1) Main Page

WEGIS (Western Expedient Geographical Information System)

This Document outlines the Design Documentation for WEGIS in CS2212B

Version	Date	Author(s)	Summary of Changes
	04 Mar 2023	Jane	Made a basic structure of UML class diagram
	05 Mar 2023	Marc Crasto	Added an incomplete UI Mockup
	06 Mar 2023	Matthew Morelli	Finished introduction, summary, file formats page, and the remaining UI mockup descriptions
	06 Mar 2023	Daniel, Jane	Finished UML class diagram
	06 Mar 2023	Foster Giggie	Completed development environment and patterns. Updated main page

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.

Objectives

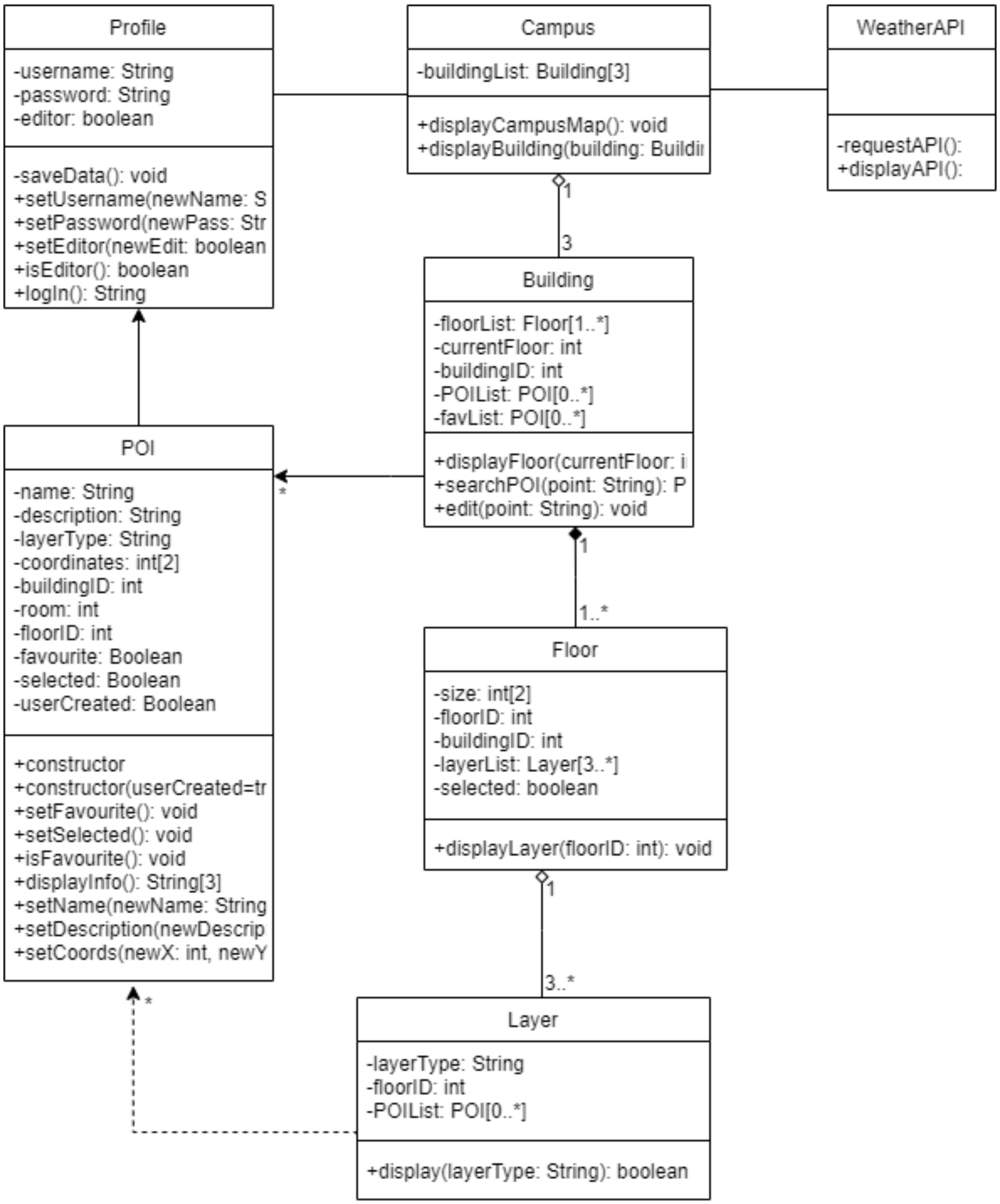
The objectives of this design document is for the reader to have a better understanding on how the UI and implementation for the program will look like. There are many reasons behind the decisions being made in regards to each design choice, and this document outlines the decision making process involving:

- Classes
- User Interface
- File Formats
- Development Environment

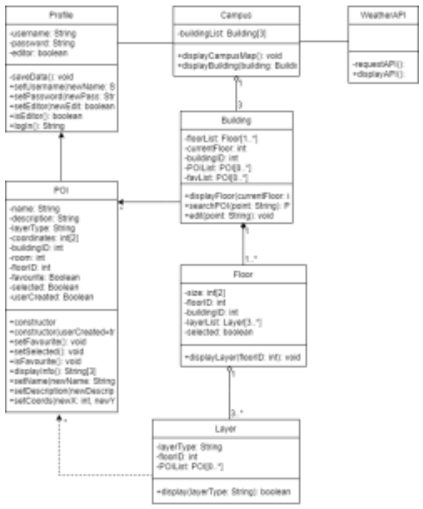
References

Requirements Documentation

Project Specification


Profile
-username: String
-password: String
-editor: boolean
-saveData(): void
+setUsername(newName: S
+setPassword(newPass: Str
+setEditor(newEdit: boolean
+isEditor(): boolean
+logIn(): String
Campus
-buildingList: Building[3]
+displayCampusMap(): void
+displayBuilding(building: Buildi
WeatherAPI
-requestAPI():
+displayAPI():
1
3
Building
-floorList: Floor[1..*]
-currentFloor: int
-buildingID: int
-POIList: POI[0..*]
-favList: POI[0..*]
+displayFloor(currentFloor: i
+searchPOI(point: String): P
+edit(point: String): void
*
POI
-name: String
-description: String
-layerType: String
-coordinates: int[2]
-buildingID: int
-room: int
-floorID: int
-favourite: Boolean
-selected: Boolean
-userCreated: Boolean
+constructor
+constructor(userCreated=tr
+setFavourite(): void
+setSelected(): void
+isFavourite(): void
+displayInfo(): String[3]
+setName(newName: String
+setDescription(newDescrip
+setCoords(newX: int, newY
1
1..*
Floor
-size: int[2]
-floorID: int
-buildingID: int
-layerList: Layer[3..*]
-selected: boolean
+displayLayer(floorID: int): void
1
3..*
*
Layer
-layerType: String
-floorID: int
-POIList: POI[0..*]
+display(layerType: String): boolean

3) Class Diagrams



Profile
-username: String
-password: String
-editor: boolean
-saveData(): void
+setUsername(newName: String): void
+setPassword(newPass: String): void
+setEditor(newEdit: boolean): void
+isEditor(): boolean
+logIn(): String

The class Profile is used to define any user's individual identity in the program. It stores the user's username and password as Strings so that they can be referenced while logging in. The boolean variable editor is used to allow administrators to edit the built-in points of interest (POIs). When a user attempts to edit a built-in POI, the program will check the user's "editor" status using the isEditor() method, which returns the editor variable. If the variable is true, the user is an administrator and has permission to edit. Otherwise, if the variable is false, the user is not allowed to edit. The method saveData() saves the user's data persistently so that it can be referenced later. It is used in the constructor after initial data is collected. The methods setUsername(), setPassword(), and setEditor() are used to update the username, password, and editor variables respectively. All three setter methods call saveData() at the end in order to ensure the updated information is saved persistently. The method logIn() processes the information entered by the user and allows or blocks admission into the map viewer. It ensures that the data entered on the login page is the same as the user's saved data. If it matches, the user has permission to continue into the application. Otherwise, we allow the user to try again 3 times. If the user fails all 3 times, we ask them to update their password or create a new account.

POI
-name: String
-description: String
-layerType: String
-coordinates: int[2]
-buildingID: int
-room: int
-floorID: int
-favourite: Boolean
-selected: Boolean
-userCreated: Boolean

+constructor
+constructor(userCreated=true)
+setFavourite(): void
+setSelected(): void
+isFavourite(): void
+displayInfo(): String[3]
+setName(newName: String): void
+setDescription(newDescrip: String): void
+setCoords(newX: int, newY: int)

The class POI is used to define a point of interest on the map. It contains a name and a short description saved as Strings. Another String variable, layerType, denotes what kind of layer the POI belongs to. The array coordinates contain two integers representing the POI's location on the map. The integer variables buildingID and floorID are used to identify the location of the POI with respect to the floor and building that it is in. The room integer corresponds with the room number. The boolean favourite denotes whether or not the POI has been selected as a favourite POI. The userCreated boolean denotes whether or not the POI was created by a user or an admin. The selected Boolean is set to true when the POI is selected. When true, the POI will be highlighted and the method displayInfo() will be called. There are two constructors for this class, with the difference being that one sets userCreated to true, while the other sets it to false. When the user creates a POI, it will automatically use the constructor that sets userCreated to true. The methods setFavourite() and setSelected() are used to update the favourite and selected booleans at the user's request. The method isFavourite() allows favourite POIs to be recognized and added to a list in the Building class. The method displayInfo() collects the variable's name, description, and room into a String array which is sent over to the GUI component in order to be displayed. The setters setName(), setDescription(), and setCoords() are used when editing a POI in the Building class.

Layer
-layerType: String
-floorID: int
-POIList: POI[0..*]
+display(layerType: String): boolean

The class Layer is used to hold a certain category of POI. The String variable layerType is used to denote the type of layer. The int variable floorID is used to mark which floor the layer is associated with. The array POIList is taken from the Building class and contains all POIs on that specific layer and floor. The method display() is used to communicate with the GUI that the layer should be displayed if the return value is true, or hidden if it is false.

Floor
-size: int[2]
-floorID: int
-buildingID: int
-layerList: Layer[3..*]
-selected: boolean
+displayLayer(floorID: int): void

The class Floor is used to hold the information for a certain floor. The integer array size is used by the GUI to display the map as its specific size. The integers floorID and buildingID are used to denote the floor number and which building it is part of. The layerList array holds all the layers associated with the floor. The boolean selected is true when the associated floor is currently being displayed and false otherwise. The method displayLayer() ensures that the floor is selected and calls display() from the Layer class to communicate with the GUI.

Building
-floorList: Floor[1..*]
-currentFloor: int
-buildingID: int
-POIList: POI[0..*]
-favList: POI[0..*]

+displayFloor(currentFloor: int): void +searchPOI(point: String): POI +edit(point: String): void

The Building class is used to hold information for a certain building. The array floorList contains the floors that are in the building from the ground up. The integer currentFloor keeps track of what floor is currently selected. The integer buildingID is used to identify the building. The array POIList contains a list of all POIs in the system in order to search for the POIs throughout the whole building. The array favList contains all of the POIs from POIList that are favourited. The method displayFloor() communicates with GUI to display a certain floor. The searchPOI method takes a String entered by the user and searches the POIList in order to find it, returning the POI that is being looked for. The edit() method is used to edit the POIList taking input from the user (only for user-created POIs) or the admin. It updates a specific POI denoted by a String, editing the information and updating the POIList and favList in this class, as well as the POIList in the Layer class.

Campus
-buildingList: Building[3]
+displayCampusMap(): void +displayBuilding(building: Building): void

The Campus class is used to hold information about the initial Campus map. The array buildingList holds the 3 buildings that are viewable. The method displayCampusMap() communicates with GUI to display the campus map. It is called immediately after the login screen is successful. It also calls displayAPI() from the WeatherAPI class. The method displayBuilding() will communicate with GUI to display the selected building.

Weather API
-requestAPI(): +displayAPI():

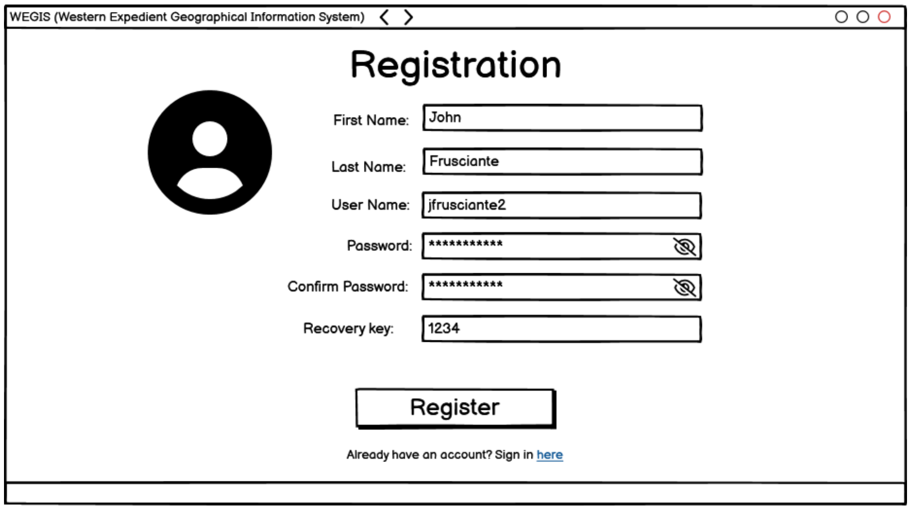
The WeatherAPI class is used to read and display the weather from an external API. The method requestAPI() requests the weather from the API. The method displayAPI() displays the weather that has been requested.

4) User Interface Mockup

File

Images

1) Registration Page: This is the registration page. There is also a recovery key that each user and developer will have in case they forgot their password. There is also the 'Sign in here' option which takes the user to the login page in case they already have an account.

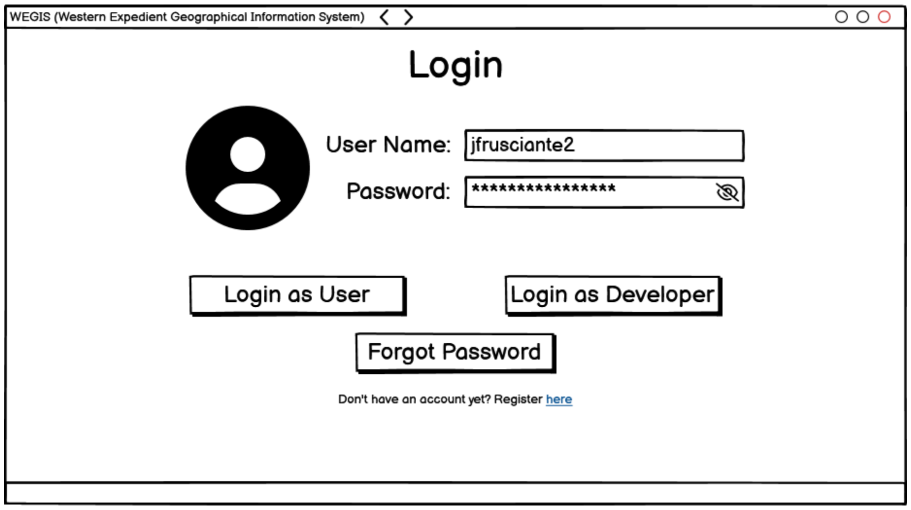


Registration

First Name:	John
Last Name:	Frusciante
User Name:	jfrusciante2
Password:	***********
Confirm Password:	***********
Recovery key:	1234

Already have an account? Sign in here

2) Login: Ask the user to enter the username and password. They can either log in as a developer or as a user. There is also the forgot password option.


WEGIS (Western Expedient Geographical Information System)

Login

User Name: jfrusciante2

Password: ****************

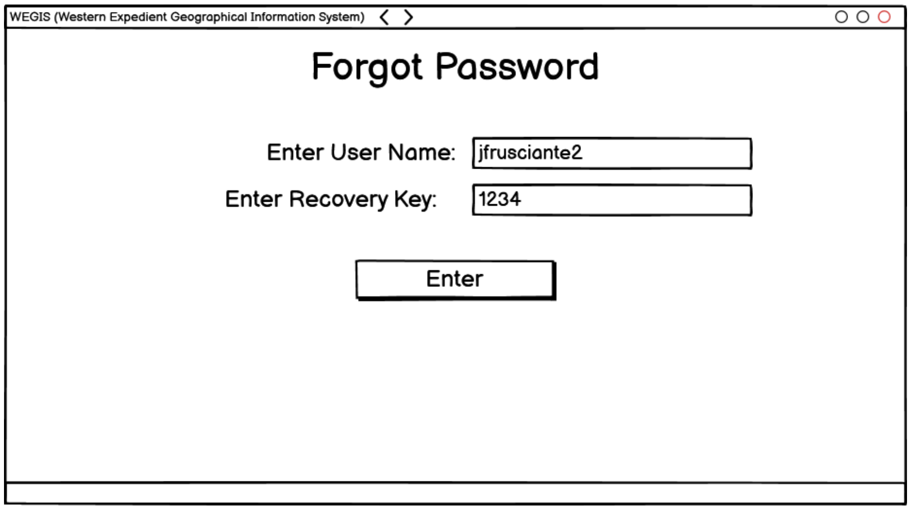
Login as User

Login as Developer

Forgot Password

Don't have an account yet? Register here

3) Forgot Password: Allows the user to enter their recovery key instead of their password if they can not remember their password.


WEGIS (Western Expedient Geographical Information System)

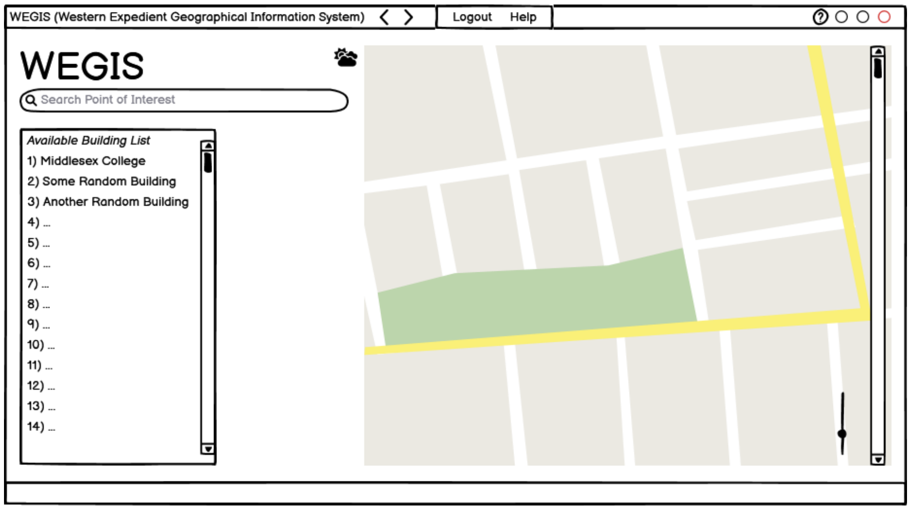
Forgot Password

Enter User Name: jfrusciante2

Enter Recovery Key: 1234

Enter

4) Buildings Map: Displays a list of available buildings for the user to choose from, and displays the map of the building selected.


WEGIS (Western Expedient Geographical Information System)

Logout Help

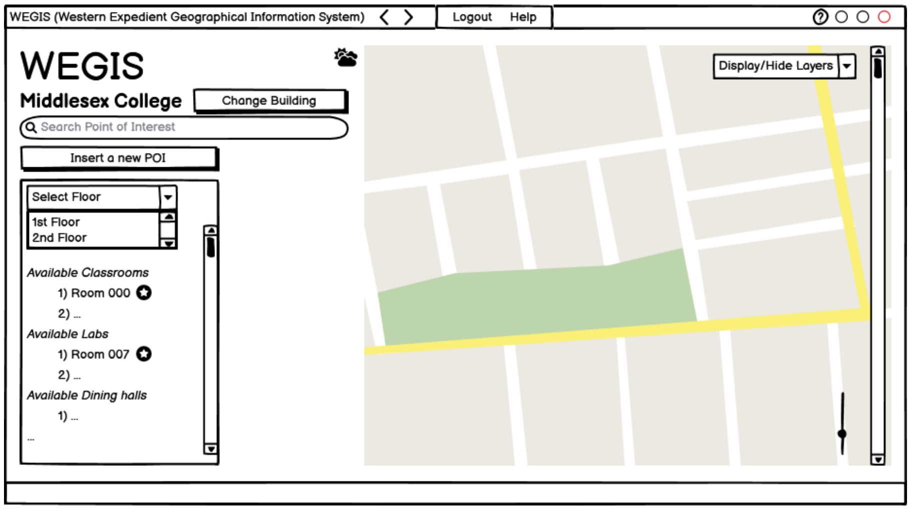
WEGIS

Search Point of Interest

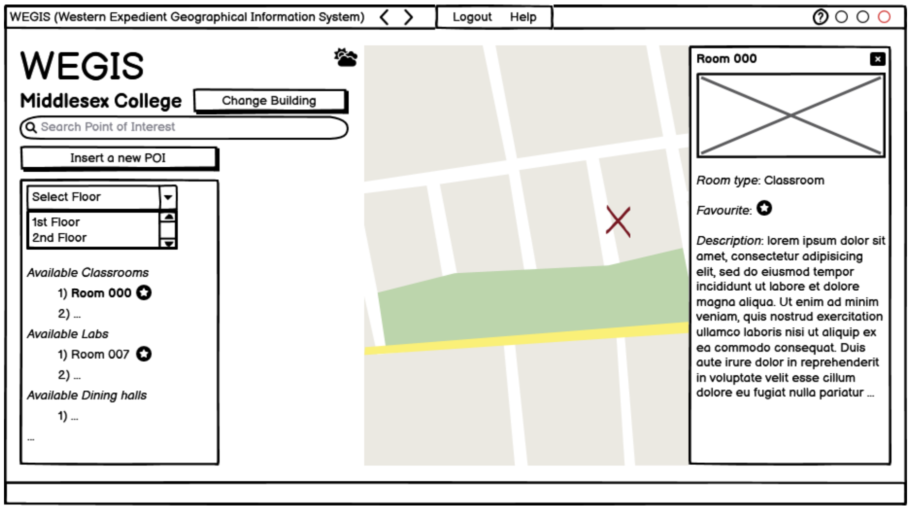
Available Building List

1) Middlesex College
2) Some Random Building
3) Another Random Building
4) ...
5) ...
6) ...
7) ...
8) ...
9) ...
10) ...
11) ...
12) ...
13) ...
14) ...

5) Floor for Selected Building Map: Allows the user to see a drop-down list of floors for the selected building, and displays the selected floor map.



6) Selected Room: Displays information about the selected room.



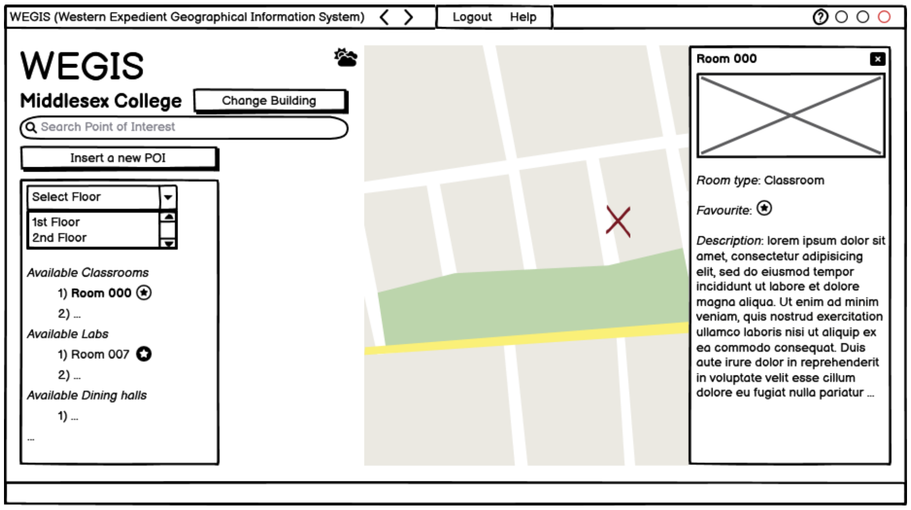
Room 000

Room type: Classroom

Favourite:

Description: lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur ...

7) Marking as Favourite



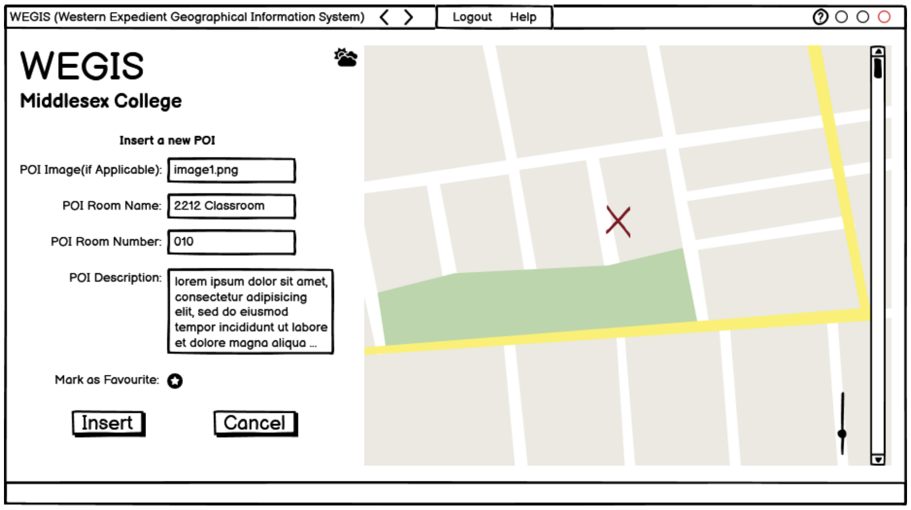
Room 000

Room type: Classroom

Favourite:

Description: lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur ...

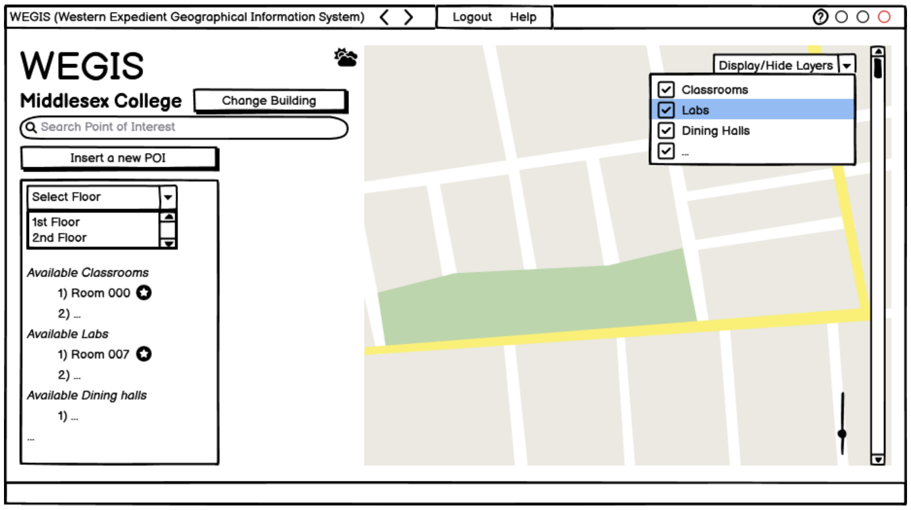
8) Insert a new POI (user): Displays text boxes for the user to enter an image for a new point of interest, the name for the new POI, the room number, description, and whether or not to mark it as favourite.



Insert a new POI

POI Image(if Applicable):	image1.png
POI Room Name:	2212 Classroom
POI Room Number:	010
POI Description:	lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua ...
Mark as Favourite:	✓

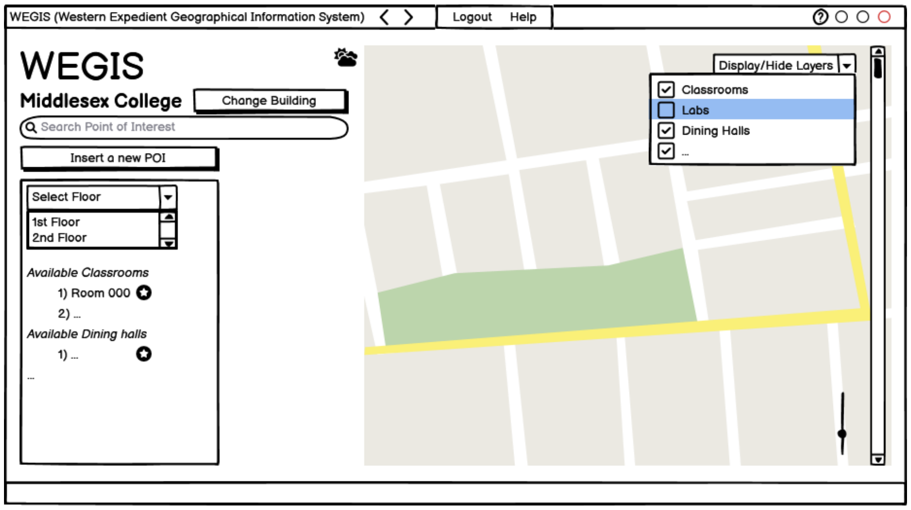
9) Display/Hide Layers: A drop-down menu that lets you toggle the currently active layers.



Available Classrooms
1) Room 000
2) ...
Available Labs
1) Room 007
2) ...
Available Dining halls
1) ...
...

- [✓] Classrooms
- [✓] Labs
- [✓] Dining Halls
- [✓] ...

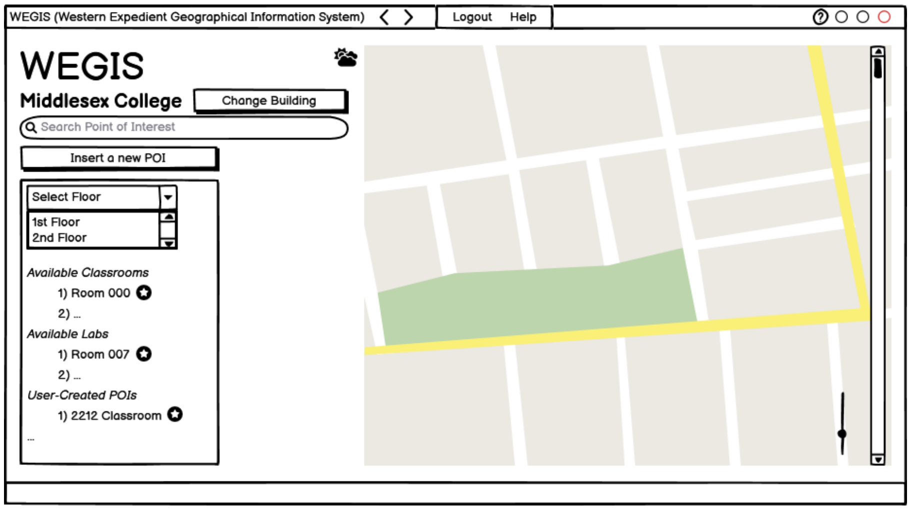
10) Hide Layers



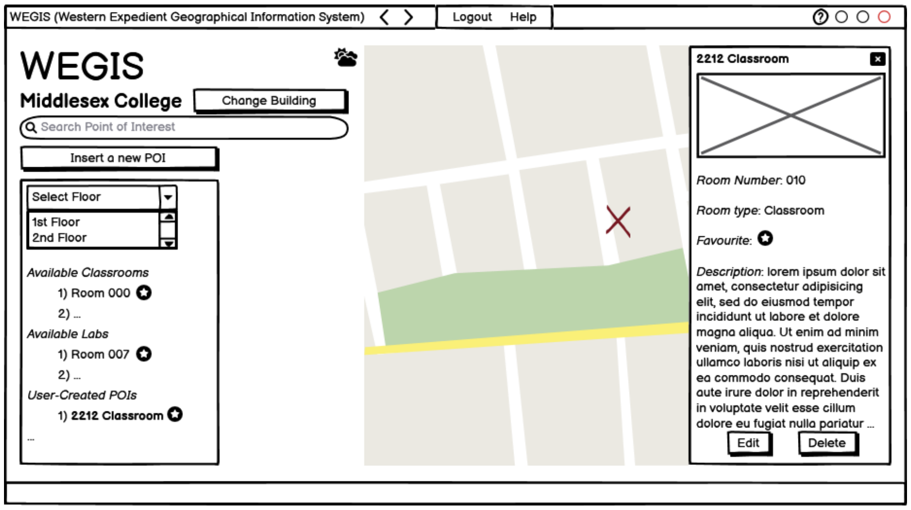
Available Classrooms
1) Room 000
2) ...
Available Dining halls
1) ...
...

- [✓] Classrooms
- [ ] Labs
- [✓] Dining Halls
- [✓] ...

11) User-Created POIs: Displays a list of user-created POIs that the user can click and jump to on the map.



12) Selecting User-Created POI



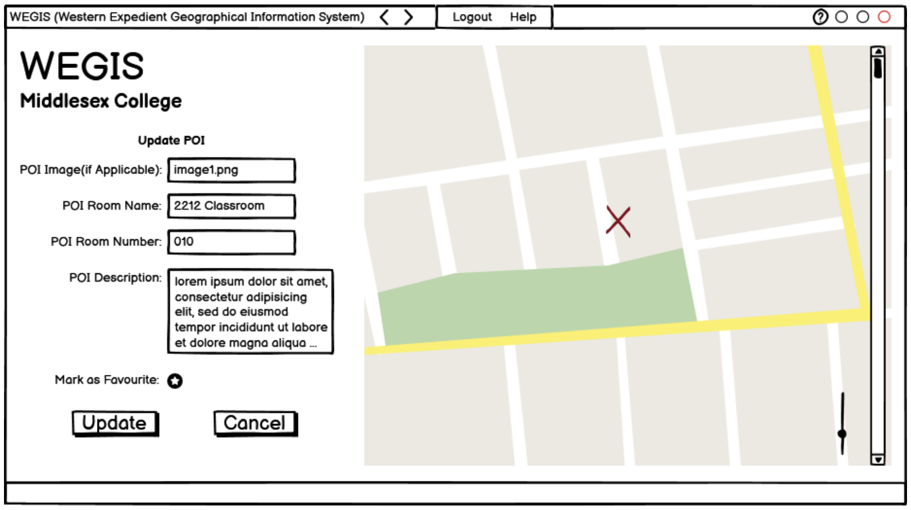
Room Number: 010

Room type: Classroom

Favourite:

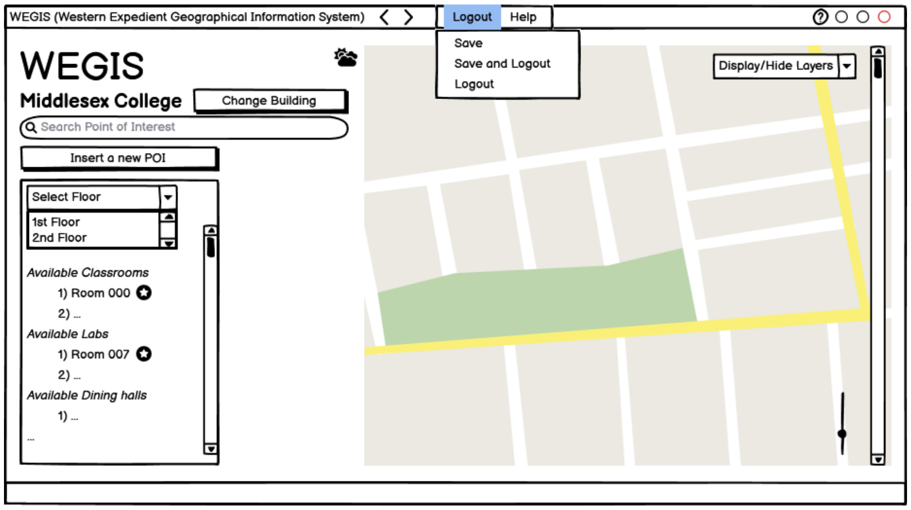
Description: lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur ...

13) Edit User-Created POI: A screen that lets the user edit the information of a user-created POI.

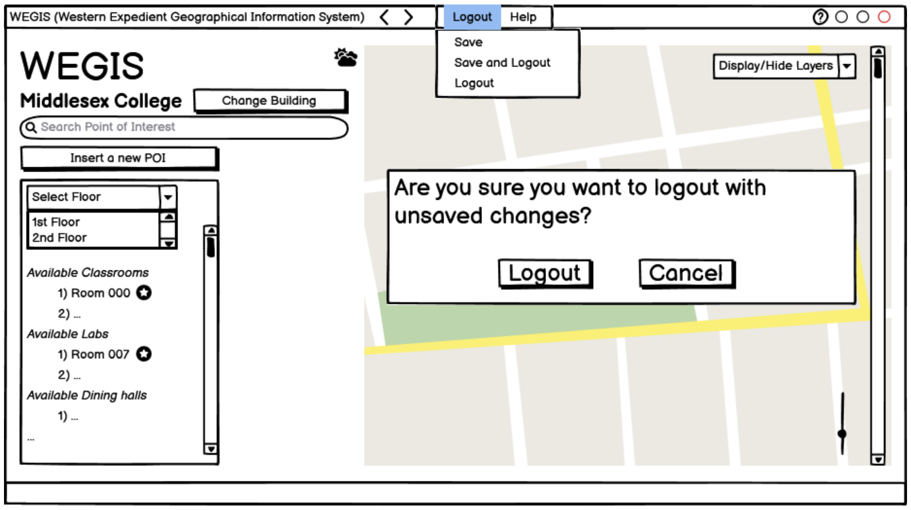


POI Image(if Applicable):	image1.png
POI Room Name:	2212 Classroom
POI Room Number:	010
POI Description:	lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua ...
Mark as Favourite:	

14) Exit/Close and Navigation: A drop-down menu that allows the user to either save, logout, or both,

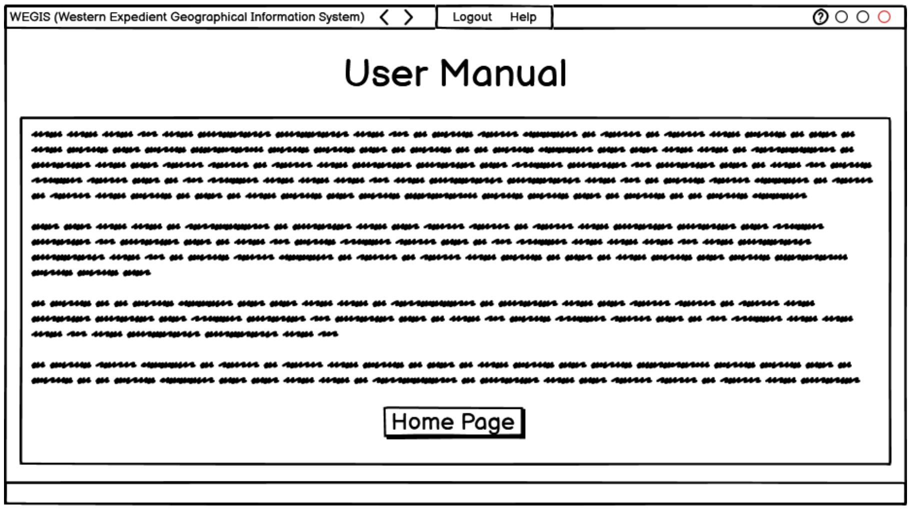


15) Logout with Unsaved Changes: A prompt will appear asking you if you are sure you want to logout without saving your changes incase the user did not mean to logout with unsaved changes.

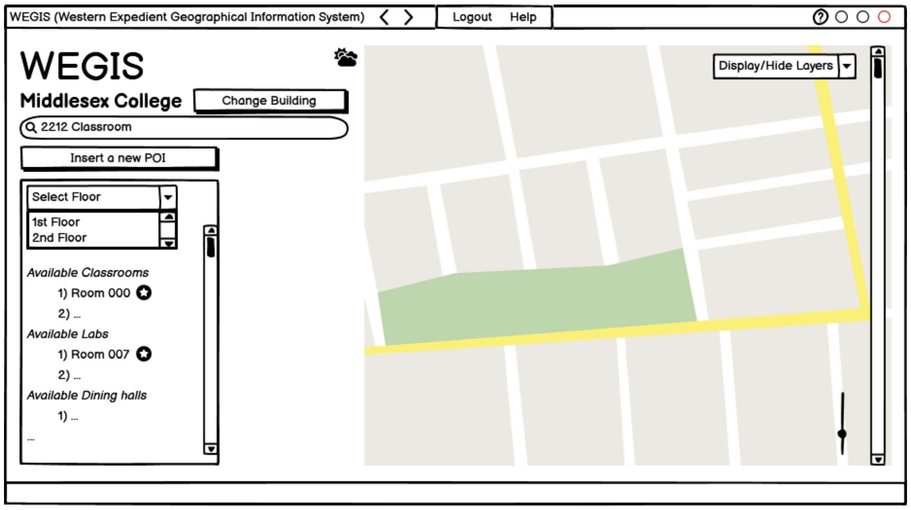


Are you sure you want to logout with unsaved changes?

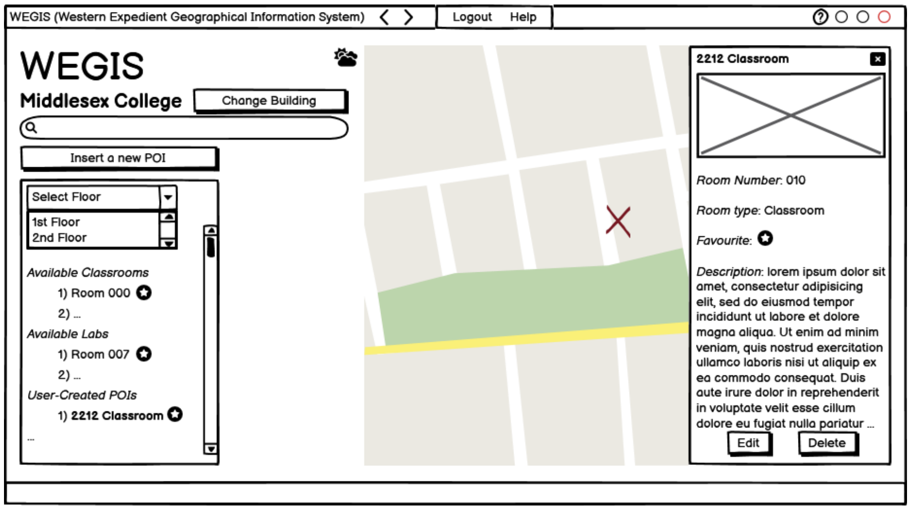
16) Help Guide: Clicking on the help icon or button above will display this wireframe.



17) Search: Allows the user to search through the available POIs.



18) Search Success: If a result matches your search, the information on it will be displayed on the right side of the screen.



2212 Classroom

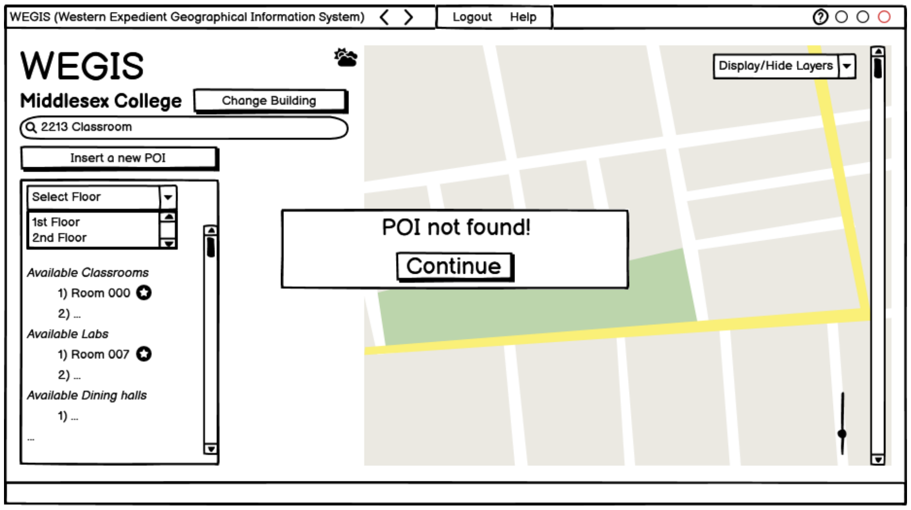
Room Number: 010

Room type: Classroom

Favourite:

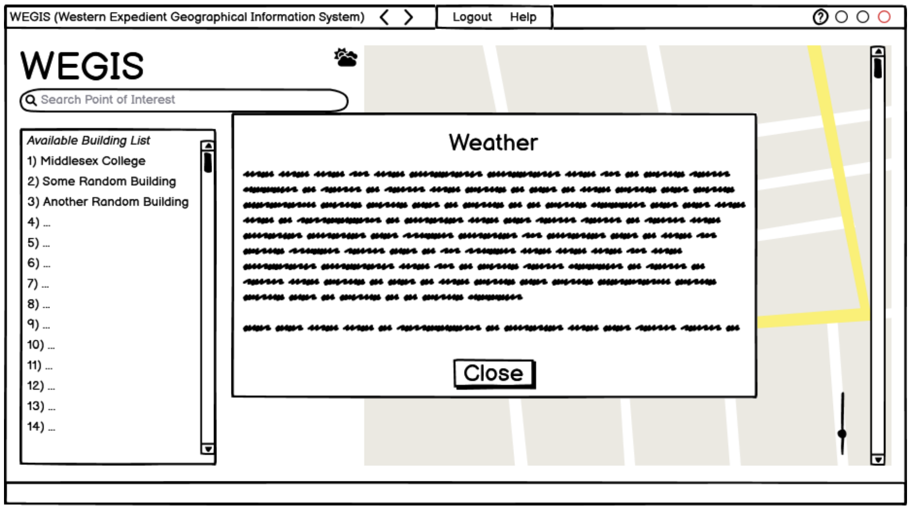
Description: lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur ...

19) Search Failed: If nothing matches your search, a box will appear informing the user that there was no POI found that matches their search.

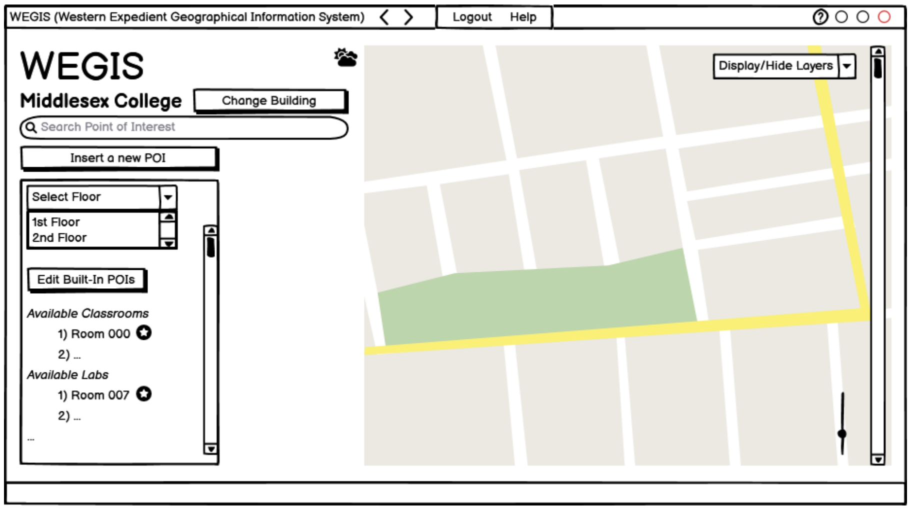


POI not found!

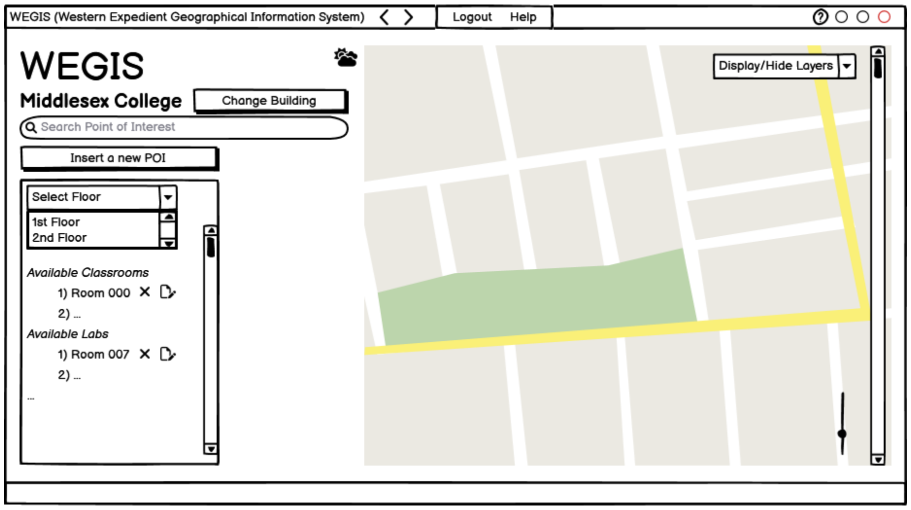
20) Weather: After clicking the weather icon next to the map, a pop-up message will be displayed letting the user know what the current weather is.



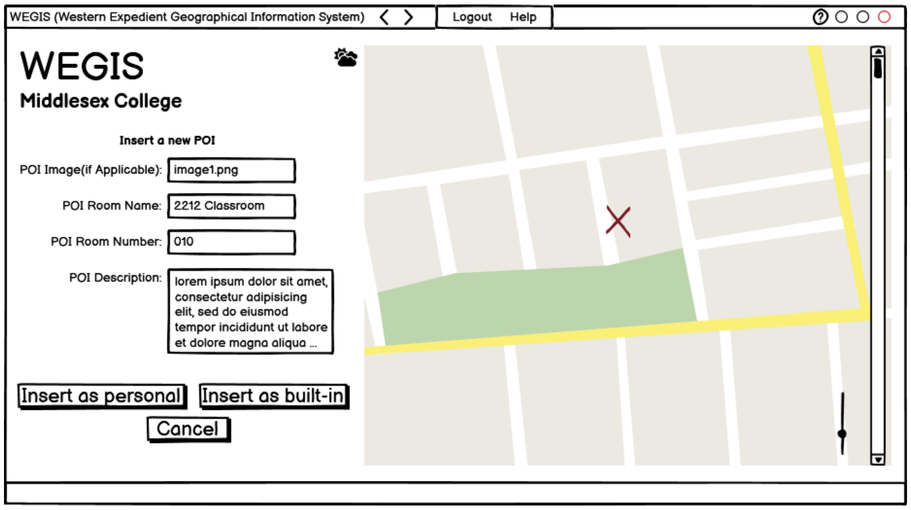
21) Editing Built-in POIs (developer): Clicking on 'Edit Built-in POIs' will lead to the next image.



22) Editing Built-in POIs (developer) 2



23) Insert a new POI (developer): Clicking on the 'Insert a new POI' (previous image) in developer mode will lead to this wireframe. Almost the same as a user but with extra functionality.



WEGIS (Western Expedient Geographical Information System)

Logout Help

WEGIS

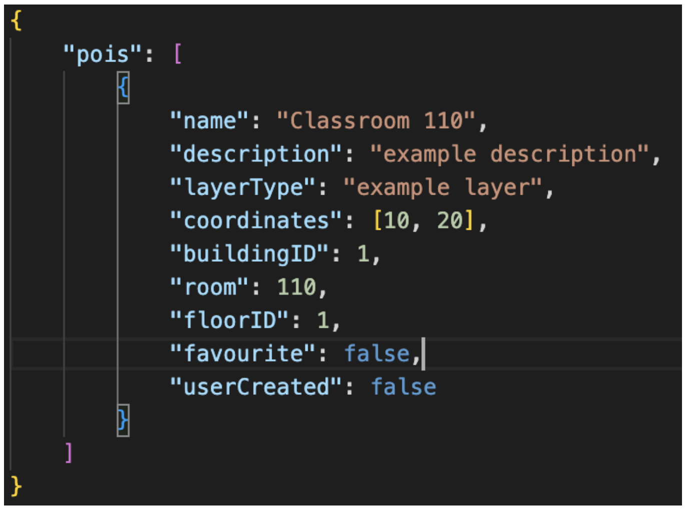
Middlesex College

Insert a new POI

POI Image(if Applicable):	image1.png
POI Room Name:	2212 Classroom
POI Room Number:	010
POI Description:	lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua ...

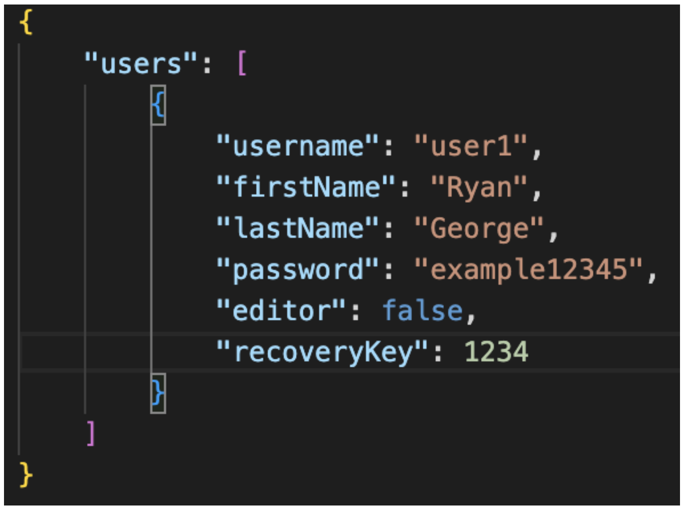
5) File Formats

We will be using JSON files to store our data. We will have one file for the Points of Interest. It will have a key for its name, description, layer, coordinates, building id, room number, floor id, whether it is a favourite or not, and whether it is user created or not.



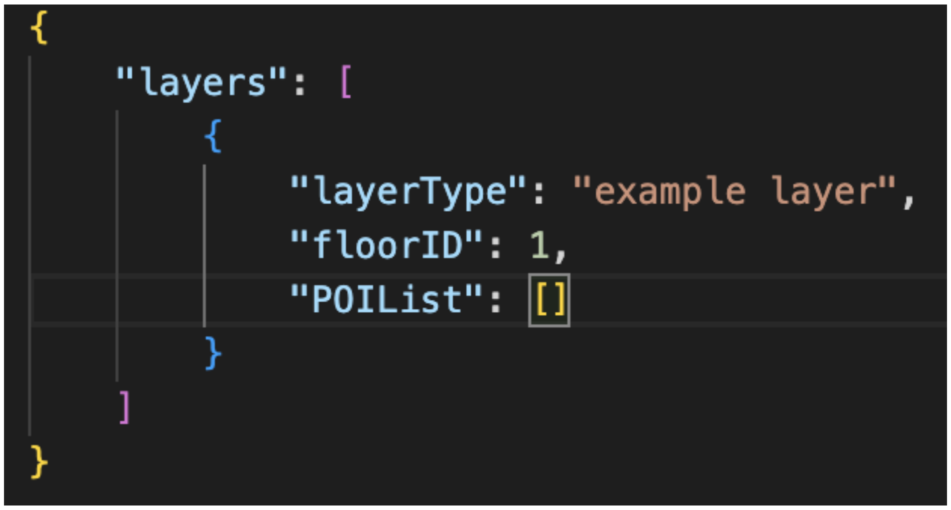
```json
{
    "pois": [
        {
            "name": "Classroom 110",
            "description": "example description",
            "layerType": "example layer",
            "coordinates": [10, 20],
            "buildingID": 1,
            "room": 110,
            "floorID": 1,
            "favourite": false,
            "userCreated": false
        }
    ]
}
```

There will also be a JSON file for user information which stores their username, password, first name, last name, whether or not the user is an editor, and a recovery key.



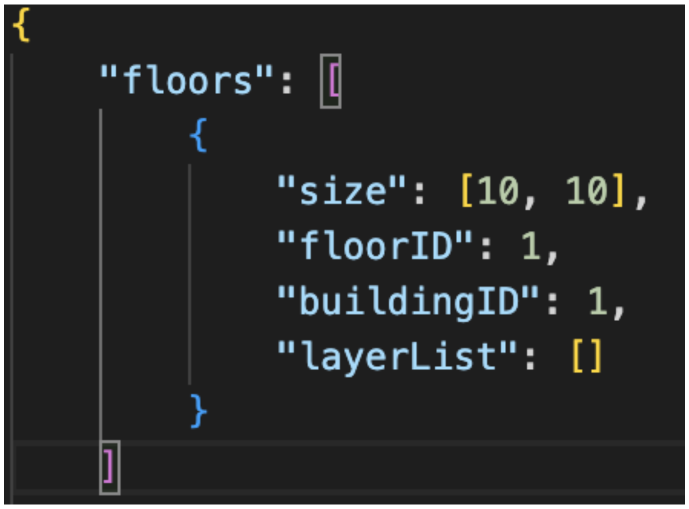
```json
{
    "users": [
        {
            "username": "user1",
            "firstName": "Ryan",
            "lastName": "George",
            "password": "example12345",
            "editor": false,
            "recoveryKey": 1234
        }
    ]
}
```

Another JSON file for the layers, which will store the layer type, floor id, and list of POIs in that layer for each layer.



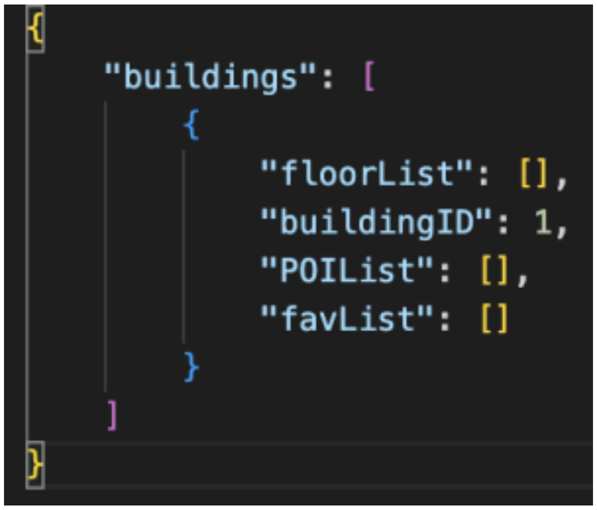
```json
{
    "layers": [
        {
            "layerType": "example layer",
            "floorID": 1,
            "POIList": []
        }
    ]
}
```

Another JSON file for the floors, which will store the floor size, floor id, building id, and a list of the layers in that building.



```json
{
    "floors": [
        {
            "size": [10, 10],
            "floorID": 1,
            "buildingID": 1,
            "layerList": []
        }
    ]
```

Another JSON file for the buildings which will store a list of the floors in that building, the building id, a list of the POIs in that building and a list of the favourites in that building.



```json
{
    "buildings": [
        {
            "floorList": [],
            "buildingID": 1,
            "POIList": [],
            "favList": []
        }
    ]
}
```

We will be using json.simple to read and write JSON files.

For the map files, we will be converting them to image files.

6) Development Environment

NetBeans IDE

We will be using NetBeans IDE to develop our program. This IDE is useful for creating our GUI and is able to automatically generate JavaDoc templates. The Git integration will allow us to easily transfer and update files across workstations.

We will first create a NetBeans Platform application build structure, then use the Swing UI toolkit and "Matisse" GUI Builder to create a window component.

The project will then be stored as a Maven Project with an included pom.xml file.

JavaDoc

We will utilize JavaDoc to generate API documentation in HTML format from Java source code. Our IDE of choice NetBeans automatically generates JavaDoc templates.

JUnit

The Java library JUnit provides a testing framework for testing our code.

7) Patterns

The patterns we will be utilizing in our program are as follows:

- Singleton
- Composition
- Registration
- Partial Map

7.1 Singleton

Pattern Name:	***Singleton***
Problem:	*There are times when it is necessary to only have one instance of a class.*
Motivation:	*Any situation in which it is necessary that there is only one instance of a class.*
Context:	*Within our program there is an object that must only have one instance, which is important for internal consistency. For ease of storage and execution only one* *instance must be created, and a global access point to that instance is utilized.*
Solution:	*Only one instance of a class is created, along with a global access point to reference it.*
Intent:	*Ensure that only one instance of a class is created, and provide a global access point to that object.*
Known Uses:	*Accessing resources in shared mode, logger classes.*

7.2 Composition

Pattern Name:	***Composition***
Problem:	*While programming it is necessary for us to treat Branches and Leaf Nodes the same.*
Also-known-as:	*Composite pattern.*
Motivation:	There are times when a program needs to manipulate a tree data structure and it is necessary to treat both Branches and Leaf Nodes the same.
Context:	*Within our program there is a tree data structure in which Branches and Leaf Nodes must be treated the same.*
Solution:	*Within a tree data structure Branches and Leaf Nodes are treated the same.*
Intent:	*This pattern composes objects into tree structures to represent part-whole hierarchies.*
Implementation:	*Methods need to perform regardless of if a node is a composite or a leaf.*
Known Uses:	*Graphics Drawing Editor*

7.3 Registration

Pattern Name:	***Registration***
Problem:	*Users are required to create a profile to hold their information and personalized POIs.*
Motivation:	*A new user is looking to create an account for WEGIS use.*
Context:	*WEGIS is a personalized application, so it is necessary for users to create a unique profile. A new user is creating a profile.*
Forces:	*Only necessary information should be required, as too many fields may cause frustration on the user's end.*
Solution:	*Offer a registration process which must be findable before the login process. Users will complete forms with their information, some of which* *like passwords must be hidden while typing and secured. When the registration process is complete users will be able to access the Login page.*
Intent:	*By using this pattern users are able to create a unique profile for use within WEGIS.*

Consequences:	<ul><li>User profiles and their relevant information must be stored within the application</li><li>User passwords, access keys must be secured</li><li>Users must remember their personal information to login</li><li>There must be a process to retrieve your account if a user forgets their password</li></ul>
Known Uses:	*Facebook registration.*
Related Patterns:	*Login, Forgot Password*

7.4 Partial Map

Pattern Name:	***Partial Map***
Problem:	*It is necessary for users to utilize a map while the application displays other information.*
Motivation:	*A user must obtain information from the map while navigating the side panel.*
Context:	*A user requires equal information from the map and side panel. A user is able to see their selected POI layers in a list on the side panel while the POIs are displayed on the map. A user looks at the campus map while deciding which building's maps to explore.*
Forces:	*It is essential for users to view a map while using a geographical information system. Users need panel information and map data to refine their searches.*
Solution:	*Display the side panel and map data in equal parts. Map data may include a campus map or building floors depending on what the user has selected, and different POI layers on building floors also determined by user selection. A user should be able to see a list of buildings alongside the positions of those buildings on the campus map.*
Intent:	*With this pattern users are able to explore the maps while accessing side panel utility.*
Consequences:	<ul><li>*This dual focus limits the amount of information displayed by both the map and side panel*</li><li>*Side panel utilities must make sense within the context of the map*</li><li>*Scroll features must be incorporated into the map, which may unexpectedly affect the side panel*</li></ul>
Known Uses:	*LTC live transit information.*
Related Patterns:	*Side Panel, Map*

8) Summary

The design objectives of this program are listed in the introduction, where you can see what kind of information is being presented in this document. The class diagrams page explains the major classes that will be used for implementation. For each class, you will be able to see the attributes with their type and visibility, methods including their parameters, parameter types, return types, and visibility, and associations, generalizations, and multiplicity showing the relationship between each of the classes. There is also a user interface mockup page which shows the general idea for how the program will look and how the user will interact with the program. It is great for getting a good idea of the layout of the interface and the flow between the different interfaces. The file formats page focuses more on implementation, and explains how all of the data will be handled, and how it will be stored. The development environment page explains the tools that will be used to create the program, and the patters page explains any patterns that will be used in the project, either UI related or implementation (UI/UX patterns, object oriented patters, etc).