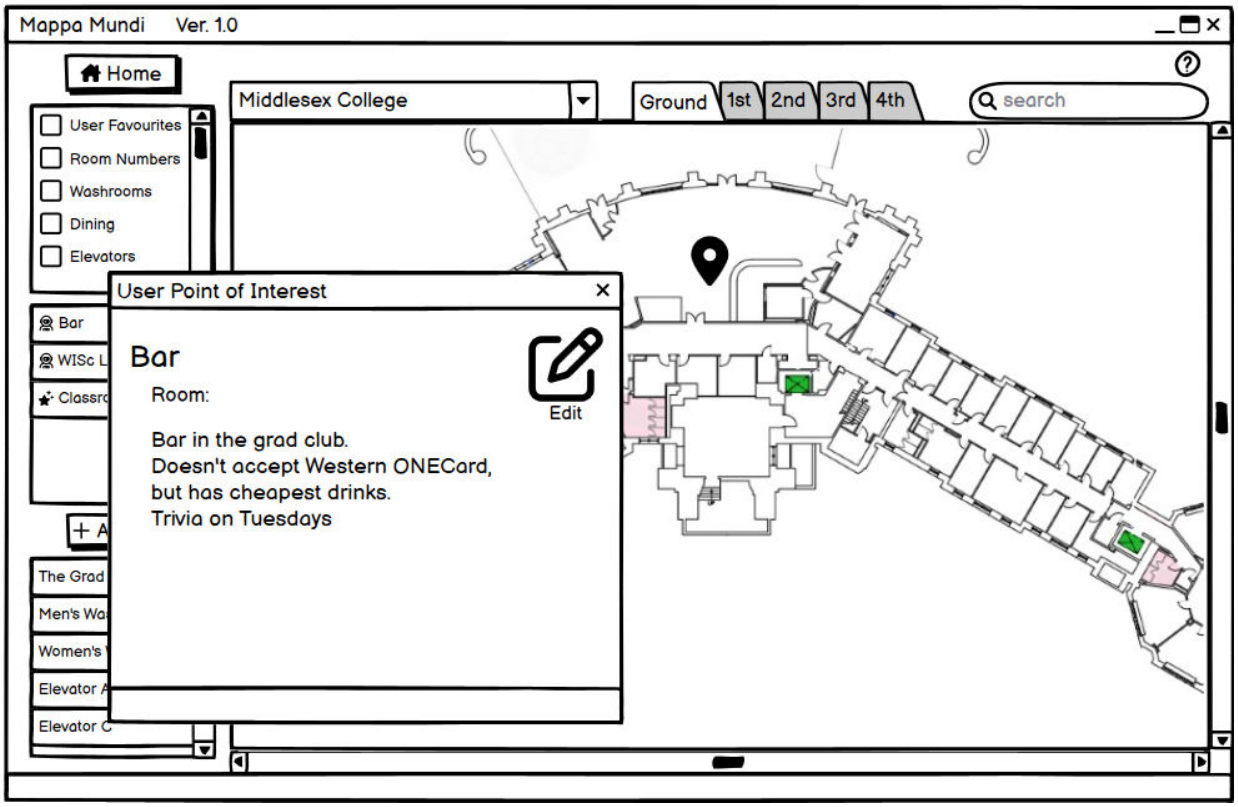
The User can switch between various floors of the currently displayed map through the labelled tabs across the top of the map graphic (pictured below).



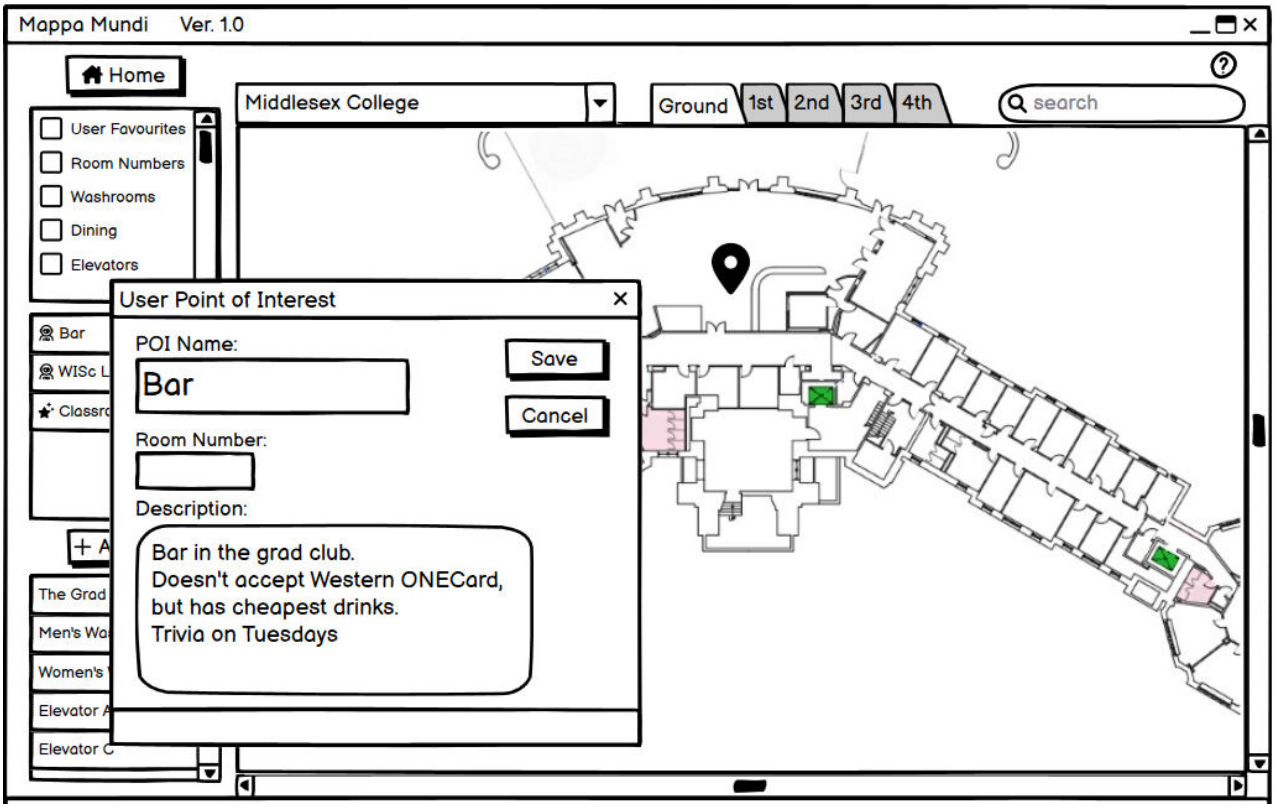
Displaying Layers and Points of Interest

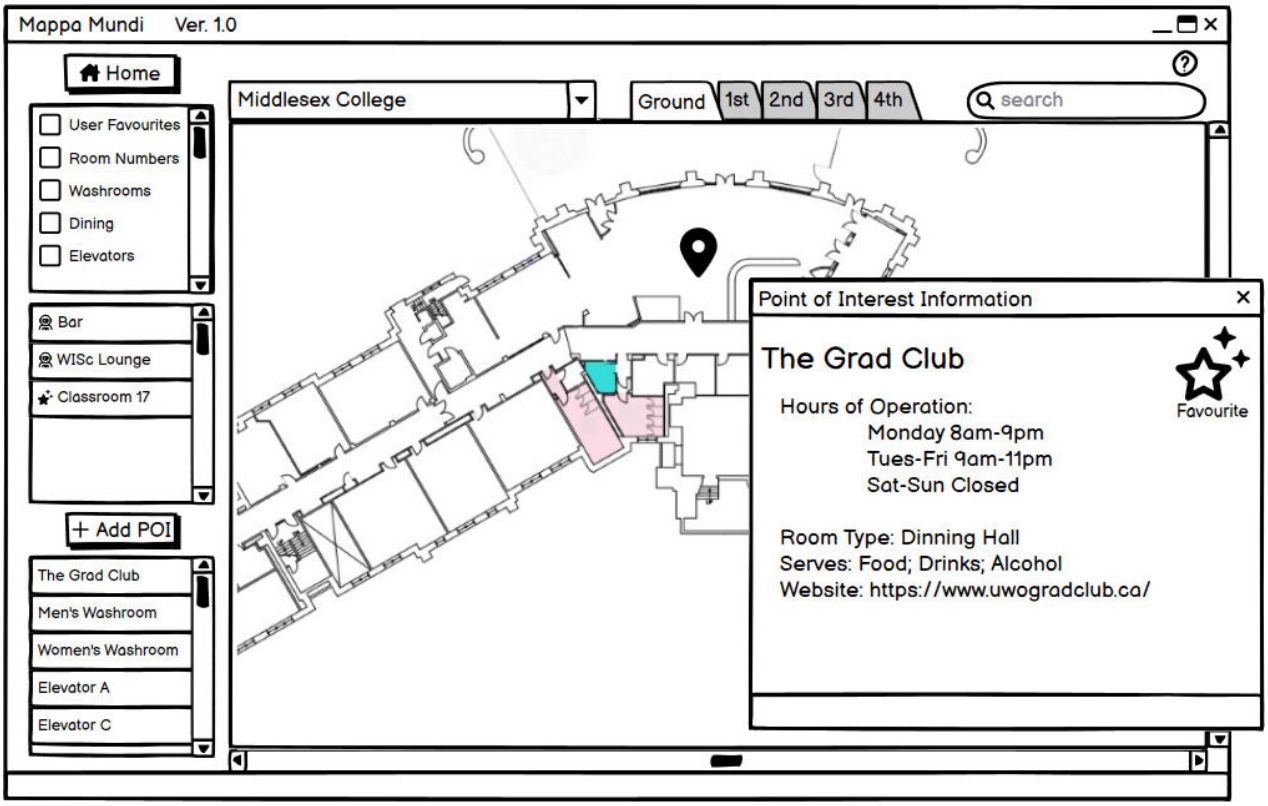


The User can view multiple layers at once (pictured above), including a layer containing any Points of Interest the User has created (shown above). Unfortunately, version 1 stores favourited POIs with User made POIs. Version 3 addresses this design flaw by separating them and showcasing a labelled box of built-in POIs and a box of user-created POIs.

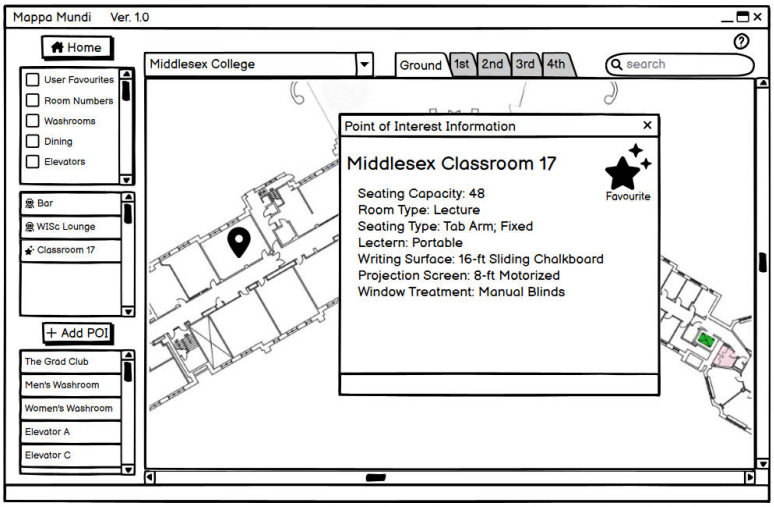


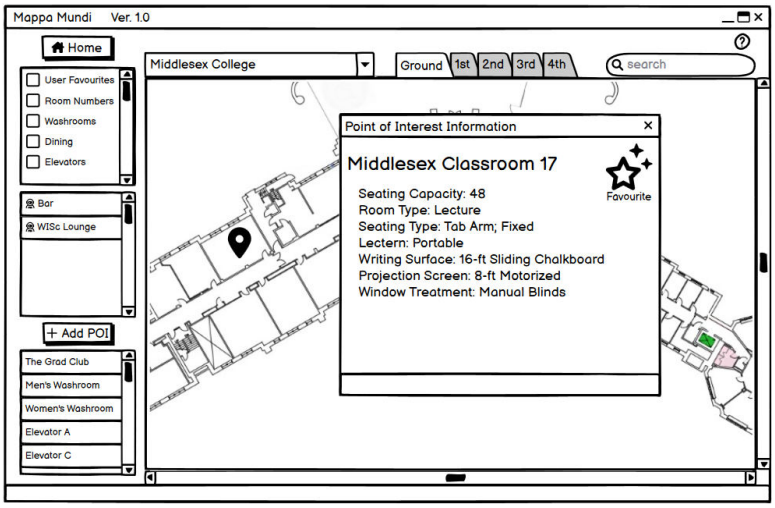
Clicking on the point of interest on the displayed user layer will highlight the POI on the map graphic and open a User Point of Interest window with the information the User provided (pictured above). This window can be closed by the User when finished. The User Point of Interest window also features an *Edit* icon, which allows the User to change the user-created point of interest (pictured below).

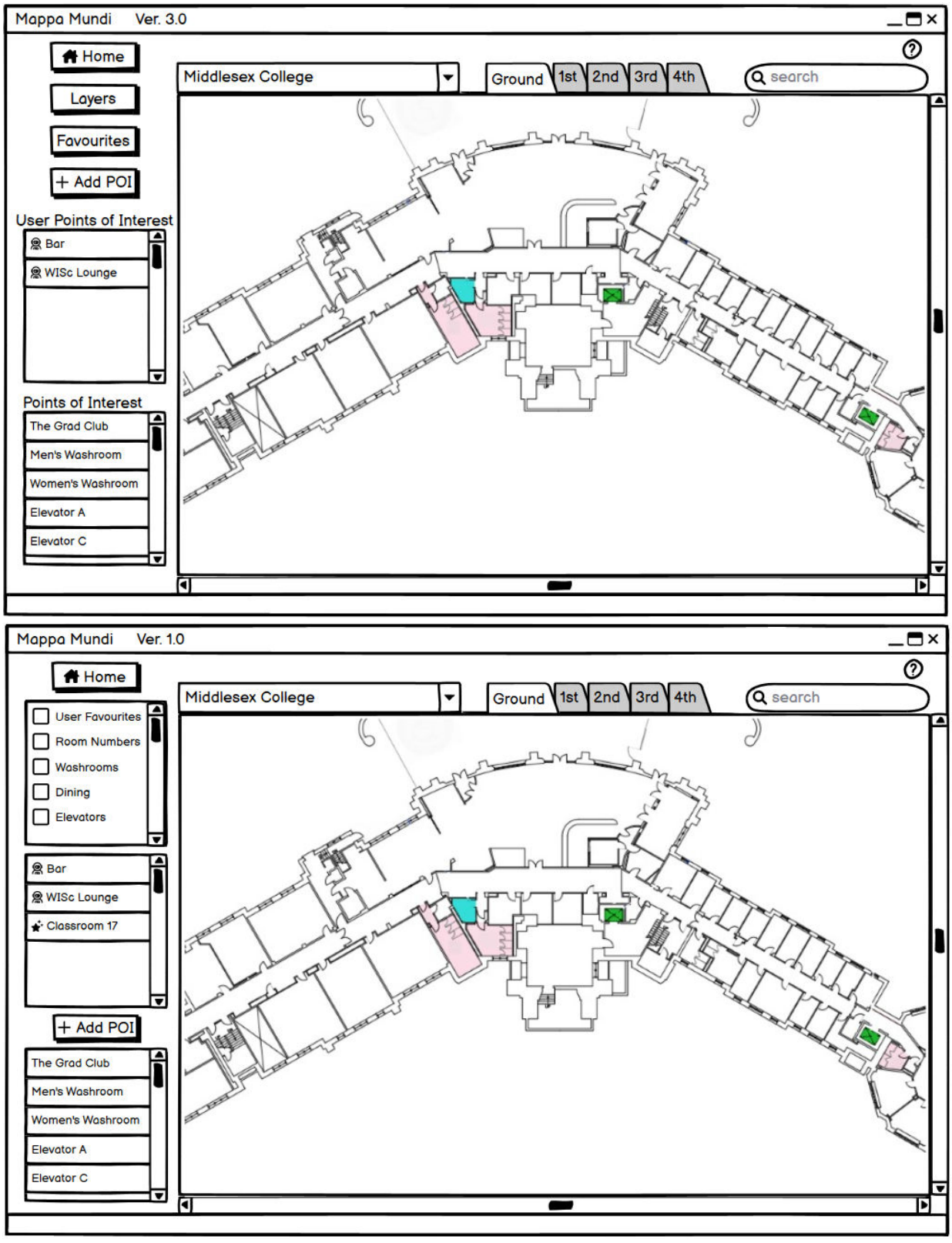




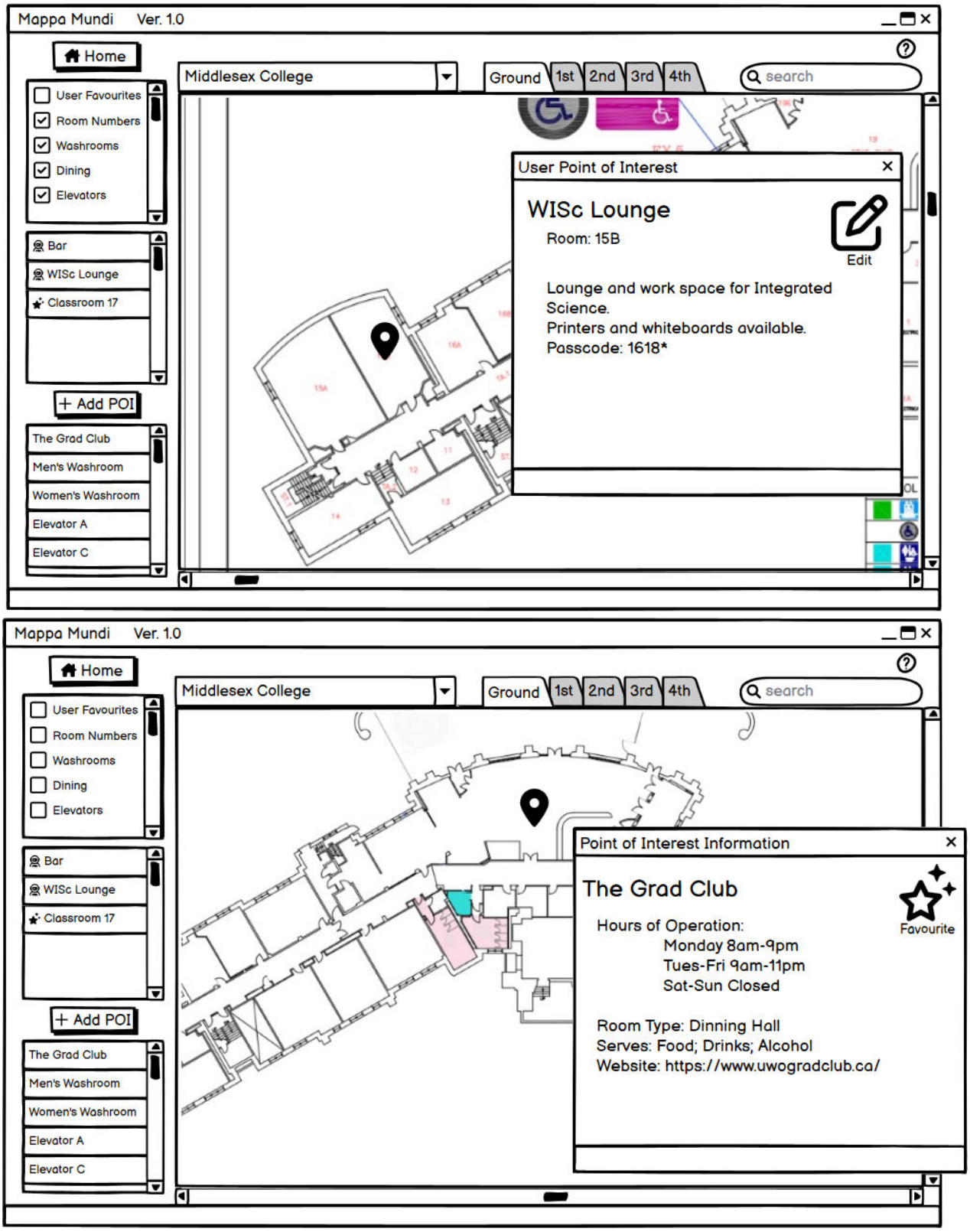
Built-in Point of Interest features a *Favourite* option (pictured above), adding the point of interest to the User's list of favourites (accessed through a button toggled window in Version 3). Toggling the *Favourites* icon on a favourited POI will remove it from the User's list of favourites (pictured below).


Mappa Mundi Ver. 1.0
Home
User Favourites
Room Numbers
Washrooms
Dining
Elevators
Bar
WISc Lounge
Classroom 17
Add POI
The Grad Club
Men's Washroom
Women's Washroom
Elevator A
Elevator C
Middlesex College
Ground
1st
2nd
3rd
4th
search
Point of Interest Information
Middlesex Classroom 17
Favourite
Seating Capacity: 48
Room Type: Lecture
Seating Type: Tab Arm; Fixed
Lectern: Portable
Writing Surface: 16-ft Sliding Chalkboard
Projection Screen: 8-ft Motorized
Window Treatment: Manual Blinds


Mappa Mundi Ver. 1.0
Home
User Favourites
Room Numbers
Washrooms
Dining
Elevators
Bar
WISc Lounge
Add POI
The Grad Club
Men's Washroom
Women's Washroom
Elevator A
Elevator C
Middlesex College
Ground
1st
2nd
3rd
4th
search
Point of Interest Information
Middlesex Classroom 17
Favourite
Seating Capacity: 48
Room Type: Lecture
Seating Type: Tab Arm; Fixed
Lectern: Portable
Writing Surface: 16-ft Sliding Chalkboard
Projection Screen: 8-ft Motorized
Window Treatment: Manual Blinds



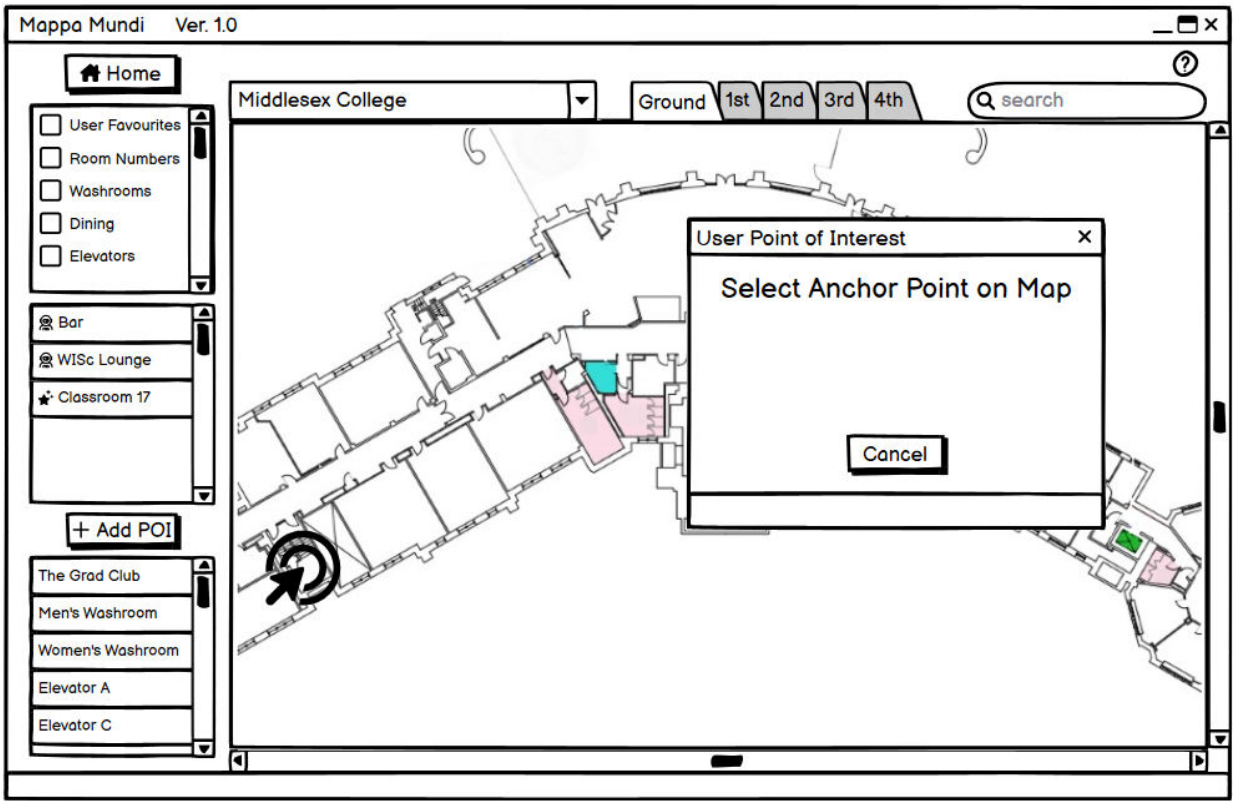
The User can also select Points of Interest available on the current map from the left side lists, separated into User Points of Interest and Built-in Points of Interest (pictured above).

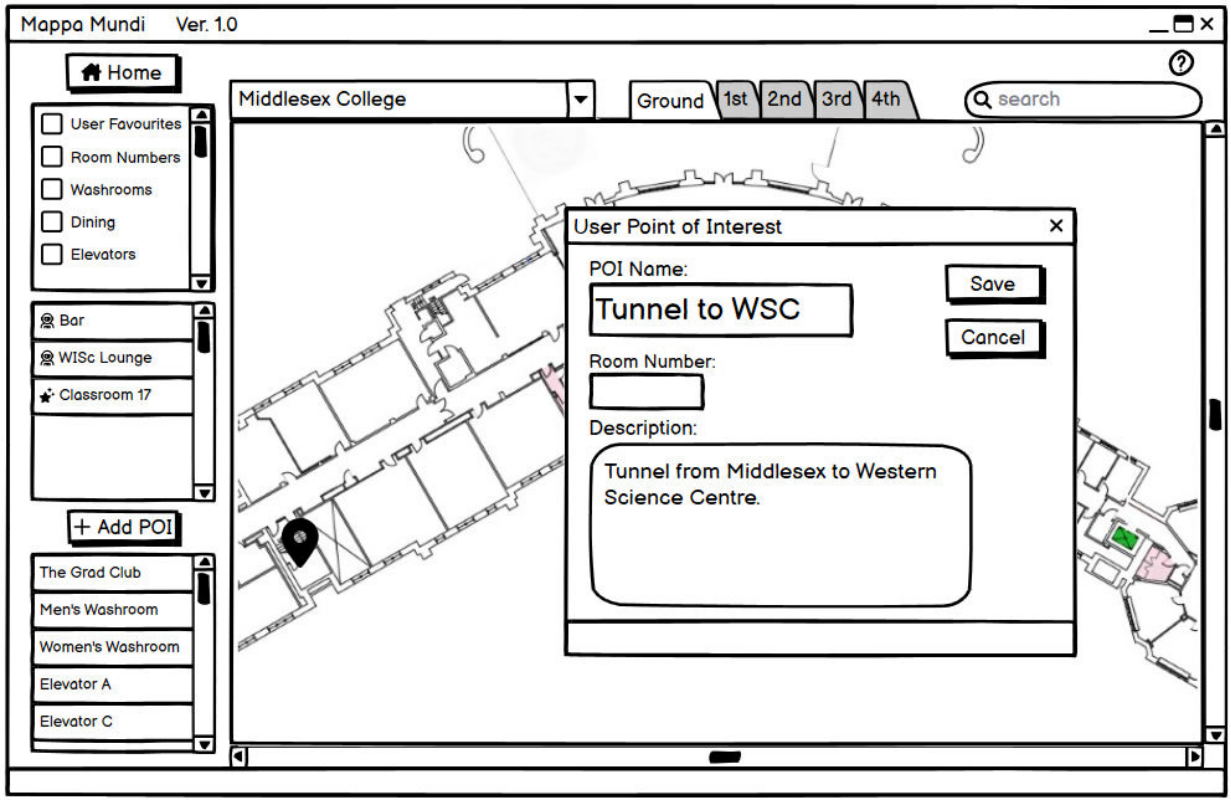


Selecting a Point of Interest from the left side lists will result in the map graphic automatically scrolling to display the POI, the POI will be highlighted, and the POI information window will open, regardless if it is a built-in or user-created Point of Interest (pictured above). (information portrayed in wireframe windows is fabricated).

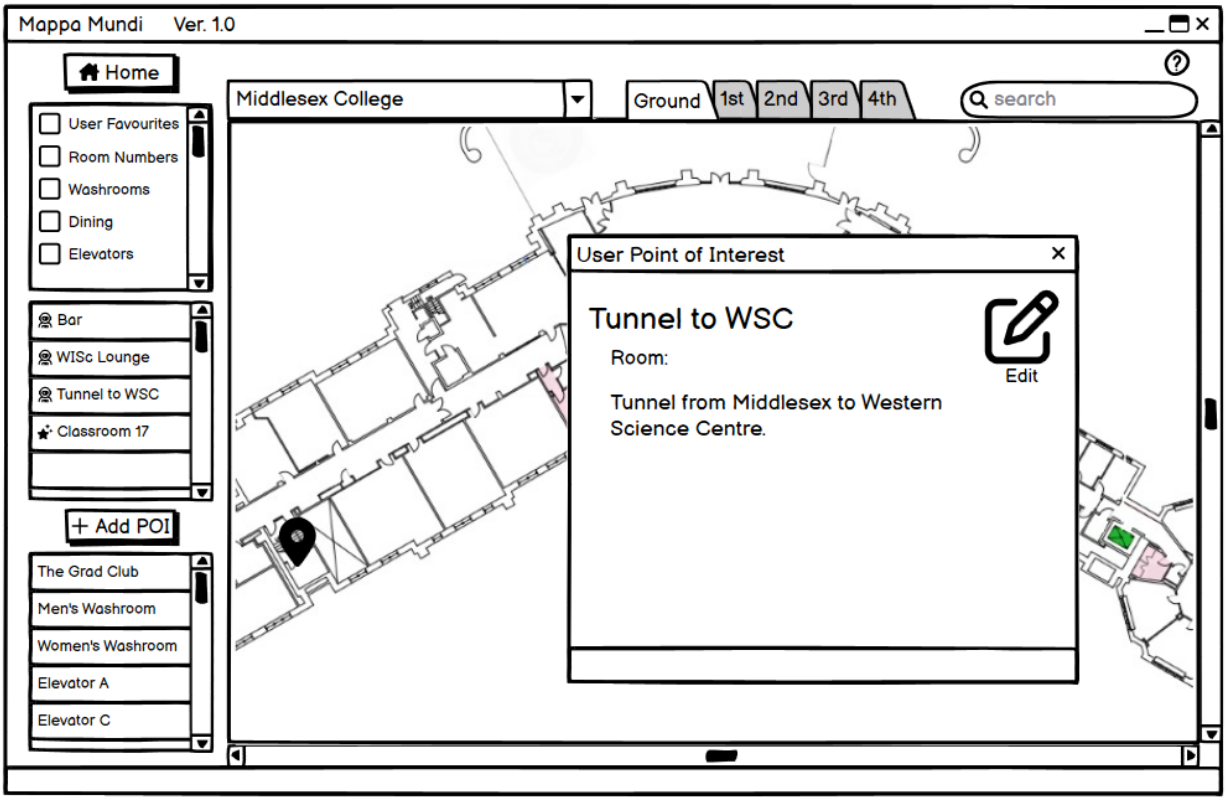
User-Created Point of Interest

The User can create new Points of Interest by toggling the *Add POI* button above or below the User Points of Interest list in all three versions. Upon toggling the *Add POI* button, the User will be prompted to select an anchor point for the new user-created point of interest (pictured below).



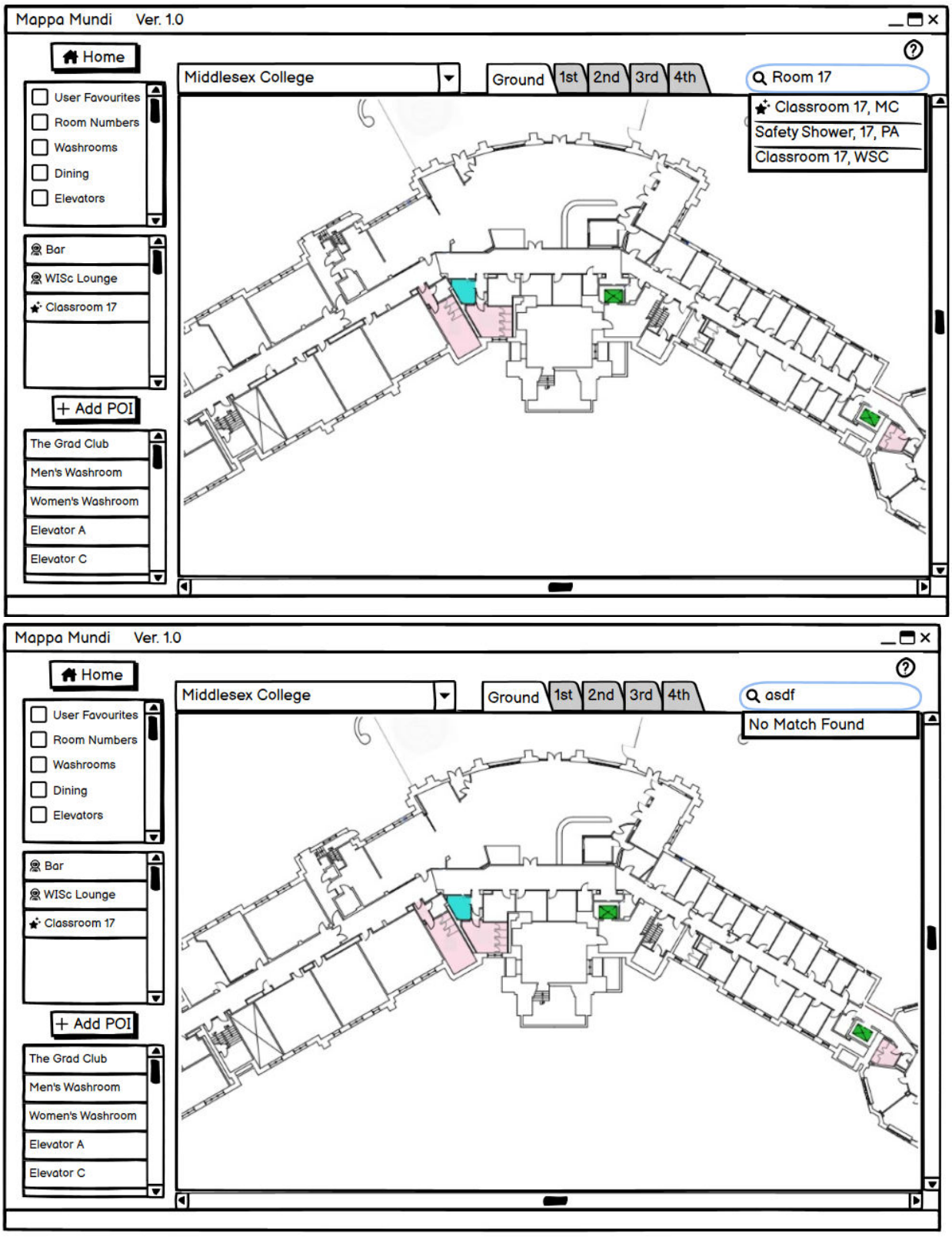


At any point in this process, the User may click *Cancel* on the *User Point of Interest* window (pictured above), and the anchor point and Point of Interest will be discarded. Once the User selects an anchor point by clicking on the map graphic in the desired location, the anchor will be highlighted and linked to a User Point of Interest information window (pictured above on the right). Finally, the User can add the desired information and click *Save* to finalize the user-created point of view (shown below).

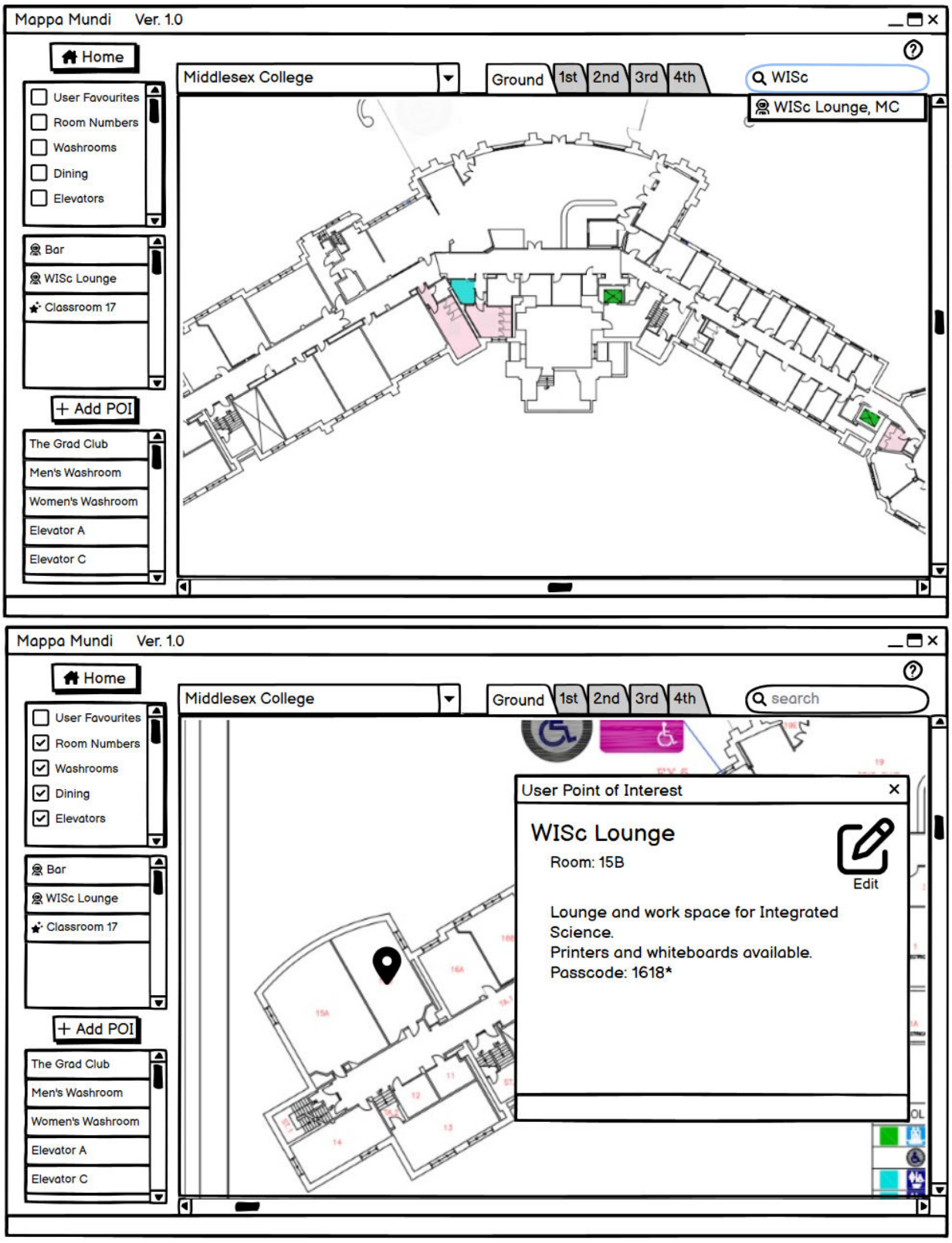


Searching Points of Interest

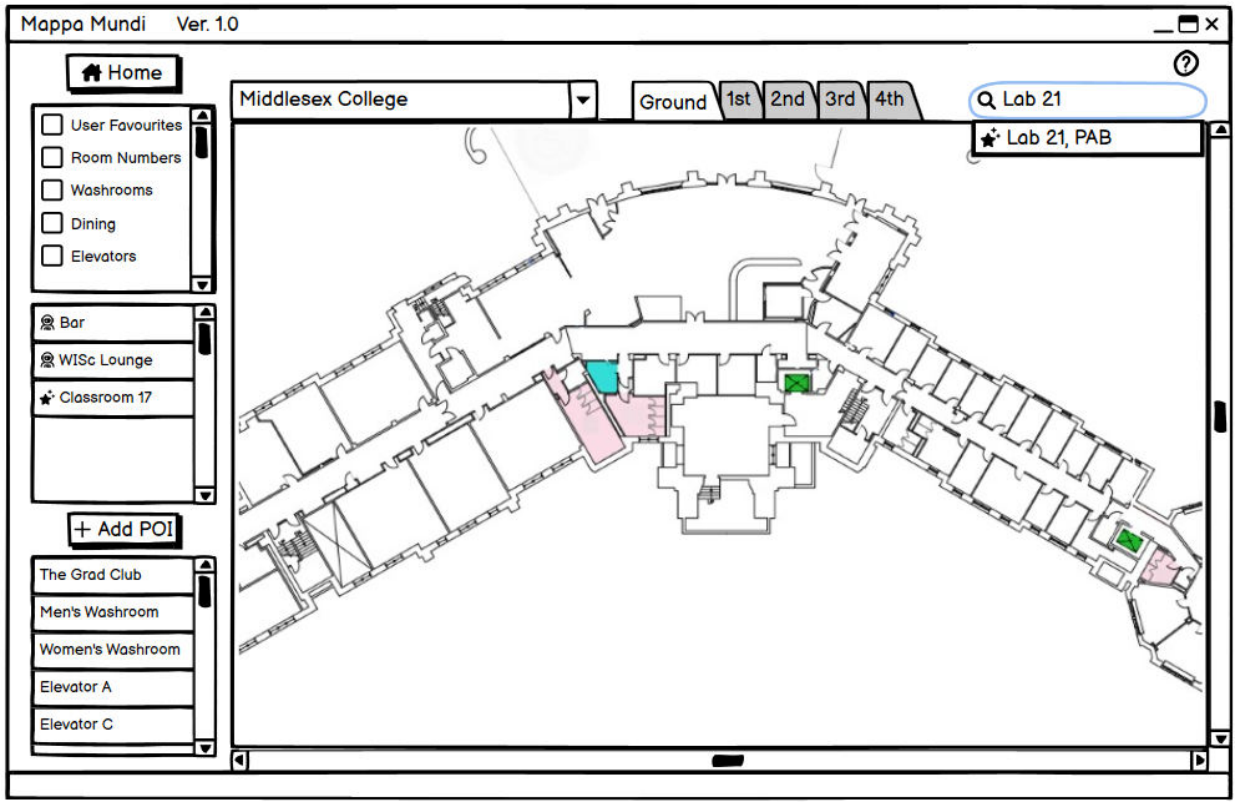
The User may also use the *Search Bar* on the top right of the window above the map graphic to find Points of Interest across all the maps (pictured below). This feature takes in characters from the User and searches them against all built-in Points of Interest, layers, and user-created Points of Interest. The application informs the User (pictured below on the right) if no match is found.

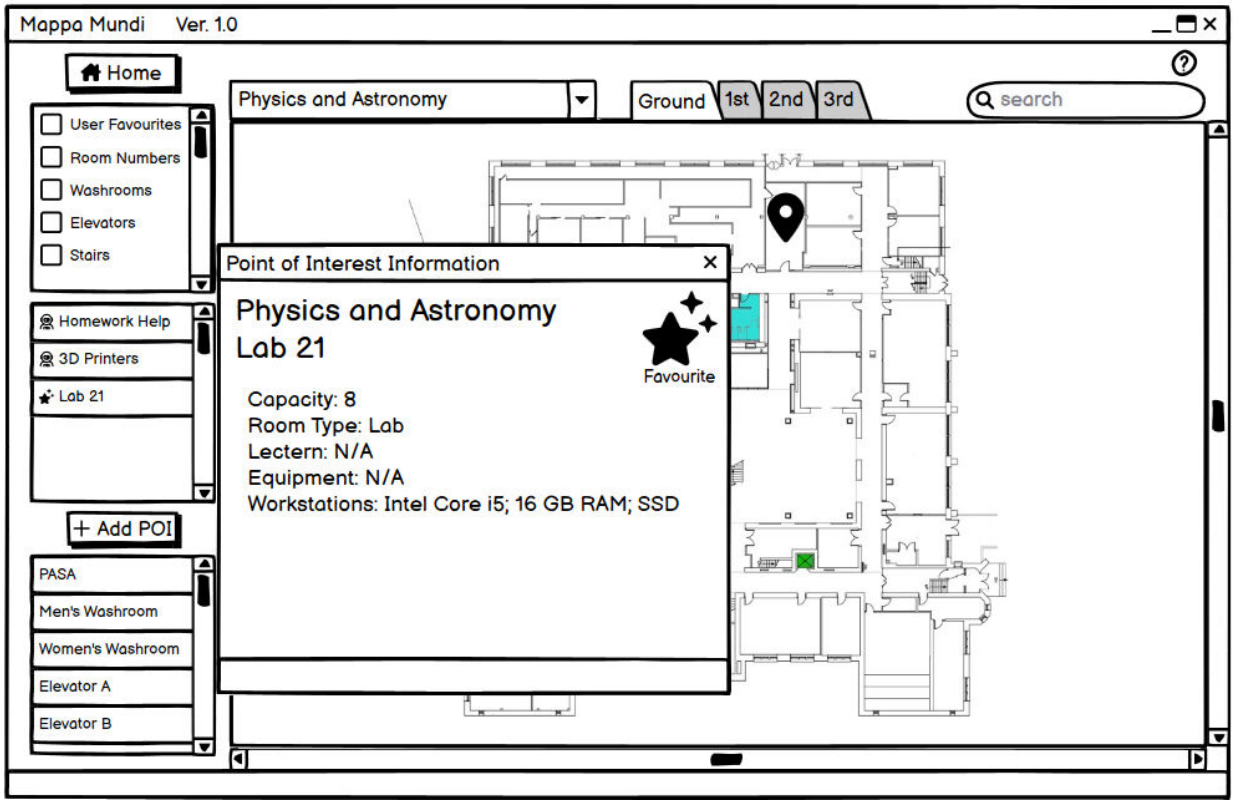


Selecting a Point of Interest from the *Search Bar* to drop down will highlight the POI on the map, scroll to the POI, and display the Point of Interest information window (pictured below).



Selecting a Point of Interest from the *Search Bar* drop-down located on another map causes the map graphic to switch to the map containing the Point of Interest, scroll to the POI, highlight it, and display the Point of Interest's information window (pictured below).





Notes

These design mockups are not finalized, and there will likely be many changes before we are left with a final product. Therefore, we anticipate many more design iterations before eliminating the potential issues in Versions 1-3. Map information displayed in these designs is fabricated; for example, the application will collect correct metadata.

The following link will allow you to download this Balsamiq Wireframes File (.bmpr) from our Confluence space so you may interact with it further:

GISforWesternU.bmpr

File Formats

The data for the project will be stored in several CSV files, detailed below. The CSVs allow us to easily add information, such as user-created points of interest (POI), neatly and keep everything organized; this also means we don't have to use external libraries. In the CSV files listed below, blue items refer to files, and red items refer to primitive data types and variable instances. Each field in each CSV file is given a description and an example.

The map files will be stored as .png files, with POIs removed. Then, the user can select layers to display POIs on top of the base map. We include Middlesex College, Physics and Astronomy, and the Western Science Centre. In addition, each building has a master CSV file, Specific Building's CSV noted below, and each floor in that building will have its own CSV file for POIs on that floor. Finally, we have a file for generic POIs (not user created and might be used across several buildings) and a file for all the user-created POIs.

Buildings CSV:

String Building name | String Building's CSV file | Integer Number of floors | Boolean If a building is favourited

---

String Building name

The name of a building that has a map.

Example:

("Middlesex College")

String Building's CSV file

The filename is used for the data about each building.

Example:

("middlesex.csv")

Integer Number of floors

The number of floors the building has, including the basement and ground floor. For example, if this was 3, the building might have a ground floor, a 1^{st} floor, and a 2^{nd} floor; it could also mean it has a basement, a ground floor, and a 1^{st} floor.

Example:

(4)

Boolean If a building is favourited

Represents if the user has favourited this building. 1 means this building is favourited, and 0 means it is not.

Example:

(1)

Specific building's CSV: Integer Floor level | String Filename of floor graphic

---

Integer Floor level

References the floor number (for example, ground floor, 3rd floor, etc.). Therefore, for every floor in the building, there will be 1 record in this CSV file. As an example, if a

building's CSV file has 5 records, starting from -1 to 3, then there is a basement, a ground floor, and a 1st, 2nd and 3rd floor.

Example:

(4)

String Filename of floor graphic

The filename used for the image of each floor ("floor1.png", "floor2.png", etc.).

Example:

("middlesex_college_floor2.png")

Specific floor's CSV:

String POI index | String POI coordinates | Boolean IsPOIFavourited

---

String POI index

We can understand which POI is on this floor with a POI index that can be understood as follows:

1. If the POI index's first character is an "x" (as in the string "x8"), then:
    a. Truncate the x (string "x8" becomes "8"), convert this truncated number to an Integer ("8" becomes an Integer=8), then lookup this Integer in the **User** POI CSV file
2. Else (such as the string "15"), convert the whole String to an Integer ("15" becomes an Integer=15), then lookup this Integer in the **Generic** POI CSV file

Thus, with 1 field, we can identify whether the record is giving the coordinates of a generic POI (like a bathroom) or a user POI (like a specific room the user frequents) and which POI that is.

Example:

("x3")

String POI coordinates

We can save coordinates as (872, 654) as a String, such as "872,654", and delimit using anything but the delimiter used in the CSV file itself. In our case, we will delimit our CSV files with a "|," so using a comma inside the strings of this field would be unambiguous (and, therefore, allowed).

Example:

("1920,1080")

Boolean IsPOIFavourited

Provides a way to store the user's favourited POIs. 1 represents that this POI is favourited, and 0 means it is not.

Example:

(1)

Generic POI CSV:

Integer Index of item | String Name of item | String Room number | String Description | String Filename of POI icon

---

Integer Index of item

This is an index to uniquely identify a type of generic POI that might be seen in different buildings. For example, we might say that a men's bathroom has an index of 4, so in one of the Specific Floor CSV files, if we mention a generic POI with an index of 4, it would mean a men's bathroom.

Example:

(1)

String Name of item

The index refers to this familiar English name (bathroom, elevator, etc.).

Example:

("Elevator")

String Room number

This is the room number in the building that the POI is in.

Example:

("15B")

String Description

This is the description of the POI; for example, a restaurant might have its open hours listed in this String.

Example:

("Open hours: 10 am-5 pm")

String Filename of POI icon

The filename of the POI icon used.

Example:

("elevator.png")

User POI CSV:

Integer Index of item | String Name of item | String Room number | String Description

---

Integer Index of item

This is an index to identify a user-created POI uniquely. So, for example, if a user created a POI such as "Locker," we can append this file with a unique index and form a record in this file.

Example:

(1)

String Name of item

The index refers to this familiar English name (locker, closest water dispenser, etc.).

Example:

("2212 TA meeting room")

String Room number

This is the room number in the building that the POI is in.

Example:

("15B")

String Description

This is the description of the user's POI that appears when selected.

Example:

("My favourite study spot!")

Development Environment

The development environment for the team will be Visual Studio Code, a lightweight, open-source code editor that supports a wide range of programming languages, including Java. Visual Studio Code provides powerful code editing features, such as syntax highlighting, code completion, and debugging, which will help the team to develop the software more efficiently.

In addition to Visual Studio Code, the team will use Bitbucket, a web-based hosting service for version control repositories, to control different versions of the code base. Bitbucket will enable the team to collaborate on the software development process more effectively, allowing them to manage changes to the codebase and track issues and bugs.

The team has also decided to use external libraries to simplify the development process. For example, JavaFX will be used for the Graphical User Interface (GUI) as it can target desktop applications and provides a modern GUI library. This will enable the team to create a visually appealing and user-friendly desktop application that runs on Windows 10.

The team has also decided to use Javadoc to document the codebase. Javadoc is a tool for generating API documentation in HTML format from Java source code. It provides an easy way to document code, making it easier for other developers to understand the codebase and contribute to the software development process.

Finally, the team has decided to use JUnit to write and execute automated tests. JUnit is a popular testing framework for Java that provides a simple way to write unit tests for code. It will enable the team to test the software more efficiently and ensure it is reliable and bug-free.

Patterns

As we embark on the GIS design of the University Campus project, we have researched several software patterns that could be useful in our software design. These patterns can help us create a more robust and scalable software solution that meets the requirements of our project.

Architectural Pattern: Model-View-Controller (MVC)

We will structure our software using the Model-View-Controller (MVC) architectural pattern. This pattern separates the concerns of data modelling, user interface, and control logic into three distinct components, making it easier to maintain and modify each element without affecting the others. We will apply this pattern to our user interface design, which will display the GIS data to users. We can create a more modular and flexible software solution by separating the user interface from the data modelling and control logic.

Reference: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

Object-Oriented Pattern: Singleton

We will use the Singleton pattern to ensure that only one instance of certain classes is created. We will use this pattern to ensure we have only one example of the GIS data in memory at any given time. This pattern will help us reduce memory usage and improve performance by avoiding the creation of multiple instances of the same data.

Reference: https://en.wikipedia.org/wiki/Singleton_pattern

UI/UX Pattern: Progressive Disclosure

We will use the Progressive Disclosure pattern in our user interface design to present information to users gradually and intuitively. This pattern involves giving users only the most essential and relevant information at first and progressively revealing additional details as needed. This approach can help us avoid overwhelming users with too much information and provide a more user-friendly experience. We will apply this pattern to our map display, initially showing only the most critical data, such as buildings and major roads, and gradually revealing additional data, such as green spaces and parking lots, as users zoom in or interact with or interact with the map.

Reference: https://www.nngroup.com/articles/progressive-disclosure/

Architectural Pattern: Layered Architecture

We will use the Layered Architecture pattern to organize our software components into logical layers, each with a specific responsibility. This pattern involves dividing our software into layers, such as the presentation layer, business logic layer, and data access layer. Each layer communicates only with the layers immediately above and below, making modifying or replacing a specific layer easier without affecting the entire system. We will apply this pattern to our GIS data management and processing, with separate layers for data storage, data processing, and user interface.

Reference: https://en.wikipedia.org/wiki/Multitier_architecture

UI/UX Pattern: Feedback Loop

We will use the Feedback Loop pattern in our user interface design to provide users immediate feedback on their actions. This pattern involves providing visual or auditory cues to users in response to their activities, such as highlighting a selected item or playing a sound when a button is pressed. This approach can help us create a more interactive and engaging user interface and help users understand the effects of their actions on the system. In addition, we will apply this pattern to our map interactions, such as zooming in or selecting a point of interest. Finally, we will provide visual feedback to users to indicate the success or failure of their actions.

Reference: https://uxdesign.cc/how-to-design-feedback-loops-6c8df6e03a0f

Conclusion

These patterns will help us create a more effective and efficient GIS design for the University Campus project. Using these proven patterns, we can ensure that our software solution is scalable, maintainable, and user-friendly while making the development process as streamlined as possible. Furthermore, we can avoid antipatterns that may create difficulties when testing and maintaining the codebase. Therefore, we must take advantage of patterns in our project's UI/UX/OOP design to avoid reinventing and struggling with implementing already created, tried, and tested features.