

Contents

1	Spotlight	3
2	Introduction	3
3	Use Case Diagram	3
4	Use Cases	4
4.1	Use Cases 1	4
4.2	Use Cases 2	5
4.3	Use Cases 3	6
5	Domain Modeling	7
6	Sequence Diagram	7
7	Class Diagram	9
8	Ski Application	9
8.1	Project Configuration	9
8.1.1	GitHub Repository	9
8.1.2	Technologies Used	9
8.1.3	Prerequisites	10
8.1.4	Backend Application	10
8.1.5	Frontend Application	10
8.1.6	Additional Information	10
8.2	Entity Relationship Diagram	11
8.3	Project Deployment	12
9	Backend Implementation	12
9.1	Features	12
9.1.1	Pathfinding with Dijkstra Algorithm:	12
9.1.2	CRUD Operations:	12

9.2	Components	12
9.2.1	Zod Validation:	12
9.2.2	Error Handling Middleware:	13
10	Frontend Implementation	13
10.1	Frontend Layout	13
10.1.1	Welcome Page	13
10.1.2	Feature and Explore	13
10.1.3	Versatile Map	14
10.1.4	Start and Destination Points Selection with Difficulties	15
10.1.5	Available Path Display	16
10.1.6	Preference Selection	17
10.2	Visualize the Selected Path	19
10.3	Versatile Map Zooming function	20
10.4	Error Handling	21
10.5	Auto-Filling	22
10.6	Additional Functions	24
10.6.1	Disabled Switch button	24
10.6.2	Mutual Exclusion Between Switch Buttons	24
10.6.3	Scroll to the Map	25
	References	25

1 Spotlight

(1) Project Deployment:

We have deployed our project to an online server. Please check here [Ski Application](https://ski-resort-map.vercel.app/) or go to the URL: <https://ski-resort-map.vercel.app/>.

- Please make sure you have internet connected.
- Since we are using free tier deployment services, **please note that the backend takes approximately 50 seconds to run from a cold state**. Thank you very much for being so understanding!

(2) Versatile Map:

We have created a versatile map using the ski area image provided by Prof. Adrian Fiech. Our map can be zoomed in and out, and the coordinates of each node and path remain in the same relative position.

(3) Error Handling:

We have added error handling and data validation for both frontend and backend applications.

(4) Real Scene Restoration:

We have restored the positions of the ski slopes and the coordinates of nodes in the original image to a great extent. Our map contains 24 nodes (ends of slopes or lift stations) and 67 edges (slopes or lifts).

2 Introduction

In this project, we design a ski web application for skiers. We focus on the requirement elicitation, system analysis, system design, and implementation from the following aspects. First, we perform requirement elicitation and draw a use case diagram. Then, we provide further details for three main use cases. We also analyze the alternative scenario for the "Find Ski Path with Difficulty" use case. Thirdly, we define conceptual classes for the application and relationships among them. Fourth, we provide sequence diagrams for the "find ski path with difficulty" function and "choose ski path with priority" function. Fifth, we use a class diagram to demonstrate our implementation of the application. Finally, we illustrate our application functionalities and implementation details. Meanwhile, we also provide the guidance to set up our project. We illustrate each aspect in detail in the following chapters.

3 Use Case Diagram

We provide 14 use cases for our ski application. The actor can be a "skier", a "restaurant manager", or a "ski area manager". The use case diagram is shown in Fig. 1. We also submit an individual use case diagram (a JPG file) for the convenience of the examiners. The three main use cases are: "Find Ski Path with Difficulty", "Find Ski Lift", and "Find Restaurant".

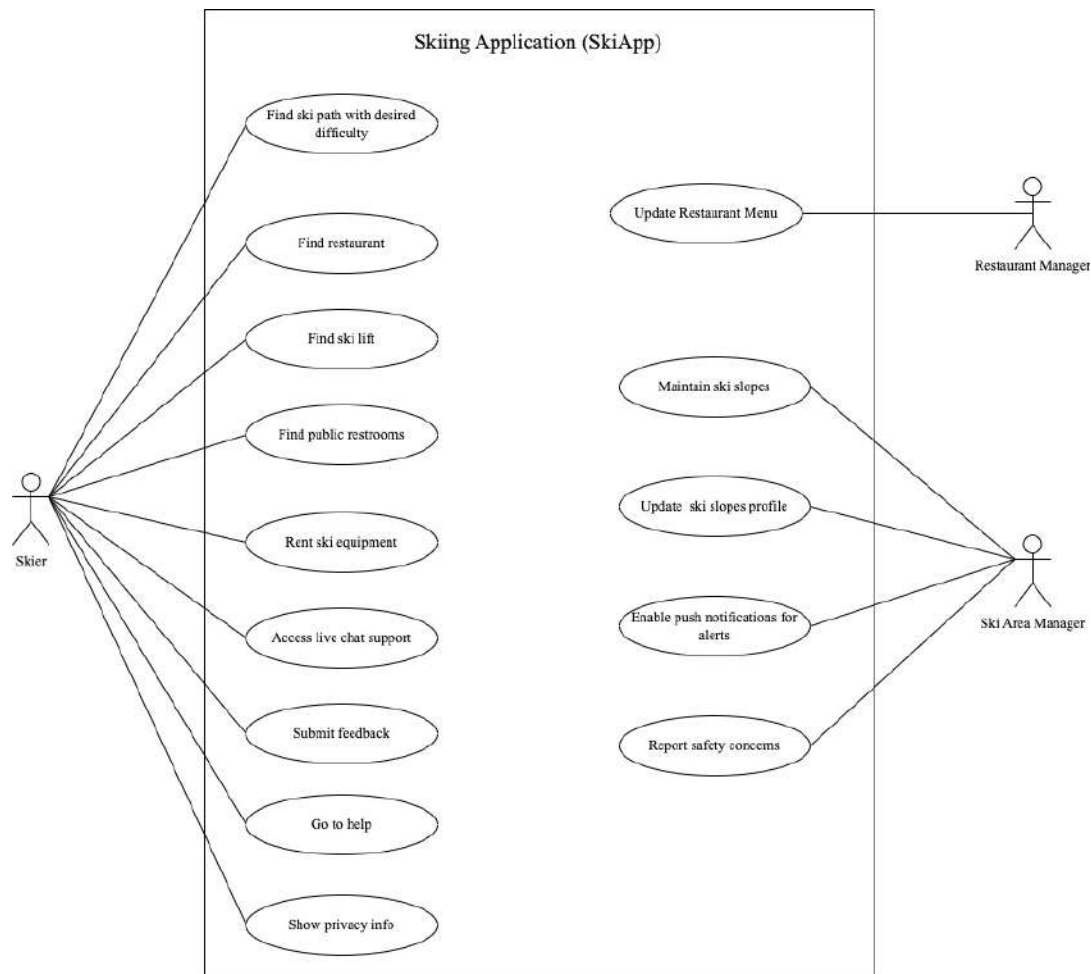


Figure 1: Use case diagram.

4 Use Cases

4.1 Use Cases 1

- Use Case Name: Find Ski Path with Desired Difficulty
- Primary Actor: Skier
- Secondary Actor: Ski Application
- Entry Condition: The skier has been authenticated by the ski application and is on the main page.
- Flow of Events:
 1. Skier selects the option "Calculate Route".
 2. Ski Application provides Skier with an overview of the ski area.
 3. Ski Application requests Skier to provide a start location.
 4. Skier selects the start location.
 - 4a. Skier requests Ski Application to determine Skier's position.

- 4b. Ski Application calculates Skier's position and selects it as the start location.
 - * 4b.1 Ski Application is unable to determine Skier's position.
 - * 4b.2 Ski Application informs Skier about failure to determine the current location.
 - * Use Case resumes at step 3.
 - 5. Ski Application requests Skier to provide end location.
 - 6. Skier selects end location.
 - 7. Ski Application requests the Skier to select a preferred difficulty level of ski slopes (e.g., easy, intermediate, advanced).
 - 8. Skier enters slope restrictions.
 - 8a. Skier requests Ski Application to use Skier's stored profile.
 - 8b. Ski Application retrieves Skier's profile and uses it to determine an acceptable difficulty level.
 - * 8b.1 No profile is available.
 - * 8b.2 Ski Application informs Skier about missing profile.
 - * Use Case resumes at step 7.
 - 9. Ski Application calculates possible routes to the requested destination (based on restrictions) and informs Skier about the options.
 - 9a. If there is no path available with the specified restrictions, Ski Application notifies the skier.
 - 9b. Ski Application retrieves alternative routes or the closest available options.
 - 10. Ski Application requests Skier to select a route based on indicated criteria (fastest, longest, easiest, and minimum lift usage).
 - 11. Skier selects a preferred path from the list.
 - 12. Ski Application displays the desired route (based on the selected criteria) and provides detailed navigation instructions.
- Exit Condition: The selected ski path is highlighted on the map, and relevant details, including difficulty levels of each segment, are displayed.
 - Special Requirement: The mobile application should have large display elements given that the weather would impact the ability of the skiers to use their phones.

4.2 Use Cases 2

- Use Case Name: Find Ski Lift
- Primary Actor: Skier
- Secondary Actor: Ski Application
- Entry Condition: The skier has been authenticated by the ski application and is on the main page.

- Flow of Events:
 1. The skier selects the destination type as the ski lift.
 2. The ski application retrieves information of a list of ski lifts.
 3. The ski application displays a list of ski lifts.
 4. The skier selects a preferred ski lift.
 5. The ski application requests for the skier's current location.
 6. The skier allows the ski application to have her/his current location.
 7. The ski application validates the skier's current location.
 8. The ski application calculates the nearest ski lift based on the skier's current location.
 9. The ski application retrieves the available paths to ski lifts.
 10. The ski application displays a list of potential paths to the skier's preferred ski lift.
 11. The skier selects a preferred path from the list.
- Exit Condition: The selected ski lift path is highlighted on the map, and relevant details, including estimated time of arrival and lift length.
- Special Requirement: The mobile application should have large display elements given that the weather would impact the ability of the skiers to use their phones.

4.3 Use Cases 3

- Use Case Name: Find Restaurant
- Primary Actor: Skier
- Secondary Actor: Ski Application
- Entry Condition: The skier has been authenticated by the ski application and is on the main page.
- Flow of Events:
 1. The skier selects the destination type as the restaurant.
 2. The ski application retrieves information of a list of restaurants.
 3. The ski application displays a list of restaurants.
 4. The skier selects a preferred restaurant.
 5. The ski application retrieves the profile of the selected restaurant.
 6. The ski application displays the profile of the selected restaurant.
 7. The ski application requests for the skier's current location.
 8. The skier allows the ski application to have her/his current location.
 9. The ski application validates the skier's current location.
 10. The ski application retrieves the available paths to the restaurant based on the skier's current location.

11. The ski application displays a list of potential paths to the skier's preferred restaurant.
 12. The skier selects a preferred path from the list.
- Exit Condition: The selected restaurant path is highlighted on the map, and relevant details, including menus and prices.
 - Special Requirement: The mobile application should have large display elements given that the weather would impact the ability of the skiers to use their phones.

5 Domain Modeling

We design 10 conceptual classes for our ski application. The domain model is shown in Fig. 2. We also submit an individual domain model (a JPG file) for the convenience of the examiners.

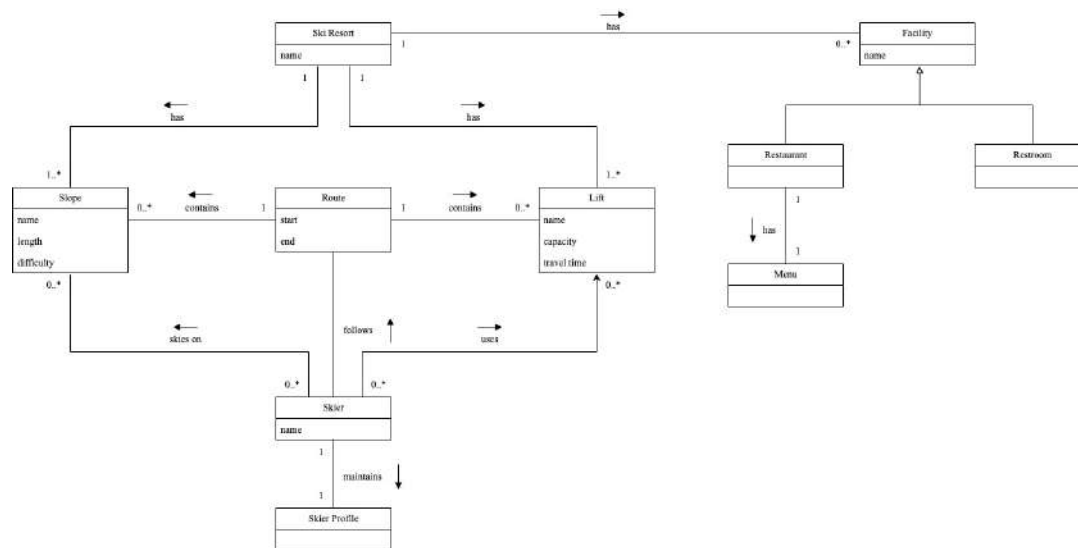


Figure 2: Domain model.

6 Sequence Diagram

In this chapter, we provide sequence diagrams for the "find ski path with difficulty" function and "choose ski path with priority" function. The sequence diagram for the "find ski path with difficulty" function is shown in Fig. 3. We also submit an individual sequence diagram (a JPG file) for the convenience of the examiners.

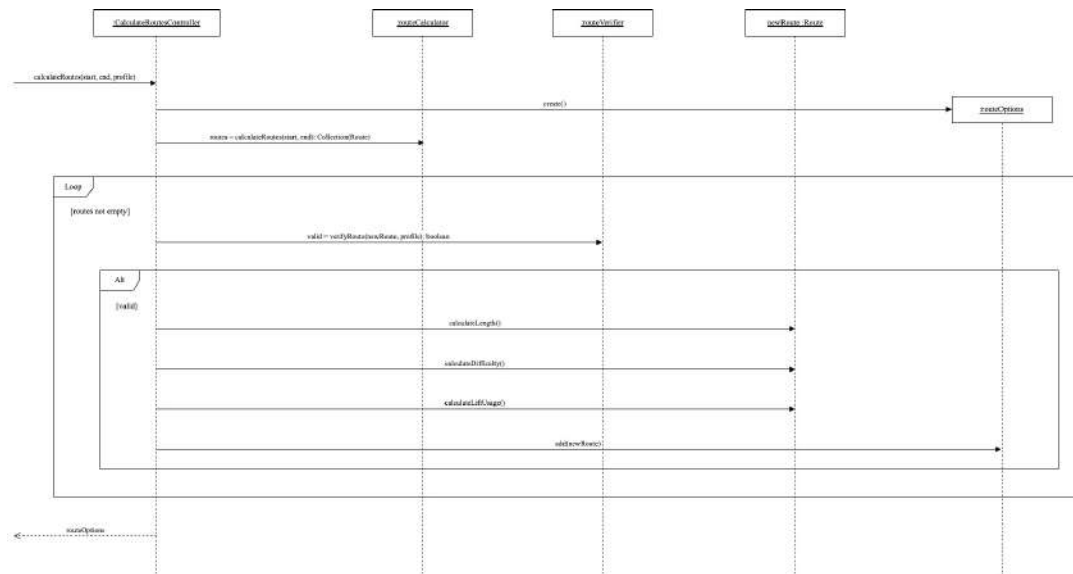


Figure 3: Sequence diagram for the "find ski path with difficulty" function.

The sequence diagram for the "choose ski path with priority" function is shown in Fig. 4. We also submit an individual sequence diagram (a JPG file) for the convenience of the examiners.

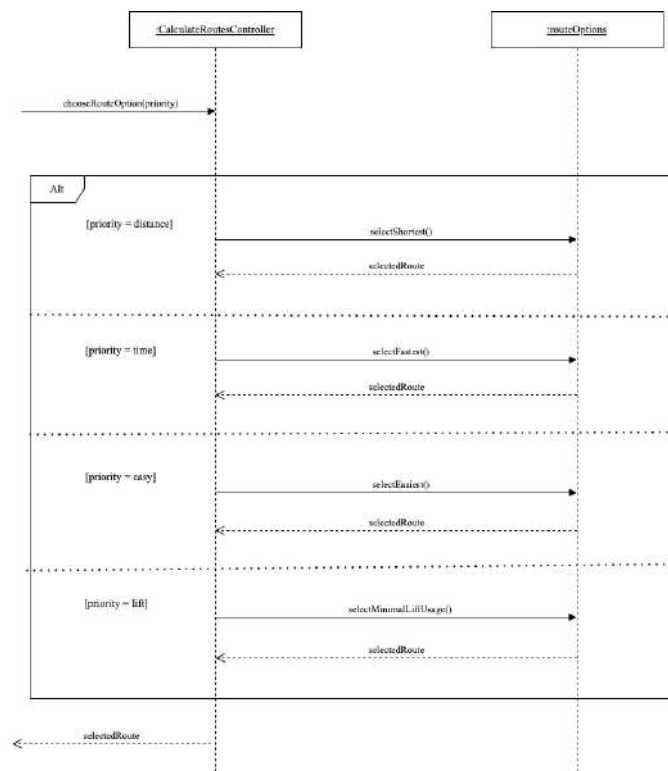


Figure 4: Sequence diagram for the "choose ski path with priority" function.

7 Class Diagram

In this chapter, we provide a class diagram for our implementation. The class diagram is shown in Fig. 5. We also submit an individual class diagram (a JPG file) for the convenience of the examiners.

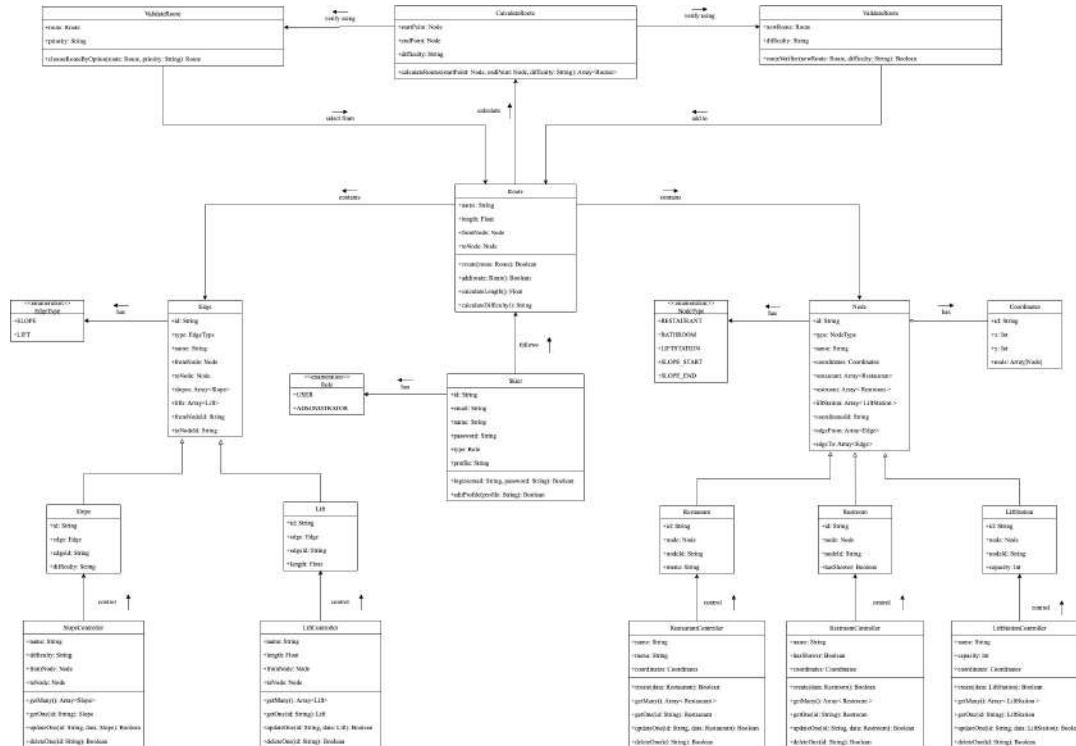


Figure 5: Class diagram.

8 Ski Application

In this chapter, we first describe how to set up the backend and frontend applications of our project. Then, we show the entity relationship diagram (ERD) of our database.

8.1 Project Configuration

8.1.1 GitHub Repository

We have uploaded all the project code to a private GitHub repository. Please check here [GitHub repository](https://github.com/coco-kit/COMP-6905) or go to the URL: <https://github.com/coco-kit/COMP-6905>.

8.1.2 Technologies Used

- TypeScript: Utilized for better limiting and type validation.
- Prisma: Used as an ORM with MongoDB as the database, deployed on Atlas.
- Environment Variables: Environment variables are not included, but you can reference the `.example.env` file for setting them up.
- Zod: Employed for schema validation with static type inference.

- Faker: Used for generating dummy test data.

8.1.3 Prerequisites

Make sure you have the following installed on your machine:

- Node.js
- Yarn

8.1.4 Backend Application

- (1) Navigate to the Backend directory:

```
cd Backend
```

- (2) Install the required packages:

```
yarn
```

- (3) Run the development server:

```
yarn dev
```

- (4) Prisma types generation:

If changes are made to the Prisma schema, generate the corresponding TypeScript types with:

```
yarn prisma:generate
```

- (5) This will start the backend server.

8.1.5 Frontend Application

- (1) Open a new terminal window and navigate to the Frontend directory:

```
cd Frontend
```

- (2) Install frontend dependencies:

```
yarn
```

- (3) Run the frontend development server:

```
yarn dev
```

- (4) This will start the frontend server.

8.1.6 Additional Information

- This project uses Yarn as the package manager.
- Make sure to configure any environment variables if necessary.
- You can use the provided Postman collection *MUN 6905.postman_collection.json* for testing the API.

8.2 Entity Relationship Diagram

The entity relationship diagram of our database is shown in Fig. 6. We also submit an individual entity relationship diagram (a PNG file) for the convenience of the examiners.

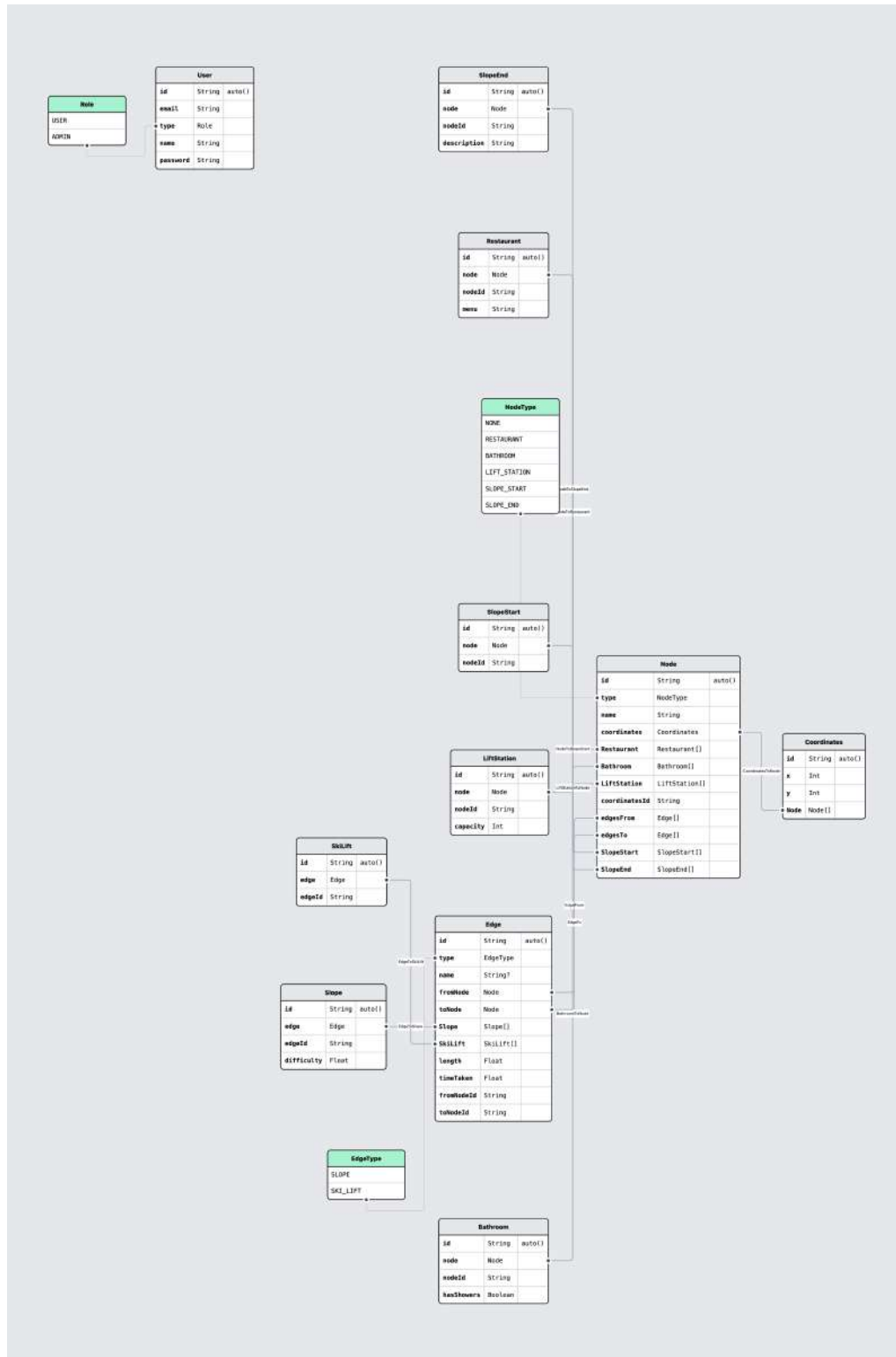


Figure 6: Entity relationship diagram.

8.3 Project Deployment

We have deployed our project to an online server. Please check here [Ski Application](#) or go to the URL: <https://ski-resort-map.vercel.app/>. Since we are using free tier deployment services, **please note that the backend takes approximately 50 seconds to run from a cold state**. Thank you very much for your patience!

9 Backend Implementation

This backend is built using Node.js, and TypeScript, and leverages Prisma ORM for database access, Zod for data validation, and custom error handling middleware. Additionally, it employs the Dijkstra algorithm to find optimal paths on a ski resort map, considering various factors such as shortest distance, fastest route, and easiest path. All the computations are performed in the backend server. The frontend is only for display and user interaction.

9.1 Features

9.1.1 Pathfinding with Dijkstra Algorithm:

The backend utilizes the Dijkstra algorithm to calculate three types of optimal paths on a ski resort map:

1. Shortest path (based on minimum accumulated "length" score)
2. Fastest path (based on minimum accumulated "timeTaken" score)
3. Easiest path (based on minimum accumulated "difficulty" score)

9.1.2 CRUD Operations:

The backend handles all CRUD operations (Create, Read, Update, Delete) for the main entities of the application, which include nodes and edges.

Nodes: All types of nodes have different tables, but they all inherit from the node table and can extend additional information. In previous iterations, nodes exhibited a polymorphic relationship, allowing for various types such as restaurants with menus, etc. However, with changing requirements across iterations 2 to 3, there is no longer a distinction between node types

Edges: There are two types of edges: "SKI LIFT" and "SLOPE". These edges implement another polymorphic relationship. A "SLOPE" has additional information such as the difficulty score.

9.2 Components

9.2.1 Zod Validation:

Zod is employed for schema-based validation, ensuring that the data received through the API is correct and adheres to predefined schemas.

9.2.2 Error Handling Middleware:

Custom middleware has been developed to handle errors effectively. It can identify various types of errors, including Prisma errors, Zod validation errors, or runtime errors, and return descriptive error messages along with the correct status code to the client side. This ensures better error reporting and debugging capabilities.

10 Frontend Implementation

In this section, we first show the overall layout of our frontend. Then, we describe how to use our system. Finally, we discuss our key designs of the system, including versatile maps, error handling, auto-filling, and some additional functions.

10.1 Frontend Layout

Our frontend contains six different modules.

10.1.1 Welcome Page

The welcome page is shown in Fig. 7.

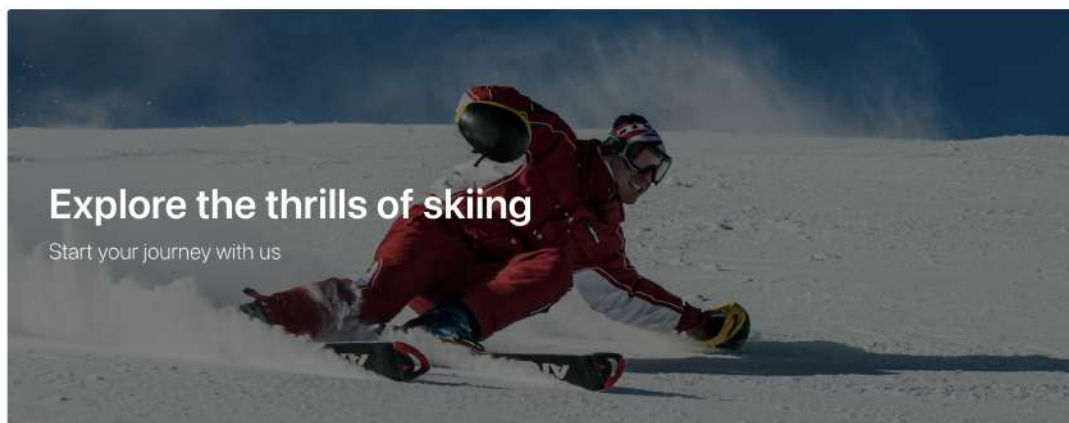


Figure 7: Welcome page.

10.1.2 Feature and Explore

The feature and explore module is shown in Fig. 8. We provide some necessary information for skiers using a horizontal scroll bar.

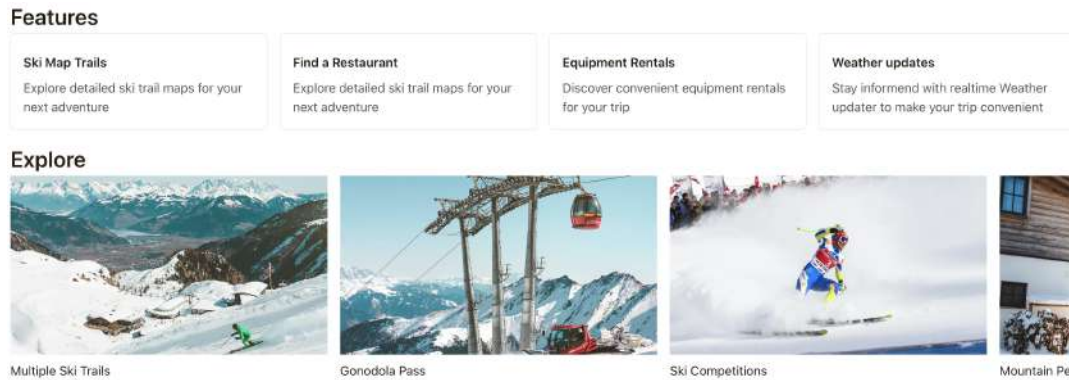


Figure 8: Feature and explore.

10.1.3 Versatile Map

The versatile map module is shown in Fig. 9 with the following functionalities:

- The map shows all the ski slopes, lifts, slope endpoints, and lift stations. The edges are drawn using green lines and the nodes are labeled using blue markers. We have restored the positions of the ski slopes and the coordinates of nodes provided by Prof. Adrian Fiech to a great extent.
- Our map can be zoomed in and out, and the coordinates of each node and path remain in the same relative position. We will thoroughly explain how we made the map in the next section.
- By clicking a marker, we can see the detailed information of that node in a popup.
- Within a message popup, we are able to select the current node as a start point or a destination point for path-finding.

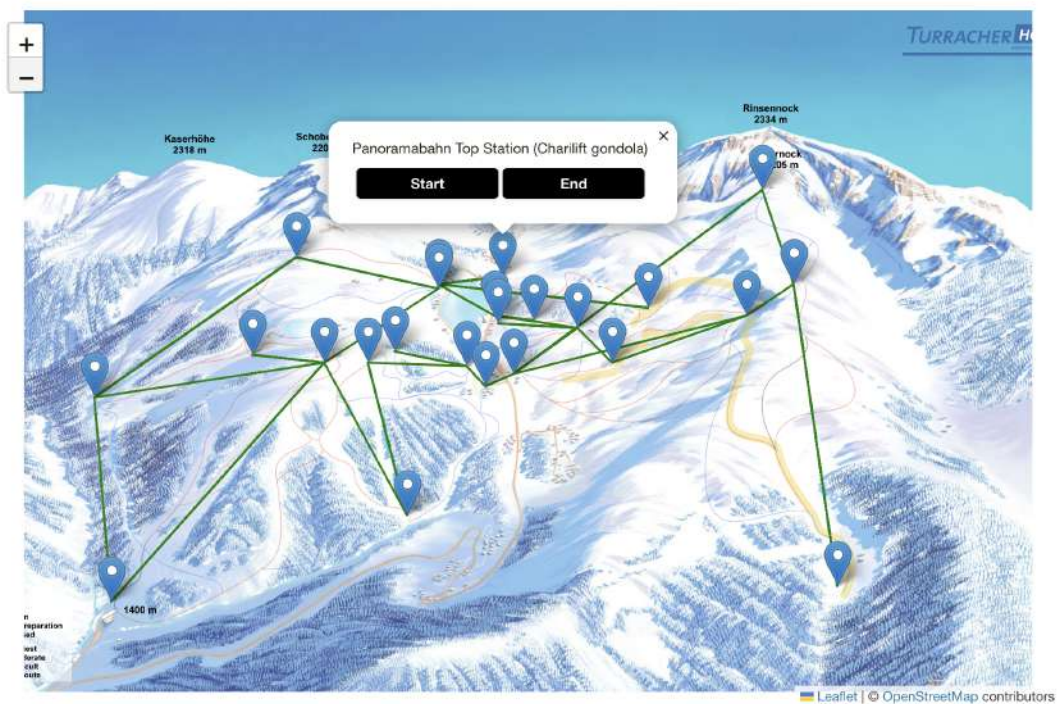


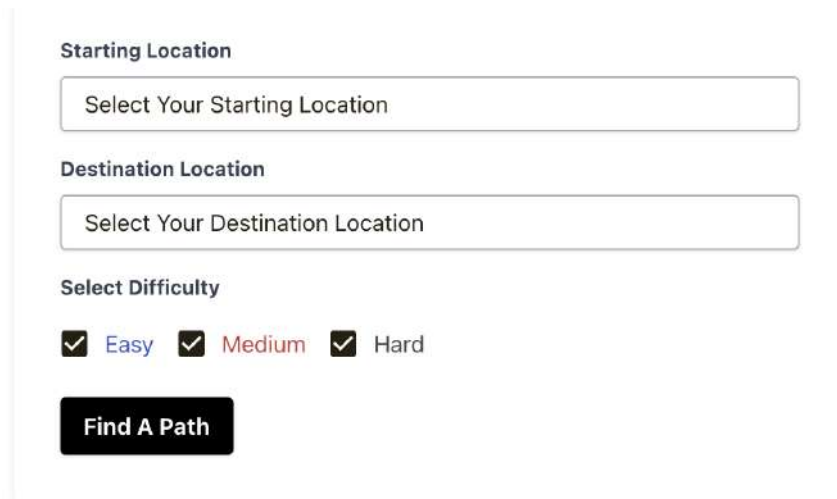
Figure 9: Versatile map.

10.1.4 Start and Destination Points Selection with Difficulties

In this module, skiers are able to select the start and destination points they are interested in. Skiers also can specify the difficulties of slopes that they prefer. The start and destination points selection with difficulties module is shown in Fig. 10.

The two dropdown menus provide all the available nodes to skiers. The difficulties are selected from multi-selection checkboxes. By default, we select all three levels of difficulties. We also use blue, red, and black colors of text to label different difficulty levels. After selecting the points, skiers then click the "Find A Path" button to search for multiple routes between two selected points.

It is worth mentioning that our application provides two input methods for the skiers to select their start and destination points. (1) The first method is to select points directly on the map. (2) The second one is to select points from the dropdown menus. If skiers select locations using the buttons in the marker popups, our application automatically fills the dropdown menus with skiers' selections. We will further discuss this function in the next section.



The form is titled "Starting Location" and "Destination Location". It has two input fields for selecting locations. Below these is a section titled "Select Difficulty" with three checkboxes: "Easy", "Medium", and "Hard". The "Easy" checkbox is checked. Below the checkboxes is a button labeled "Find A Path".

Figure 10: Start and destination points selection with difficulties.

10.1.5 Available Path Display

In this module, we list all the available paths between the start and end points with the given restrictions (difficulty levels). The skiers are able to see each path on the map by clicking the switch button corresponding to each path.

- A switch button has "on" and "off" modes.
- When the switch button is in "on" mode, we show the selected path on the map.
- When the switch button is in "off" mode, we cancel the path selected and show the original map.
- Only one of the switch buttons of all available paths can be in "on" mode because we show one path on the map each time. If one switch button is in "on" mode, and the skiers want to check another path, the skiers only need to click on the switch button corresponding to the new path. Our system will automatically deactivate the previous switch button and activate the new one.

Our available path display module works as follows.



The module has a header with "Returned Route Name" and "Which Path Would You Like to See?". Below the header, it says "No Path Found" and "Show Path on Map (scroll up)". A toggle switch is present, which is currently in the "off" position.

Figure 11: Initial state of the available path display module.

- Initially, the available path display module shows no path, and the switch button is disabled (as shown in Fig. 11).

- After we select the start and end points, the available path display module shows all the available paths with the options to see the paths on the map (as shown in Fig. 13) using switch buttons.

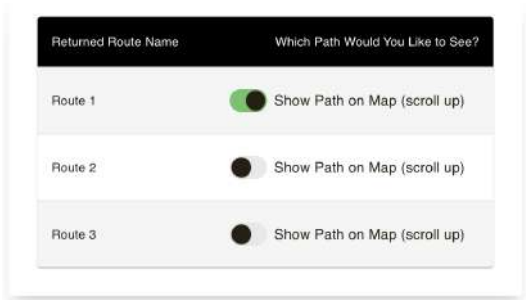


Figure 12: Display a list of available paths.

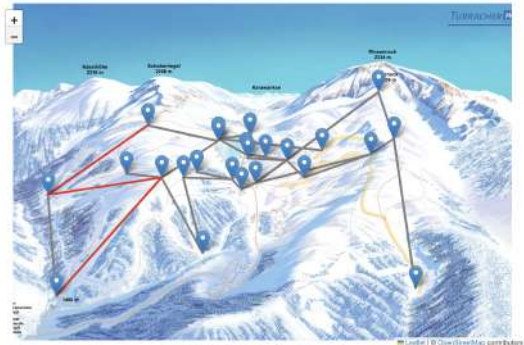


Figure 13: View the selected path.

10.1.6 Preference Selection

The preference selection module is shown in Fig. 14. In this module, skiers can check paths found with different preferences. Our application provides four different types of routes, which are the easiest path, the fastest path, the shortest path, and the minimum lift usage path.



Figure 14: Preference selection.

We use an accordion component to demonstrate different priorities. We can check path information by clicking on each preference (as shown in Fig. 15). All different priorities are described in the same pattern.

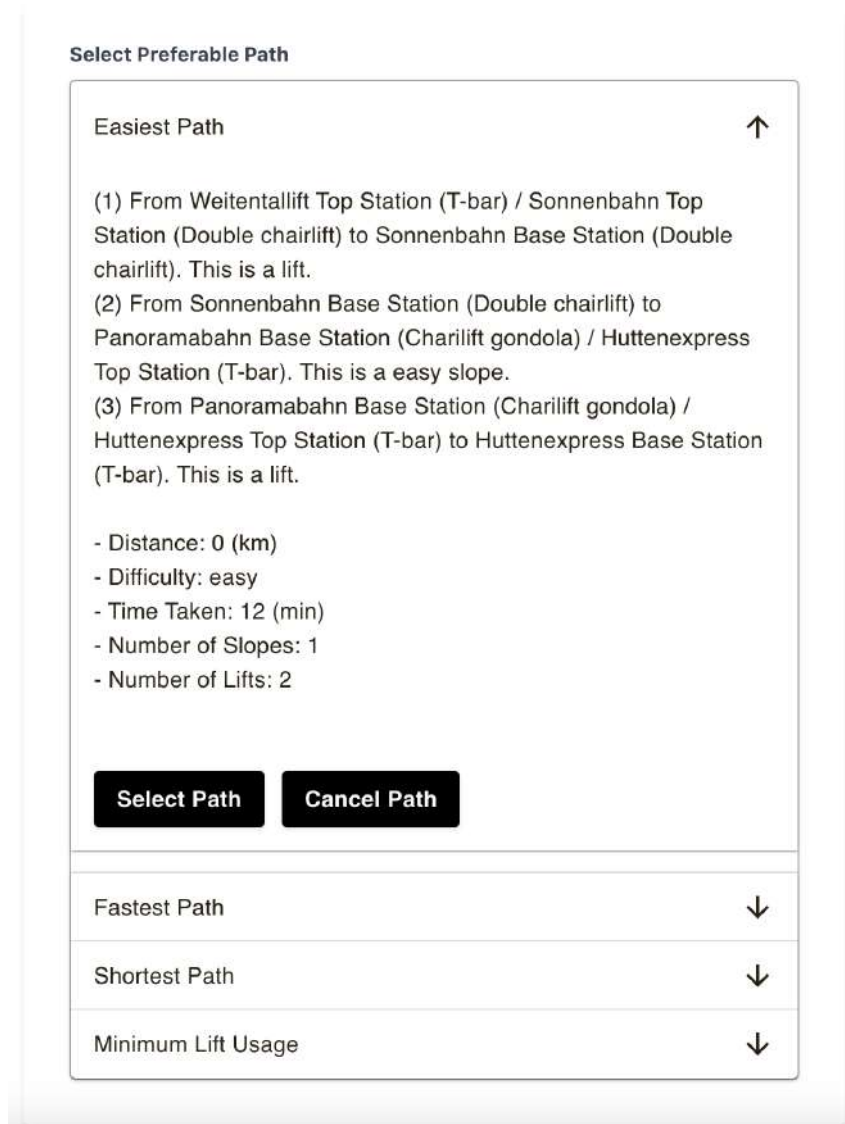


Figure 15: Detailed path information.

1. We illustrate the selected path. We show the start and end points of each edge in the path and give the type of edge (slope or lift). If it is a slope, we also provide the difficulty level in the description.
2. The total distance of the selected path.
3. The general difficulty of the selected path (if there are multiple slopes). If the average path difficulty is less than or equal to 1.0, then the path is an easy one. If the average difficulty is greater than 1.0 and less than 2.0, then the path is a medium path. Otherwise, it is a difficult path.
4. The total time taken to traverse through the path.
5. The total number of slopes in the selected path.
6. The total number of lifts in the selected path.

10.2 Visualize the Selected Path

After we select a preferred path, we can then click the "Select Path" button in Fig. 15. Our application will then automatically scroll back to the map module and show the selected path on the map. The selected path from the start point to the end point is shown in a thick red line on the map. Other paths are greyed out to highlight the chosen path as shown in Fig. 16.

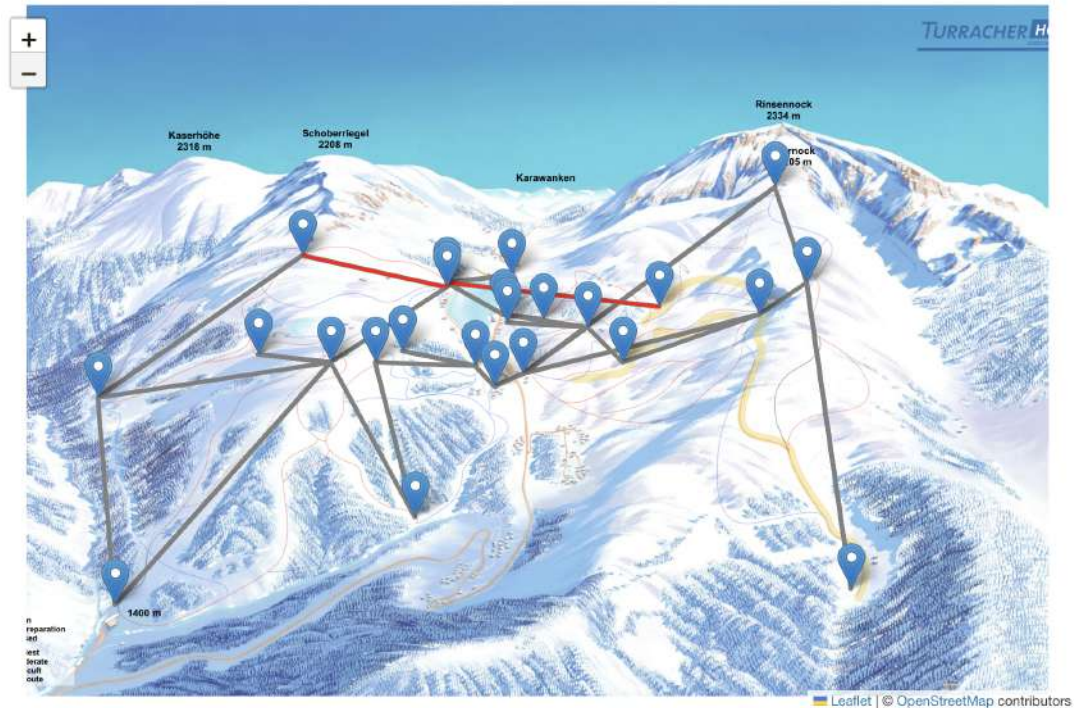


Figure 16: Visualize the selected path on the map.

When skiers want to check on the map or want to select another path, we can click the "Cancel Path" button in Fig. 15. Then, skiers can have the original map again (as shown in Fig. 17).

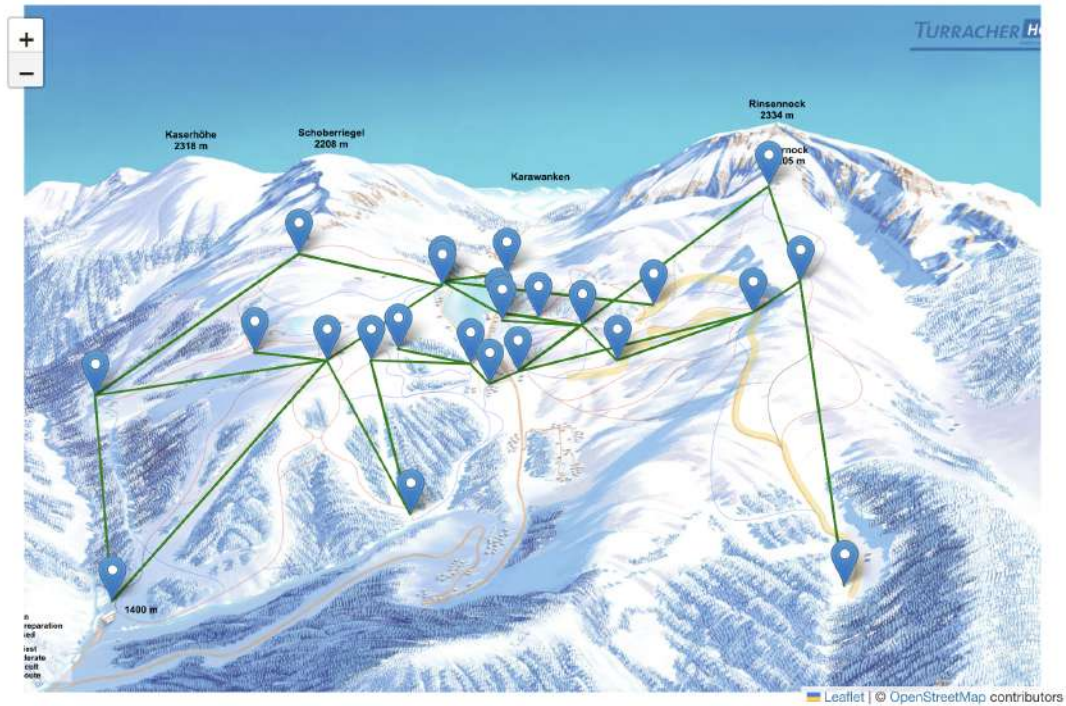


Figure 17: Click the "Cancel Path" button.

10.3 Versatile Map Zooming function

We built our map with Leaflet which is a JavaScript library for creating interactive maps. To create a scalable map with a specific image, we first need to obtain the map image and store it in the project folder so that Leaflet can have access to it. We learned the steps from Sigon's blog [1]. Then, we need to convert the image into tiles with open layers (xyz URL). We learned how to do it from Orblivion's blog [2]. During this process, we also need to perform coordinate system transformation to convert the coordinates of the nodes on the map based on different coordinate reference systems (CRS). The transformation equations we used are shown as follows.

$$x' = \frac{607 \times y}{174} - 607$$

$$y' = 150 + \frac{(1049 - 150) \times x}{255}$$

Where (x, y) are in the local coordinate reference system (the origin locates at the bottom-left corner) and (x', y') are in the Leaflet's coordinate reference system where the origin locates at the upper-left corner. The x-axis (or y-axis) in the local CRS matches the y-axis (or x-axis) in the Leaflet's CRS. The scales of the original coordinates are $[0, 255]$ and $[0, 174]$ for x and y-axes, respectively. The scales of the Leaflet's coordinates are $[0, 607]$ and $[150, 1049]$ for x and y-axes, respectively.

We show the zooming effect at different zooming levels in Fig. 19 - Fig. 21. Our map can be zoomed in and out, and the coordinates of each node and path remain in the same relative position.

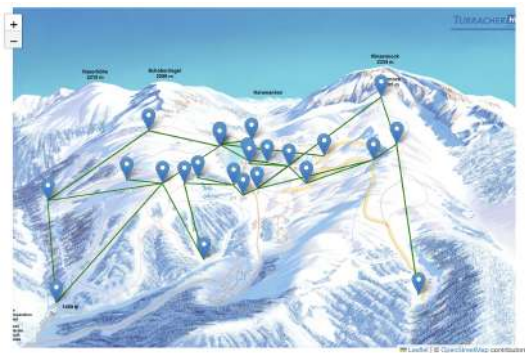


Figure 18: Zooming in level equals 1.

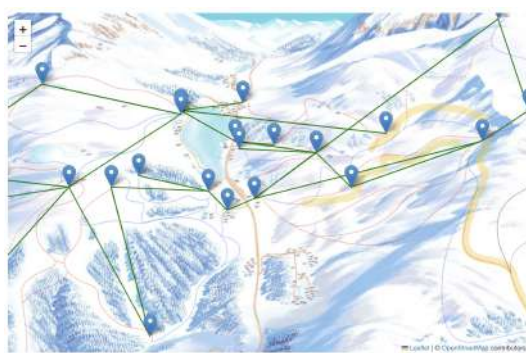


Figure 19: Zooming in level equals 2.

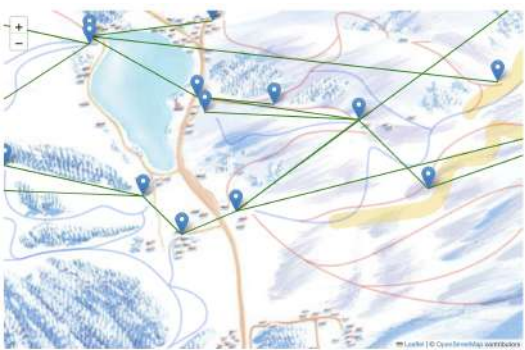


Figure 20: Zooming in level equals 3.

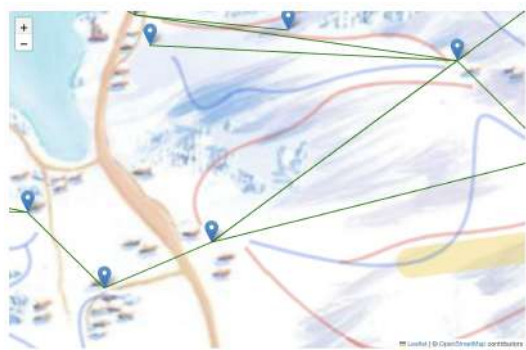


Figure 21: Zooming in level equals 4.

10.4 Error Handling

Another main function that can improve user experience is error handling. In our application, skiers may accidentally search for a path without providing full information on the start and end points. In such cases, our application can alert the skiers about which information they should provide to proceed with the path-finding, respectively. The alert messages under different circumstances are shown in Fig. 23 - Fig. 26.

Please select the start node!

Please select the end node!

Starting Location

Select Your Starting Location

Destination Location

Select Your Destination Location

Select Difficulty

☒ Easy ☒ Medium ☒ Hard

Find A Path

Figure 22: Missing start and end points.

Please select the start node!

Starting Location

Select Your Starting Location

Destination Location

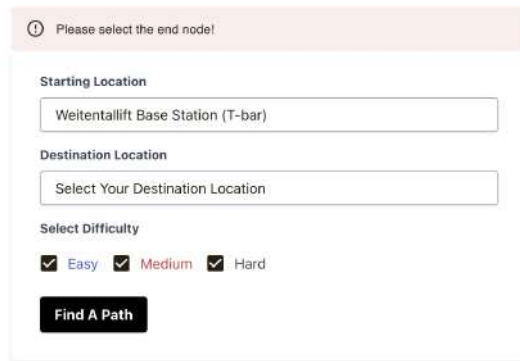
Sonnenbahn Base Station (Double chairlift)

Select Difficulty

☒ Easy ☒ Medium ☒ Hard

Find A Path

Figure 23: Missing the start point.



Please select the end node!

Starting Location

Weitentallift Base Station (T-bar)

Destination Location

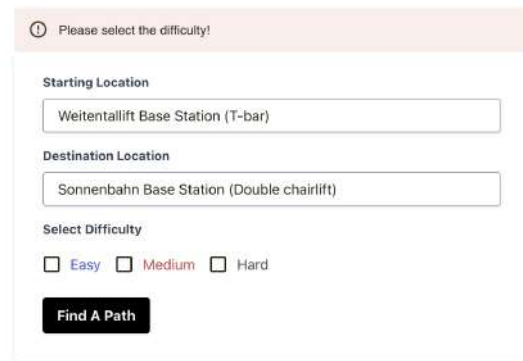
Select Your Destination Location

Select Difficulty

☒ Easy ☒ Medium ☒ Hard

Find A Path

Figure 24: Missing the end point.



Please select the difficulty!

Starting Location

Weitentallift Base Station (T-bar)

Destination Location

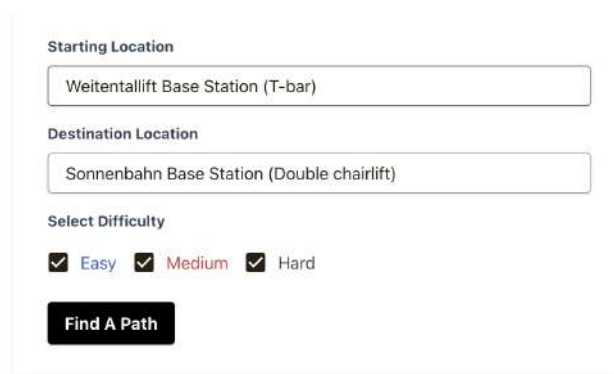
Sonnenbahn Base Station (Double chairlift)

Select Difficulty

☐ Easy ☐ Medium ☐ Hard

Find A Path

Figure 25: Missing difficulties.



Starting Location

Weitentallift Base Station (T-bar)

Destination Location

Sonnenbahn Base Station (Double chairlift)

Select Difficulty

☒ Easy ☒ Medium ☒ Hard

Find A Path

Figure 26: Correct user input.

10.5 Auto-Filling

As we mentioned before, our application provides two input methods for the skiers to select their start and destination points. If skiers select locations using the buttons in the marker popups directly on the map, our application automatically fills the dropdown menus with skiers' selections. Since we have 24 nodes in the database, purely searching locations from long lists of dropdown menus could be difficult when the skiers are skiing. This function allows skiers to select the points directly from the map and are still able to revise their selections by checking on the names of points they selected in the dropdown menus. The auto-filling function is shown in Fig. 27 and Fig. 28.

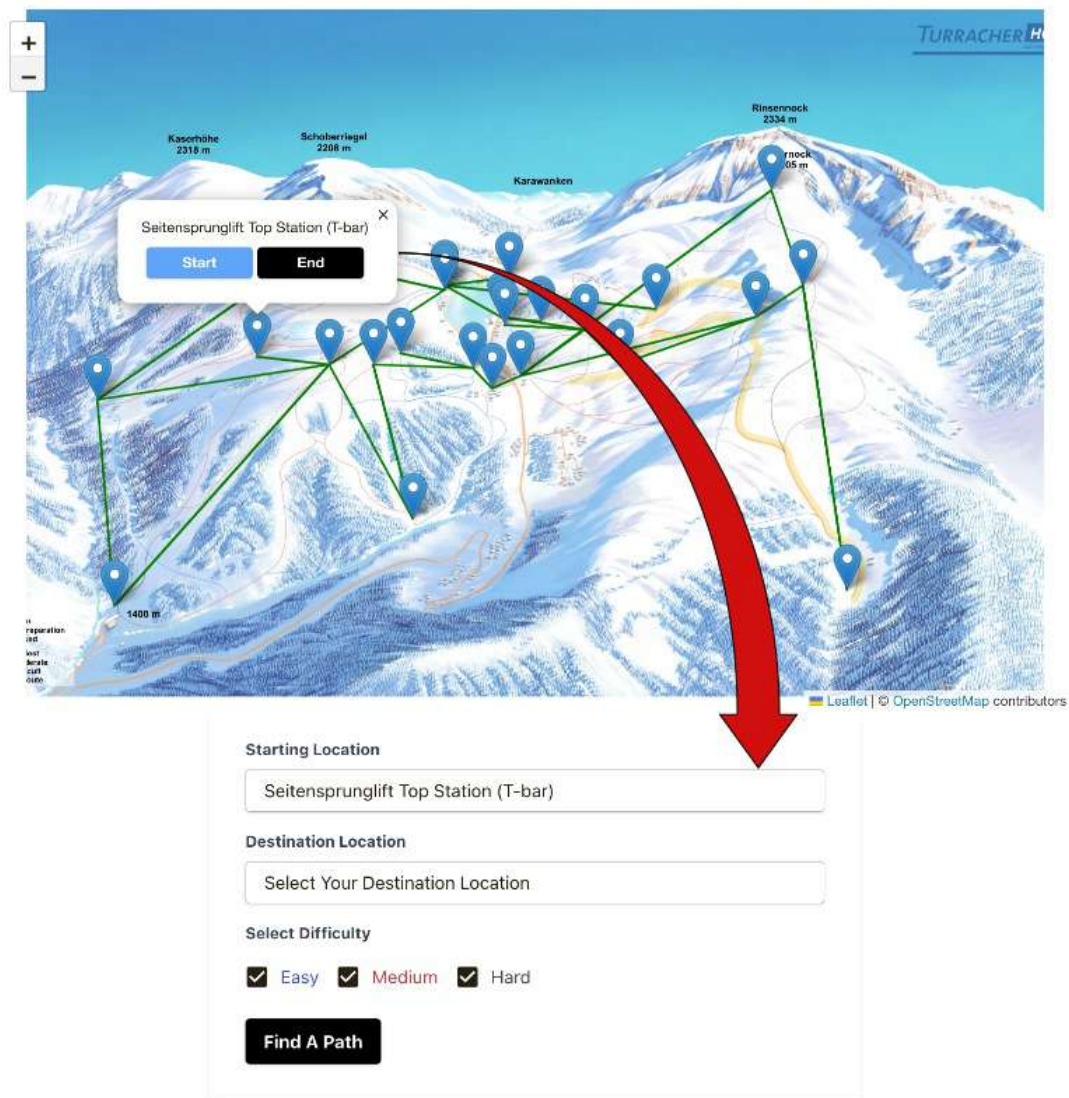


Figure 27: Auto-filling for the start point.

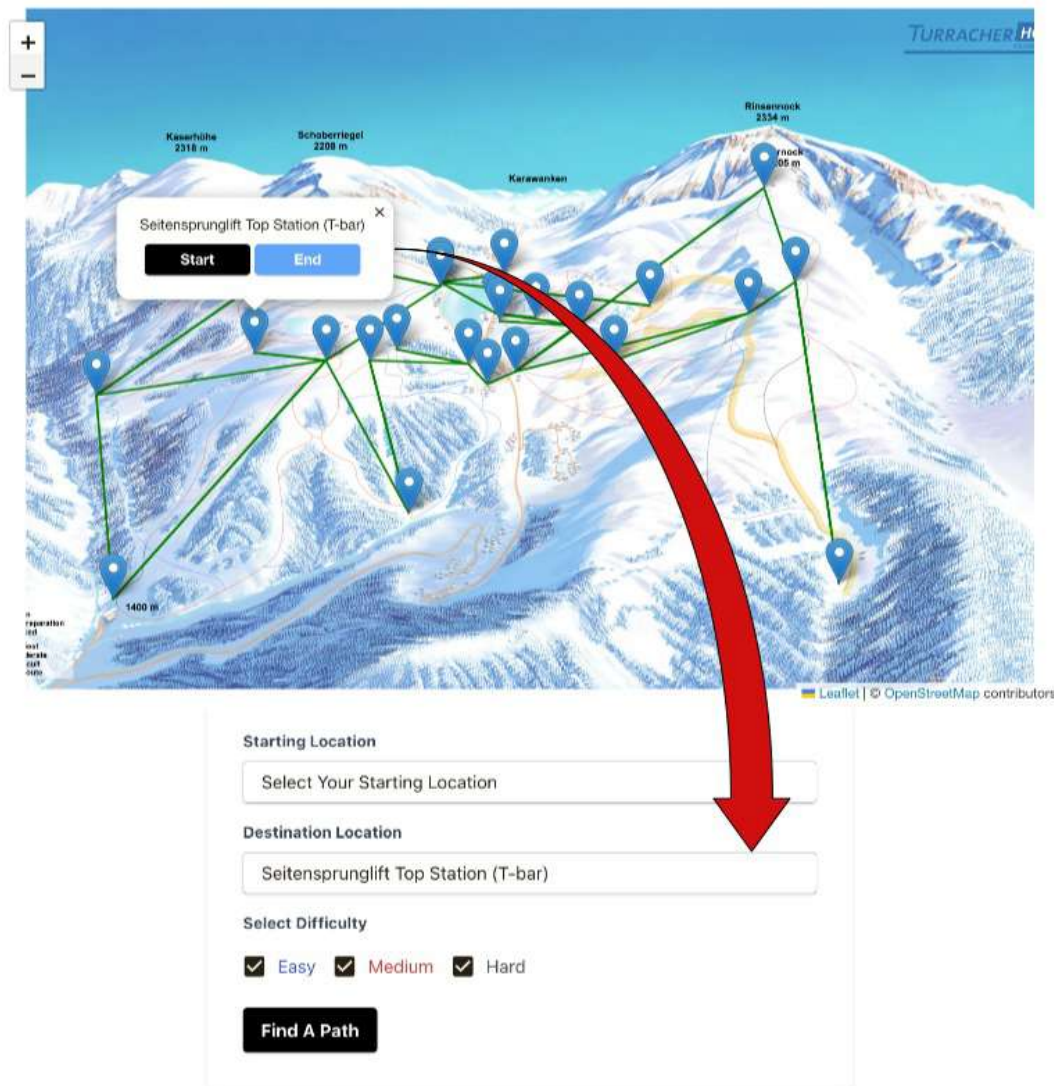


Figure 28: Auto-filling for the end point.

10.6 Additional Functions

In this section, we discuss three additional functions we designed to improve user experience.

10.6.1 Disabled Switch button

Initially, the switch button in the available path display module is disabled because there are no paths. On the other hand, the switch button is also disabled if there is no path between selected start and end points. This function gives skiers an intuitive understanding of the routes they decide to travel.

10.6.2 Mutual Exclusion Between Switch Buttons

Only one of the switch buttons of all available paths can be in "on" mode because we show one path on the map each time. If one switch button is in "on" mode, and the skiers want

to check another path, the skiers only need to click on the switch button corresponding to the new path. Our system will automatically deactivate the previous switch button and activate the new one.

10.6.3 Scroll to the Map

When the paths and detailed information are shown in the accordion component, it could take some space. Skiers have to manually scroll up back to the map to see the visualization of the selected path on the map. Therefore, we add a "scroll to the map" function so that when skiers click the "Select Path" or "Cancel Path" buttons, our application will automatically scroll back to the versatile map smoothly and display the results.

References

- [1] Sigon. (2020, May 19). Convert an image to a web map. Sigon ./. <https://sigon.gitlab.io/post/2020-05-19-image-to-leaflet-maps/>
- [2] Orblivion. (62022, October 14). Convert image into tiles with xyz url (open layers). Stack Overflow. <https://stackoverflow.com/questions/48272430/convert-image-into-tiles-with-xyz-url-open-layers>