
Final Report

for

NOTL Museum Chat Bot

COSC 4P02

Prepared by Maheen Samad

Fahad Arain

David Bailey

Robert Morabito

Dana Dobrosavljevic

Jordan Bharati

Sarah Howcroft

Table of Contents

Table of Contents	ii
1. Updated Requirements Document	1
1.1 Original Requirements.....	1
1.2 Updated Requirements	1
2. Member Contributions.....	3
2.1 Member Contributions.....	3
2.2 GitHub Logs & Commits	4
3. User Manual.....	9
3.1 Home Page.....	9
3.2 Admin Page	10
3.3 ChatBot Page	11
3.4 Directions Page.....	13
4. Technical Manual.....	14
4.1 Database.....	14
4.2 GPT.....	15
3.1 Directions.....	17
3.2 Front End	19
5. Scrum Process	22
6. Testing.....	24
5.1 Autmated System Testing.....	24
5.2 User Testing.....	26
5.3 Improvements	30

The website is currently deployed. You can access it at:

<https://museum-mate.vercel.app/>

1. Updated Requirements Document

This section highlights the requirements set out by the customer in Sprint 1, and how they changed throughout the development process and why these changes were necessary to implement in order to make the project successful. This section also documents how these changes were managed by developers, scrum master and product owner.

1.1 Original Requirements

The following are a list of requirements set out by the product owner in preliminary stages of the project. This was prior to any development.

- have a homepage which allows users to select the way they want to navigate the website. This will be a button which says either admin or visitor. The admin is a member of the museum workplace or representative. A visitor is a researcher or potential visitor of the museum who is interested in using the chatbot.

- If user selects visitor, then it redirects the user to the Chabot page. The requirement here is to create a chatbot which addresses questions related to 4 different categories, users are able to select which category they have inquiries for:

- Events: this category allows users to input questions related to upcoming events at the museums such as new exhibits, activities and upcoming significant events
- Research: this category allows users to input questions related to research such as questions about history and artifacts
- Directions: this category allows users to ask questions related to navigation within the museum such as getting from one exhibit to another.
- Museum Info: this category allows users to input questions related to museum information such as hours, address, museum opening date, etc.

-If user selects admin, then it redirects the user to an admin login page. The requirement here is to create a page for admin. This will allow admin members to login with correct authentication. Once their login is authenticated the user is able to add information to the database which can be an artifact for an exhibit. The purpose of this section is to allow the museum to add artifacts that are added to an exhibit in real life, and have this change reflect onto the application as well. Once a new artifact is added, users are able to ask questions about it in the chatbot for its respected category.

- all of these requirements must be interfaced with a modern, user friendly and visually pleasing front end

1.2 New Requirements

As development began, requirements began to change from the product owner due to change in circumstances, development progress and client needs. This occurred between sprints 1 to sprint 5 where the respected requirement changed are listed below. It is important to note that not all

original requirements necessarily changed, there were requirements that were added in order to accommodate for changing needs.

Sprint	Requirements Added / Modified
1	<ul style="list-style-type: none"> Requirements did not change during this sprint
2	<ul style="list-style-type: none"> Changed requirement: eliminate need for “categories” and make a chatbot which responds to any type of question regardless of which category it fits. This requirement was changed in order to create avoid user confusion and allow them to input any type of museum related question without the need to select a category Changed fine tuning development to a prompt-based method for ease in development and low cost of resources while also being just as efficient as fine tuning
3	<ul style="list-style-type: none"> Changed path finding algorithm to A star algorithm since It is more efficient than other path finding algorithms Changed museum data to real data from museum database in order to reflect real life information rather than placeholder information for commercial use Added text to speech and speech to text features Added translation features to translate all information in the chatbot to either English, French, German, or Spanish
4	<ul style="list-style-type: none"> Changed all front-end visualizations of chatbot in order to present a complete and visually pleasing interface and the interface in previous sprints was for testing purposes. Removed visual feature of directional component and kept written instructions for simplicity, time constraints and resource constraints
5	<ul style="list-style-type: none"> Added requirement to improve response time from testing phase Added requirement to incorporate icons, rather than word buttons in order to become more user friendly Added requirement to improve image quality and sizes in chatbot responses

2. Member Contributions

2.1 Member Contributions

The following section outlines each team member and their major contributions to make this project successful. Each team member was given significant sections of the application to be responsible for:

Fahad Arain:

- Database
- Admin Authentication
- Web scraper for database and testing
- Database-Chat-Bot Integration
- Automated Testing
- Completing Sarah's tasks (see below)

David Bailey:

- Directional Component,
- Directional-Chatbot Integration

Robert Morabito:

- GPT
- Chat-bot
- Database-Chat-bot Integration
- Progress Report 2
-

Dana Dobrosavljevic:

- Final Front-end
- Early Database work (Sprint 1 and 2)

Jordan Bharati:

- Front-end and Interfacing
- Temporary front end
- Testing improvements
- Mobile Version front end

Sarah Howcroft:

- no tasks accomplished, was given testing and text to speech features
- All assigned tasks were then completed by Fahad Arain

Maheen Samad:

- Scrum Master Tasks (meetings, sprint retrospectives, sprint reviews)
- Product Owner Tasks (user stories, product backlog, etc)
- All documentation work done (Proposal Brief, Requirements document, Progress Report 1, Progress Report 2, Final Report, Final Presentation ppt)

It is important to note that documents were completed by Maheen, however, uploaded to GitHub by another member therefore commits are not showing for her, however all documentation was completed under her tasks, by her, but uploaded by another member.

2.2 GitHub Commits & Logs

The following section outlines all development members GitHub commits and Logs through screenshots from GitHub.

Maheen Samad completed all Scrum Master duties, and wrote all documentation such as proposal brief, requirements documents, progress reports, and final report. The documents were uploaded by another member. The following images below are the GitHub logs and commits of all the developers during the development of the Museum Mate Chat Bot Application.



Commits on Jan 30, 2023

Added frontend UI to the chatbot
Jordan3617 committed on Jan 30

Commits on Jan 23, 2023

added env to gitignore
Fahad0819 committed on Jan 23

First Commit w/UI for Login inside testing branch
Fahad0819 committed on Jan 23

Commits on Jan 20, 2023

Yarn working and readme steps for development
Eckhoee committed on Jan 20

Commits on Jan 17, 2023

Update README.md
Jordan3617 committed on Jan 17

Delete ProposalBrief.pdf
Eckhoee committed on Jan 17

Add files via upload
Eckhoee committed on Jan 17

Add files via upload
Eckhoee committed on Jan 17

Initialize project using Create React App
Eckhoee committed on Jan 17

Newer Older

Commits on Feb 27, 2023

Fixed UI bugs with chatbot integration
Jordan3617 committed on Feb 27

Update Chatbot.js
RobertMorabito committed on Feb 27

Merge branch 'testing' of https://github.com/Eckhoee/museum-mate into...
RobertMorabito committed on Feb 27

Implementation of the first version of the Chatbot (Rufis).
RobertMorabito committed on Feb 27

Commits on Feb 25, 2023

Updated chatbot UI/layout and changed to be a pop-up available on all...
Jordan3617 committed on Feb 25

Commits on Feb 12, 2023

Updated chatbot page and frontend. Added basic home and directions/ma...
Jordan3617 committed on Feb 12

Commits on Feb 5, 2023

Google authentication
Fahad0819 committed on Feb 5

Commits on Feb 2, 2023

Changed front end background color
Eckhoee committed on Feb 2

Commits on Feb 1, 2023

blueprint for gpt integration and working shortcut buttons
Eckhoee committed on Feb 1

Commits on Mar 10, 2023		
Merge branch 'testing' of https://github.com/Eckhoe/museum-mate into...	aa25ffb	<>
Fahad0819 committed on Mar 10		
Updated admin page with database populated	80f13f4	<>
Fahad0819 committed on Mar 10		
Commits on Mar 8, 2023		
Format fixes	72bb595	<>
RobertMorabito committed on Mar 8		
Bug fixes for recent push	f13fbaf	<>
Fahad0819 committed on Mar 8		
Commits on Mar 2, 2023		
Allowed changing of database feilds	93fa748	<>
danadobro committed on Mar 2		
Bug fixes	a006c5a	<>
RobertMorabito committed on Mar 2		
Added translation functionality	e67b910	<>
RobertMorabito committed on Mar 2		
Merge branch 'testing' of https://github.com/Eckhoe/museum-mate into...	a470468	<>
RobertMorabito committed on Mar 2		
Dynamic chatbot update	5e1c6f7	<>
RobertMorabito committed on Mar 2		
Commits on Mar 1, 2023		
Added loading animation for chatbot replies, chat default open on its...	caa151c	<>
Jordan3617 committed on Mar 1		
STT and TTS Implementation	1376835	<>
sh17kq committed on Mar 1		
Added Museum information	841a36c	<>
RobertMorabito committed on Mar 1		
Prompt Prefix revisions	fc69509	<>
RobertMorabito committed on Mar 1		
Bug fixes with prompting structure.	7fe721a	<>
RobertMorabito committed on Mar 1		
Commits on Mar 13, 2023		
Merge branch 'testing' of https://github.com/Eckhoe/museum-mate into...	08d792a	<>
Fahad0819 committed on Mar 13		
.env security fix	379de98	<>
Fahad0819 committed on Mar 13		
Removed depreciated files	4751e08	<>
Eckhoe committed on Mar 13		
Added toggle to TTS	9873966	<>
Jordan3617 committed on Mar 13		
Merge branch 'testing' of https://github.com/Eckhoe/museum-mate into...	a115832	<>
RobertMorabito committed on Mar 13		
Chatbot 2.0	ee15baa	<>
RobertMorabito committed on Mar 13		
Admin page bug fixes	8d27f47	<>
Fahad0819 committed on Mar 13		
Merge branch 'testing' of https://github.com/Eckhoe/museum-mate into...	b092b39	<>
RobertMorabito committed on Mar 13		
Formatting and bug fixes, Levenshtein distance	7c44d44	<>
RobertMorabito committed on Mar 13		
Commits on Mar 12, 2023		
Text to Speech & Speech to Text	acf227d	<>
Fahad0819 committed on Mar 12		
Added language selection to chatbot	db6b8b6	<>
Jordan3617 committed on Mar 12		
Commits on Mar 11, 2023		
Added multimedia for chatbot replies (video and image)	a5e3eec	<>
Jordan3617 committed on Mar 11		

Commits on Apr 21, 2023

- Bug fixes** RobertMorabito committed last week [497622d](#) [↔](#)
- Merge branch 'testing' of https://github.com/Eckhoee/museum-mate into...** RobertMorabito committed last week [7b18100](#) [↔](#)
- Bug fix for responses** RobertMorabito committed last week [e417a83](#) [↔](#)

Commits on Apr 9, 2023

- Allowed for the getPath function to be public** Eckhoee committed 3 weeks ago [6fad8c8](#) [↔](#)

Commits on Apr 7, 2023

- Merge branch 'testing' of https://github.com/Eckhoee/museum-mate into...** RobertMorabito committed 3 weeks ago [bae347f](#) [↔](#)
- Locations added and database search added** RobertMorabito committed 3 weeks ago [8b7c5a4](#) [↔](#)

Commits on Apr 4, 2023

- Mobile update** Jordan3617 committed last month [9b5e171](#) [↔](#)

Commits on Apr 2, 2023

- Directional map is working, just need to implement** Eckhoee committed last month [38883ba](#) [↔](#)

Commits on Mar 14, 2023

- Fixed TTS bug** Jordan3617 committed on Mar 14 [3fd85ae](#) [↔](#)
- Add image output with responses** RobertMorabito committed on Mar 14 [c418225](#) [↔](#)
- Merge branch 'testing' of https://github.com/Eckhoee/museum-mate into...** RobertMorabito committed on Mar 14 [24b9525](#) [↔](#)
- Database entry now generates a GPTName Tag** RobertMorabito committed on Mar 14 [f321c15](#) [↔](#)

Commits on Apr 24, 2023


































- Directional bug fixes** Eckhoee committed last week [ec58e0f](#) [↔](#)
- Directional bug fixes** Eckhoee committed last week [3464158](#) [↔](#)
- Updates** RobertMorabito committed last week [049a336](#) [↔](#)
- Hope this works lol** Eckhoee committed last week [2e4fc9a](#) [↔](#)
- Added Visual Components/ Directional Updates** Eckhoee committed last week [de58e21](#) [↔](#)
- Colour changes** RobertMorabito committed last week [d6199ac](#) [↔](#)
- Frontend updates** Jordan3617 committed last week [821a22c](#) [↔](#)
- Direction clarification** RobertMorabito committed last week [734d52b](#) [↔](#)
- Chat error handling overhaul** RobertMorabito committed last week [b4e6868](#) [↔](#)

Commits on Apr 23, 2023

- frontend update** danadobro committed last week [466477b](#) [↔](#)

Commits on Apr 22, 2023

- Directions bug fixes** RobertMorabito committed last week [e02c075](#) [↔](#)
- Natural Language Directions** RobertMorabito committed last week [d6a04c2](#) [↔](#)

Commits on Apr 26, 2023		
Update RobertMorabito committed 4 days ago		 
Merge branch 'testing' of https://github.com/Eckhoe/museum-mate into... RobertMorabito committed 5 days ago		 
Update RobertMorabito committed 5 days ago		 
footer update Jordan3617 committed 5 days ago		 
Merge branch 'testing' of https://github.com/Eckhoe/museum-mate into... RobertMorabito committed 5 days ago		 
Copyright notices added RobertMorabito committed 5 days ago		 
UI/bug fixes Jordan3617 committed 5 days ago		 
Commits on Apr 25, 2023		
Update RobertMorabito committed 5 days ago		 
Update RobertMorabito committed 5 days ago		 
Bug fixes and tweaking RobertMorabito committed last week		 
Added footer component/text and removed chatbot page Jordan3617 committed last week		 

3. User Manual

3.1 Home Page

The Museum Mate web app is hosted online at <https://museum-mate.vercel.app> hosted on Vercel, which is a website used for deploying web apps. Once the user has navigated the link they will be greeted with the home page of the Museum Mate website (Figure 1). As shown in Figure 1, there is a brief description of the museum itself and a set of pictures that are displayed in a slide-show-type manner as they rotate with all museum pictures. There are also featured topics displayed on the home page. Figure 2 is the lower half of the homepage, as the home page continues the user is presented with an embedded map that is pinpointing the museum on a Google map within the website. The user is able to interact with the map to find a route to the museum. The users are presented with general information about the museum that included the address, timing, contact information, and admission fees.



Figure 1: Home page of Museum Mate website

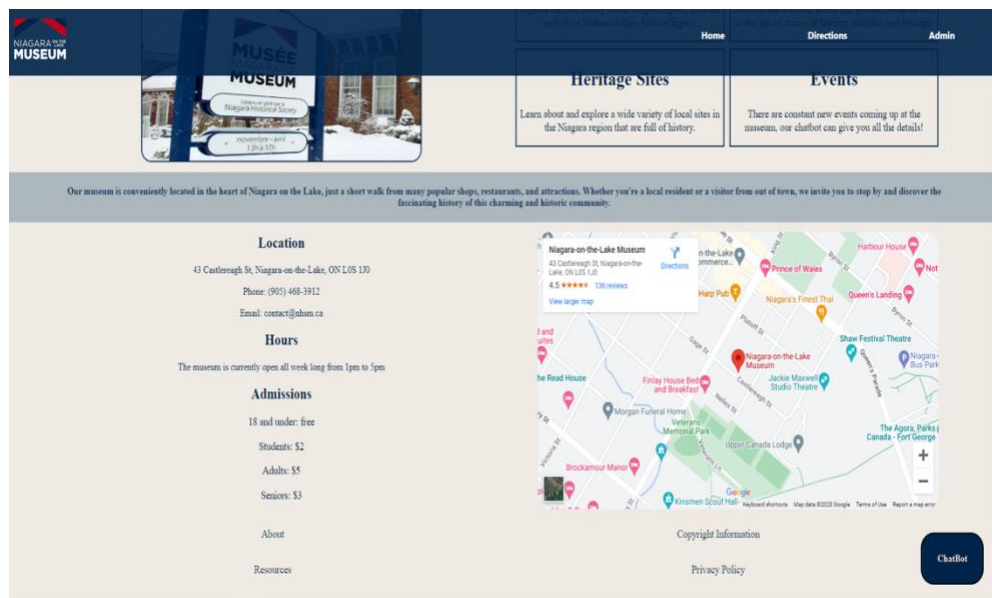


Figure 2: Home page of Museum Mate website

3.2 Admin Page

The navigation bar on the top of the home page has two other pages, the admin page is a feature that not all users will have access to, as it is made for the administrators of the Niagara-on-the-Lake Museum. The feature that the admin page has is the ability to add new artifacts to the database that can be later used to ask questions or any general inquiry. This feature is important as there are new artifacts that are being brought to the museum at all times and it is crucial that all artifacts are accounted for. Once we navigate from the current page to the Admin page we are presented with the button that reads, Sign In With Google, as seen in Figure 3.

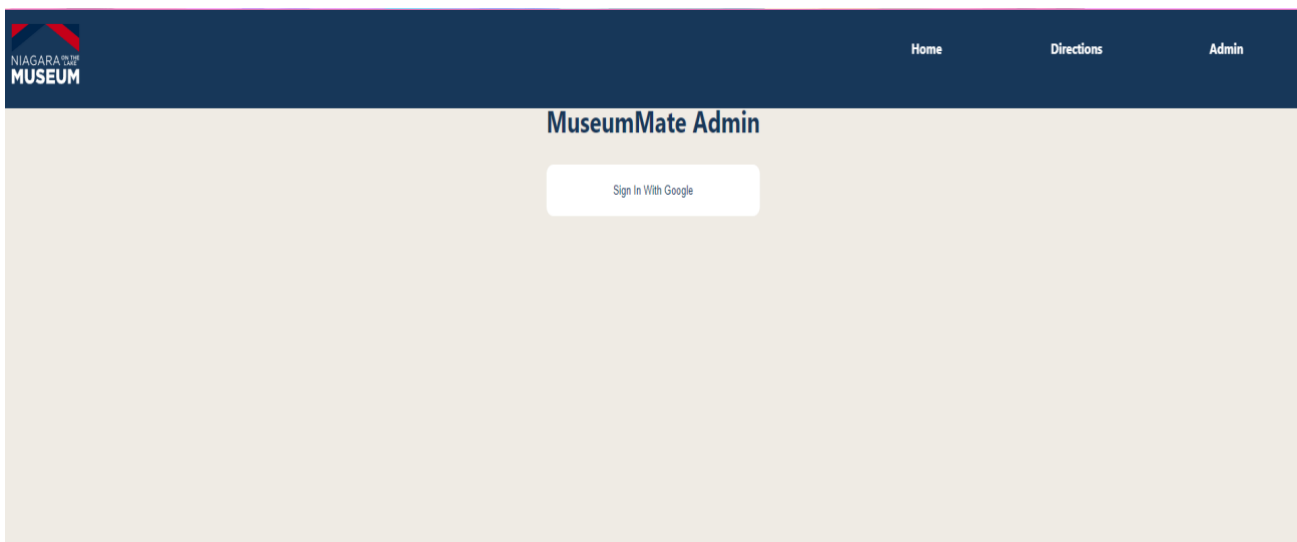


Figure 3: Admin front page

Once the Sign In With Google button is clicked the user is presented with a sign in prompt that gives them the option to choose which google account they would like to log in with (Figure 4). When the admin logs in they are presented with an Add Exhibit button and a Logout button (Figure 5). The admin is then prompted with a form that has many fields such as name, exhibit description, object Id, and more (Figure 6). The form, when filled out is able to update the database that stores all the current artifacts of the museums.

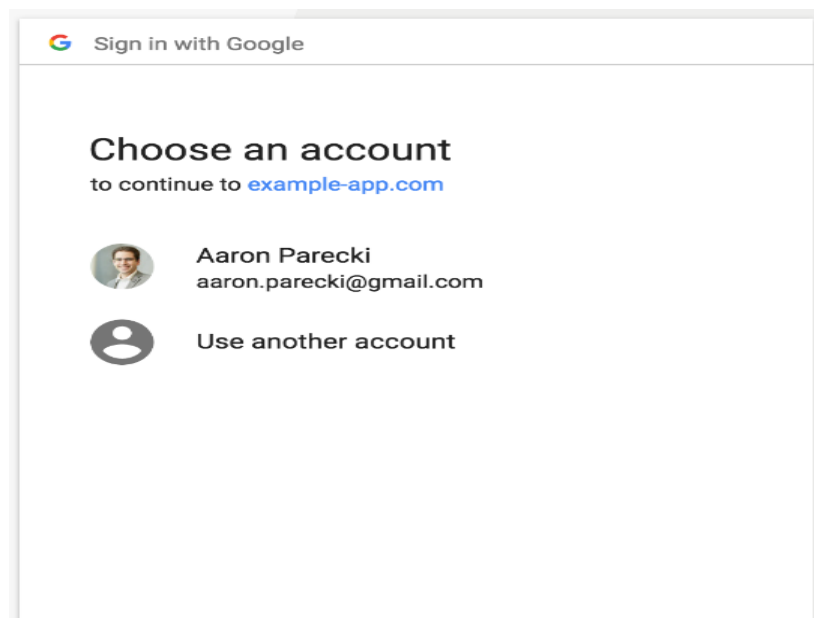


Figure 4: Google sign in prompt

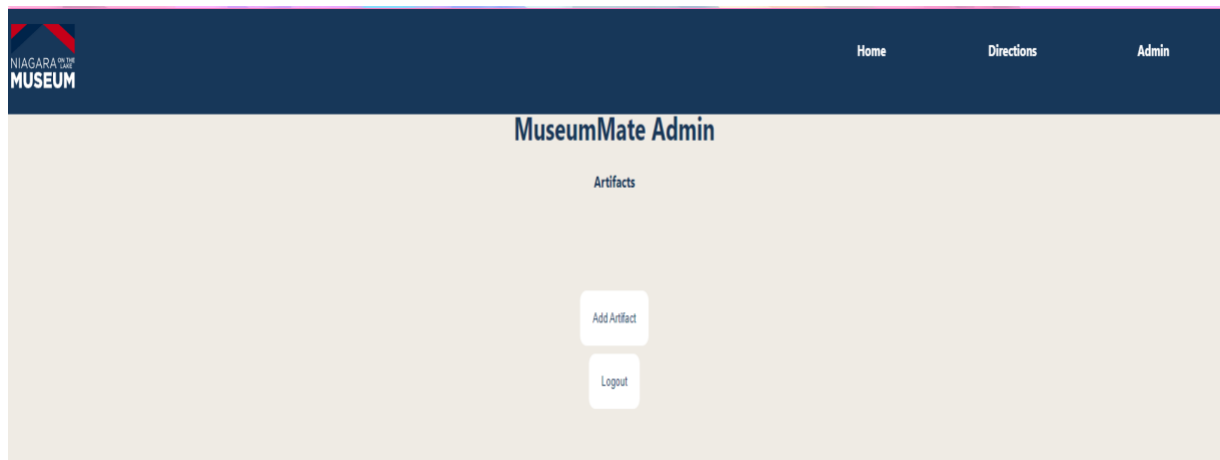


Figure 5: Admin page with add artifact button

The screenshot shows the 'Add Artifact' form within the 'MuseumMate Admin' interface. The form is a white modal box with a dark blue header containing the title 'Add Artifact' and a close button 'X'. The form contains several input fields with placeholder text: 'Enter Name', 'Enter exhibit description', 'Enter date', 'Enter Object ID', 'Enter Names of People', 'Enter the Subject', and 'Enter the Valid Image URLs'. Each input field has a small icon on the right side. At the bottom of the form, there is a dark blue button with the text 'Add' in white.

Figure 6: Add artifact form

3.3 Chatbot Page

The Chatbot is available on every page besides the admin page and is the most crucial aspect of the website as it is made to serve the users and answer any question they may have regarding the museum and the artifacts within it. Once the user clicks the chatbot button the chatbot page appears as a toggle page that introduces itself and then gives suggestions on the type of questions that can be asked (Figure 7).

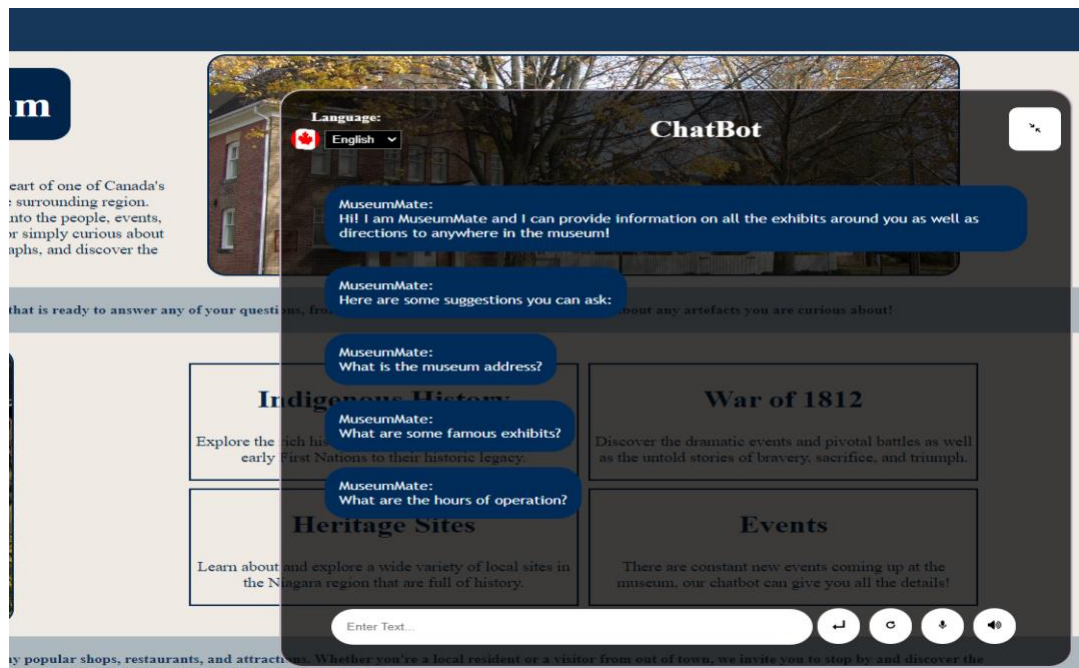


Figure 7: Chatbot toggle page

Once the user can view this page there are many features that they are able to utilize, to start the user is able to click on any of the suggestion and the input box is automatically filled with the question that was clicked making it easier to ask the question. There are three buttons besides the input field that are important for the user to use. The first button located beside the enter button is the reset button that can clear the current conversation with the Chatbot. The second mic button is used for speech to text feature that when clicked requires the webpage to access the user's mic and when enabled the user can talk and the words are translated into the input field. The next button that is the volume icon can speak the response of the Chatbot. When the user asks the Chatbot a question about any of the artifacts it responds with a short description followed by any picture of the artifact on record (Figure 8).



The Chatbot is also able to combine with the directional component and with prompts such as the “I am at the (exhibit) how do I get to (location)”. With the use of the mapped museum the Chatbot is able to give human like directions as seen in Figure 9.

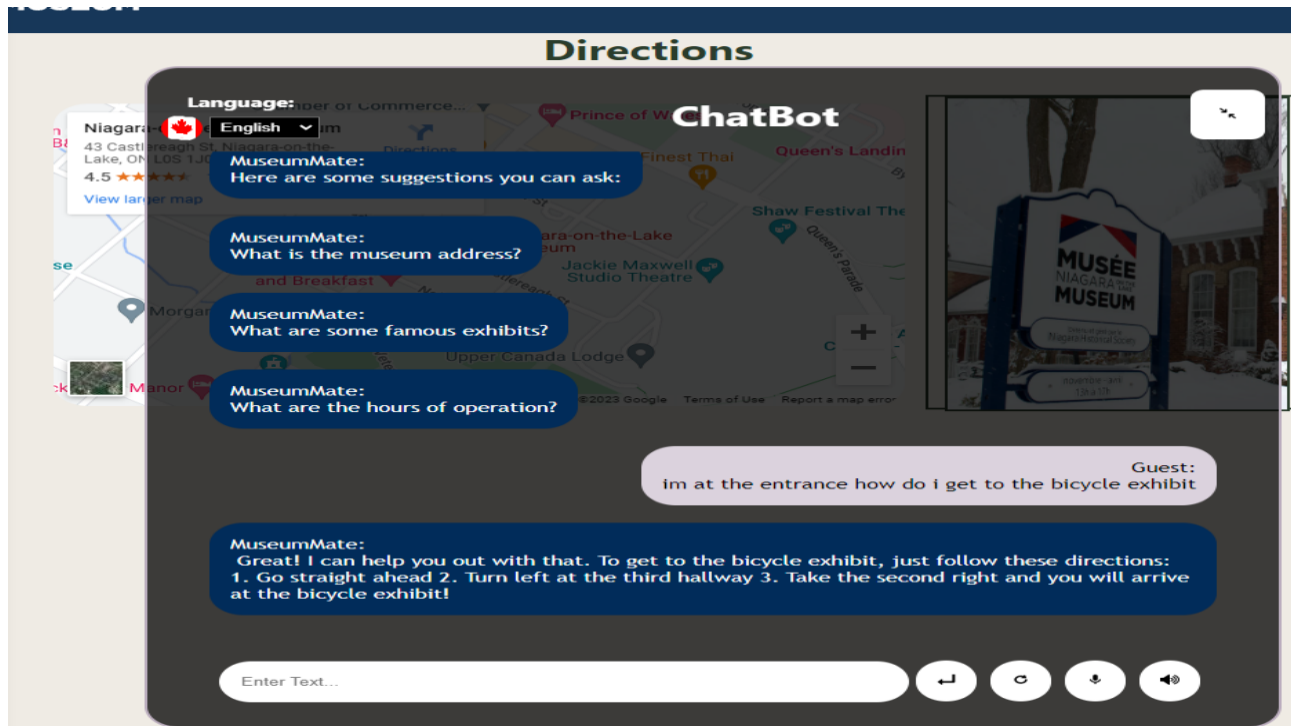


Figure 9: Reply with the directions by the Chatbot

3.4 Directions Page

The directional page is a more in depth look at the map with the ability to locate the museum and with the help of Google maps it is able set direction to the museum. This page is also collaborating with the Chatbot in order to incorporate the directional component for the chatbot and help in the effort to provide the direction to the desired location (Figure 10).

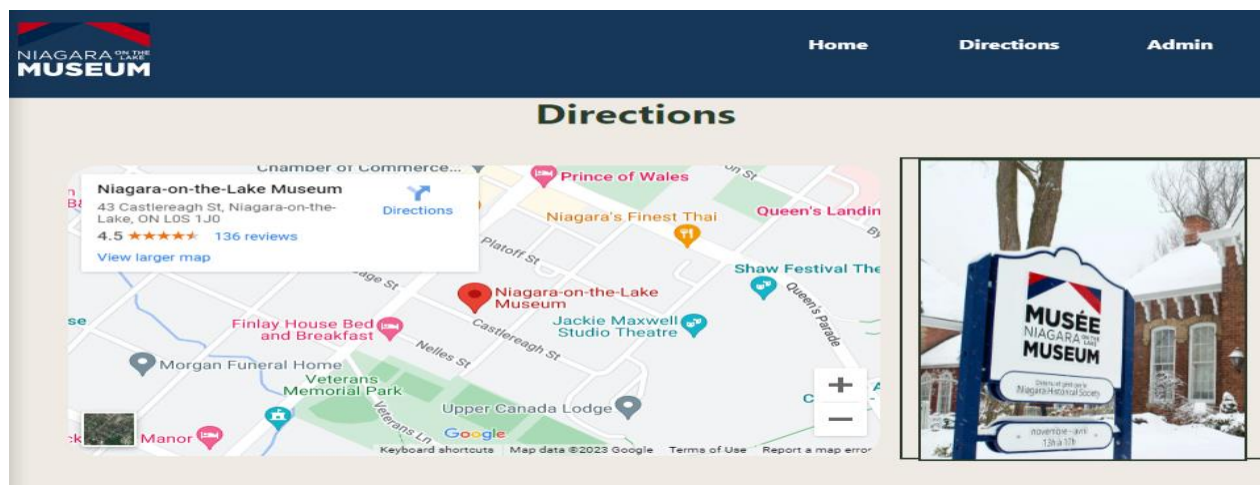


Figure 10: Directions page

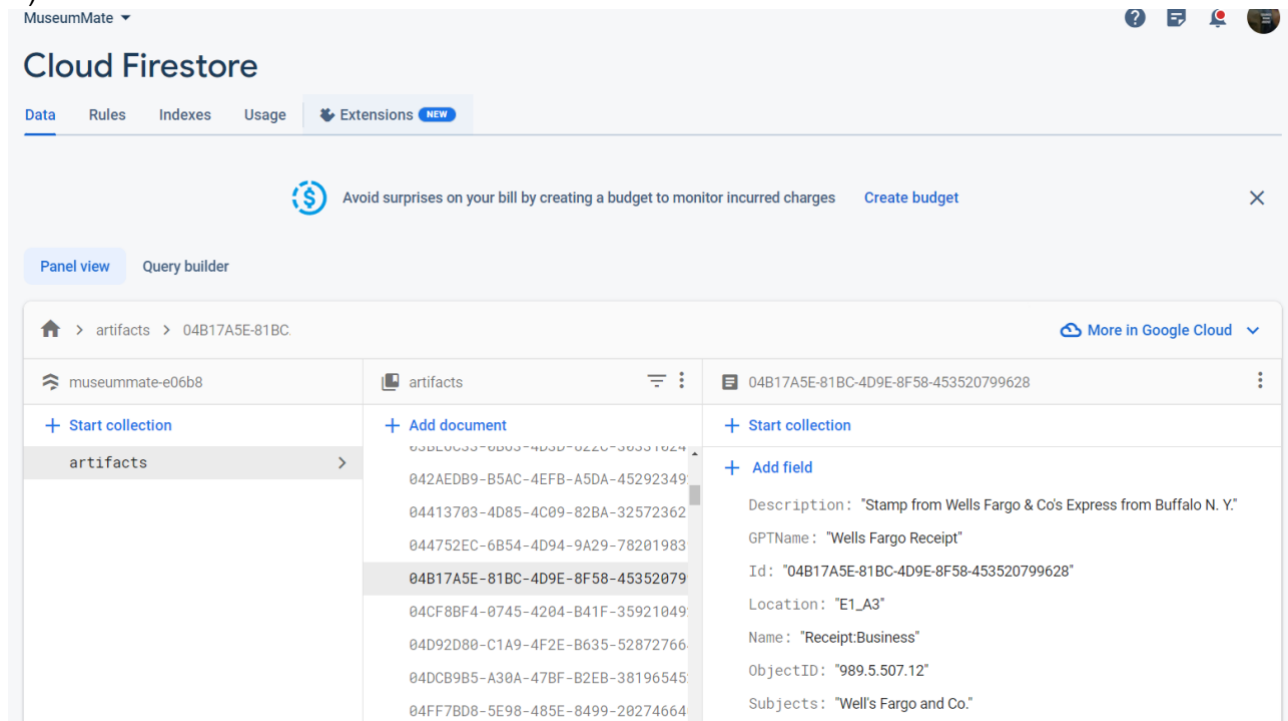
4. Technical Manual

The following section provides in-depth information for how the application works from a technical standpoint.

4.1 Database

The creation of the database had many steps. The initial step was the to choose what we would use for the database, upon further research we had reached the conclusion that the use of firebase would be very convenient and with the majority of the team having worked with such a tool it would prove quite easy to use. The reason this would be very convenient place to set up the database would be the reason that the need for an authentication service could be done on the same platform, this is helpful as there are less places the development team would need focus their attention. When looking at the database there were two options within firebase to host the data of the museum the first was the Realtime database and the second was the Firestore. The use of Firestore was appealing to the development team as it is relatively recent and has better support.

Once the database was set up the challenge became when we had to populate the database with the current artifacts that were already on the museum website database. To extract all the data from the museum website the method of scraping was used. A script was written using puppeteer. The script was created so that it can navigate to the artifact data page and went through all 113 pages and extracted over 1000 artifacts. The artifacts included subheadings such as title, subject, description, and date. Once the scrapper was able to isolate the information on the page it then created a Json file and populated the file with the subheading and data. Once the Json file was created the use of Firestore was crucial as it had a simple method of uploading a Json file that generated a whole database using the subheading and the data placed within the Json (Figure 11)



Firestore also served as a means to authenticate users for the admin page and this was due to the use of the Google authentication API that allows any users that want to login the ability to do so using their Gmail account.

4.2 GPT & Chatbot

4.2.1 Background Research and approach

The operation of using a Large Language Model (LLM's) to perform tasks within an intricate system is one that requires careful consideration of options to create something that is adaptable, while keeping performance. Most modern LLM's contain billions of parameters such as GPT-3 which has 175 billion (Brown et al., 2020) all of which can be tuned and adjusted to fit better at certain tasks. A commonly used approach is fine-tuning, which adjust parameter values at different sub-surface layers within a language model, however, this method can be costly and rigid depending on the size of the model and the variability of tasks that the LLM needs to be trained to perform (Howard and Ruder., 2018). A new method that is becoming increasingly more popular than fine-tuning is prompt engineering, which programs an LLM by instructing it to adjust its own surface level parameters when handling downstream tasks by using cleverly devised prompts, outlining what must be changed (White et al., 2023). Evidence shows that using prompt-based methods, especially alongside few-shot methods which provide examples of responses, and zero-shot methods which push a LLM to perform without examples, perform considerably well on downstream tasks while being light weight and robust (Wei et al., 2022; Kojima et al., 2022; Jin et al., 2022; Gao et al., 2021; Schick and Schütze, 2021; Brown et al., 2020). For this reason, we chose to use prompt engineering as our main form of training GPT-3 for all of our tasks, which we found to be extremely dynamic and modifiable while maintaining consistent performance. We used a Prompt Template Engineering approach, using Prefix Prompts as the prompt shape and Manual Template Engineering to craft our prompts (Liu et al., 2023). This was then combined with few-shot training to perform classification tasks and directional responses (examples of outputs provided) and zero-shot learning to perform informational responses (instructions of tone and style with no output examples).

4.2.2 GPT-3 Functions

Two functions were set up to query GPT-3, each with their own purpose. The first and most used, called GenerateBasic, is a function that queries GPT-3 using parameters tuned for general prompting, such as having a low temperature setting such that responses are consistent, and variability is reduced. GenerateBasic is used for all classification and translation tasks. The next function is GenerateChat, which has its parameters tuned to be more dynamic and creative such that when users ask the same thing twice, it is slightly different each time and increases the "human-like" feel of the bot. GenerateChat is used only at the end of the directional chat and information retrieval chat to create the final user-facing response in natural language.

4.2.3 Classification Tasks

To evaluate a user response and understand the specifics of what is being asked, some type of string parsing and/or data extraction must be done. To handle this, we implemented a series of specifically designed prompts that are paired with few-shot examples and fed into GPT-3 alongside the user input for the purpose of extracting information. The structure of these prompts are as follows:

```
[Prefix instructions]:
Examples: [List of examples]
Input: [user input]
Output: < GPT – 3 response continues from here >
```

This format implements a few of the above-mentioned techniques and is effective at extracting the information specified. This also allows us to dynamically add examples from a string variable, should they need to be changed, as well as dynamically adding the user input as it changes. The only issue with this technique is that GPT-3 frequently includes extra spaces and new lines, however, this is handled using some regex functions that either remove these extra characters or ignore them as is the case with the string comparison algorithm for database searching.

4.2.4 Directional Chat

When the chat function is run, the first task it completes is evaluating if the user is looking for directions, or just wants to know some information. If the user is in fact asking for directions, then the directional chat is entered, if not the information retrieval chat is entered. Upon entering the directional chat, the user input is evaluated by GPT-3 to extract both the current artifact the user is at and the artifact they are trying to get to (these both must be specified). If one of these variables is not found, the language model returns “N/A” for that variable to indicate it wasn’t found, and the user is asked to re-enter the missing information. The conversation type evaluation is then bypassed as they are still discussing directions and the user has a chance to enter the missing information. Once all information is gathered, it is first compared to a list of static points (e.g., washroom, front door), then, if not already found, the strings are searched within the database, finding the closest match, and returning the location of the artifact that matched. At this point, there are now two locations, which are formatted using regex functions and can be fed into the pathfinding algorithm. The pathfinding algorithm then returns the path as a list of directions such as “straight, right, left, slight right”, these are then fed into GPT-3 which uses few-shot learning to convert the path into a natural sounding, numbered list of directions for the user to follow. This interaction is then added to the current chatlog so it can be tracked and used in the next query.

4.2.5 Information Retrieval Chat

Information retrieval uses a paradigm similar to that of directional chat but is much simpler. For information retrieval, the user input is evaluated the same way that is done in the directional chat, but this time GPT-3 is asked to retrieve the subject of the conversation, which can be an artifact or exhibit name, museum info such as phone number, or just general conversation of nothing specific was found. This response is then checked and if the user was looking for museum information, then the information they asked for is retrieved from a list of museum information. If the user was asking for about a specific artifact, then the extracted artifact name is compared against the database and the description and image of the closest match is returned. If the user was just inputting general conversation, then no information is retrieved. At the end of the function a GPT-3 response is generated by passing in the user input, the chatlog, and any retrieved data.

This interaction is then added to the current chatlog as well so it can be tracked and used in the next query.

4.2.6 Translation

A small final feature for the user that is used for accessibility is a GPT-3 generated translation of the output produced by either the directional or information retrieval chats. This simply works by prompting GPT-3 with:

"Translate the following input into [language]: [input]"

Where *[language]* is a variable that can be changed by the user and *[input]* is the output that was produced earlier.

4.3 Directional

4.3.1 Pathfinding

The directional component contains a graph/map of the museum which is comprised of a custom node class. Each node in the graph has a name, x and y coordinate, a list containing neighbours of that node, and various other variables for the A-Star algorithm to determine the shortest path between a source point (ideally the location where the user is) and the destination (where the user wants to go). Nodes can have a name that represents either a decision point (i.e., Entrance, Desk, or a Corner) or a visitable location (i.e., Exhibit One or Exhibit Two). Nodes that are represented inside of an exhibit will have the exhibit number and the artifact number associated with it. An example of a node inside exhibit one and is the second artifact in that exhibit would be name "E1_A2". This naming convention follows for all nodes that are visitable. To visit a specific artifact that is held within the museum, a user would not search for the node itself, but rather the artifact in which they are coming from and going to. Each artifact has an assigned location/node within the database that is read when the user queries the chatbot. Once the locations are retrieved, we use an A-Star algorithm using the Euclidian distance heuristic to determine the shortest possible path from the source to the destination. This information can be displayed to the user in two forms, either by a written response from the chatbot, or an image visualized and exported to be displayed within the chatbot windows itself.

4.3.2 Written response

The written response is a two-stage process that begins when the optimal path is fully evaluated. The first component receives an array containing the \bar{x} and \bar{y} values of each node from the path found by the pathfinding algorithm. Then, the algorithm calculates the angle between each node using the $\text{atan2}()$ function from the Math library, which is then compared against a switch statement that categorizes the path between the nodes as their cardinal direction, based on where the angle falls within a circle. For example, if the angle is 90° , then it is classified as "north" and stored in a new array. This new array is then fed into the second component which compares the directions of two paths and determines which directional instruction to give the user, based on how close the paths are to each other. For example, if a path is "north" and the next path is "north", then the instruction is "straight", if a path is "north" and the next path is "west", then the instruction is

“left”. This path is then outputted to GPT-3 to convert into a natural language representation.

4.3.3 Visualization response

The visualization utilizes the path that is generated using the A-Star algorithm along with images that represent that various path between two nodes. Each node as n number of nodes that can be taken and n corresponding images to that node. Each image is given a name of the source node and destination node concatenated together making searching through a file structure very easy. All that must be done is checking both possible name of a node. Since there are two possible names, depending on where the user is searching from, we check to see which version is correct using the fs package. This ensures that we won't get any errors from loading the images. The images file is loaded onto a base map which contains all the possible nodes you can visit. Each of the path images has a transparent background so there is no worry about formatting the images to the correct position within the base map. All the image manipulation is done using sharp which handles the merging of the files together by simply taking the file names into a list of images to be merged. Once the final image is generated with the correct path displayed, it will output to the chatbot.

4.3.4 References

- J. Howard and S. Ruder, “Universal language model fine-tuning for text classification,” *arXiv.org*, 23-May-2018. [Online]. Available: <https://arxiv.org/abs/1801.06146>. [Accessed: 30-Apr-2023].
- J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, “Chain-of-thought prompting elicits reasoning in large language models,” *arXiv.org*, 10-Jan-2023. [Online]. Available: <https://arxiv.org/abs/2201.11903>. [Accessed: 30-Apr-2023].
- J. White, Q. Fu, S. Hays, M. Sandborn, C. Olea, H. Gilbert, A. Elnashar, J. Spencer-Smith, and D. C. Schmidt, “A prompt pattern catalog to enhance prompt engineering with chatgpt,” *arXiv.org*, 21-Feb-2023. [Online]. Available: <https://arxiv.org/abs/2302.11382>. [Accessed: 30-Apr-2023].
- P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, “Pre-train, prompt, and predict: A systematic survey of prompting methods in Natural Language Processing,” *ACM Computing Surveys*, 01-Sep-2023. [Online]. Available: <https://dl.acm.org/doi/full/10.1145/3560815>. [Accessed: 30-Apr-2023].
- T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, “Language models are few-shot learners,” *arXiv.org*, 22-Jul-2020. [Online]. Available: <https://arxiv.org/abs/2005.14165>. [Accessed: 30-Apr-2023].

- T. Gao, A. Fisch, and D. Chen, "Making pre-trained language models better few-shot learners," *ACL Anthology*, Aug-2021. [Online]. Available: <https://aclanthology.org/2021.acl-long.295/>. [Accessed: 30-Apr-2023].
- T. Kojima, S. (S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot Reasoners," *Advances in Neural Information Processing Systems*, 06-Dec-2022. [Online]. Available: https://papers.nips.cc/paper_files/paper/2022/hash/8bb0d291acd4acf06ef112099c16f326-Abstract-Conference.html. [Accessed: 30-Apr-2023].
- T. Schick and H. Schütze, "Exploiting cloze-questions for few-shot text classification and Natural Language Inference," *ACL Anthology*, May-2022. [Online]. Available: <https://aclanthology.org/2021.eacl-main.20/>. [Accessed: 30-Apr-2023].
- W. Jin, Y. Cheng, Y. Shen, W. Chen, and X. Ren, "A good prompt is worth millions of parameters: Low-resource prompt-based learning for vision-language models," *ACL Anthology*, May-2022. [Online]. Available: <https://aclanthology.org/2022.acl-long.197/>. [Accessed: 30-Apr-2023].

4.4 Front End

Made with React framework (JSX) and default CSS/HTML. The component layout and functionality including all logic were done through JavaScript, while all visual layouts and designs, including coloring, styling sizing was done through basic CSS, with no additional libraries or frameworks. Everything was rendered in HTML through React automatically for final display as a web application. React-router-dom was used to handle page navigation. Google Authentication was used for login as an admin. Google Maps was used to display locations. The application has custom Desktop and Mobile versions, with both having the same components but different layouts.

Components (main):

- Home/landing (page)

- Directions (page)

- Admin login (page)
 - Admin panel

- Chatbot component (on home/directions pages)

All pages contain:

- Navigation Bar at the top (with current links below)
 - Home
 - Directions
 - Login

- Footer at the bottom (with current links below)

- About
- Privacy Policy
- Copyright
- Resources

Chatbot contains:

Initial minimized sticky toggle button, that is displayed at the button right of all pages. This button expands into the chatbot, taking on the right half of the viewport on the desktop version, and full screen on the mobile version, when clicked.

Header section

- Language selection component
- Name/Title
- Minimize toggle option

Main section

- Text/Message List
- User input
- Chatbot response
- Possible image
- Possible video

Footer section

- Text input field
- Enter/Submit button
- Reset text/message list button
- Speech to text for input field button
- Text to speech toggle on/off button

For readability and maintainability, the frontend is separated into different JavaScript (.js) files, corresponding the visual page it represents, with each having an accompanying Cascading Style Sheet (.css) file which holds the design and styling for the specified page. The files responsible for the frontend in this project are:

- App.js
- Home.js
- Directions.js
- Login.js
- Chatbot.js

The base of the system is through the app page, which is used as a container to render the different components which makes a page in the viewport, default being the navigation bar, footer, and the current page.

To access the administrative panel for manipulating the database, which is feed into the Chatbot, Google authentication is used on to login page. Only verified accounts are allowed to login. This allows the software system to be modular depending on requirements (eg provide the same service to multiple other museums).

Within the Home/Directions page, Google maps is used to provide a visual, interactive map featuring the location of the museum, through which the user can determine the needed directions to physically locate the museum.

Within the Chatbot.js file, multiple separate components exist as helper methods.

Text creates and modifies the current list of messages that have been exchanged between the user and Chatbot. Each text input is processed to determine the sender of the message, the placement within the list (left for Chatbot replies and right for user inputs), and whether there is additional image or videos contained with the response. This component serenades the list as new text input is generated to keep it up to date. A react hook is used to keep the list scrolled to the latest message (bottom of the list). Another hook is used to display an initial suggestion set for the user of possible input for the Chatbot to respond to, after a brief delay if the user has not interacted with the text field. Loading component is used here in between response from the chatbot to indicating processing of user request.

Multiple different handleSubmit functions exist to handle resting the text list, submission of user text, toggling speech to text on or off and using speech to text.

Chatbot contains the main visual representation of the Chatbot to be displayed to users and handle the corresponding logic, while chat is used to separate the actual Chatbot response processing logic for easier maintainability.

There is an additional file, Components.js and its CSS file, which holds the individual general component used through the application, allowing for low coupling in the software and easier future maintenance and modification for the different elements, while reducing code duplication.

The individual components from Componenets.js that are used in the application are as follows:

NavigationBar:

Used to display the navigation bar, located at the top, across all pages on the application. The number of links is customizable as well, allowing future pages to be easily integrated into the software. The displayed router link in address bar being “/directions” and the JavaScript file in the software being “Directions” in the example below.

ImageChangerComponent:

Used to display a set of rotating images on the Home page. The set of images is pulled from the images array and is customizable, as is the time each image is displayed for (default value 5000).

PageRoute:

Using the React Routers, this component handles the navigation within the web application, setting up the appropriate page file for viewing when the user navigates to it through the navigation bar. An isMobile check is done when pulling a page for viewing to determine the page layout, with default value being ≤ 768 (standard mobile display size). Due to the difference in viewport size between desktop and mobile devices, the navigation bar uses this to determine the type to display, a traditional bar with options going left to right for larger screens, and a toggle to pull up drawer with the page options top to bottom for smaller screens.

Footer:

Used to display the footer component on all pages, regarding general site information. The default design is two columns containing two rows each. This is also customizable, allowing for more options to be displayed both horizontally and vertically. Each element is clickable, displaying a popup in the center of the screen with the clicked option as a header, information regarding the clicked topic (eg the Privacy Policy option brings up a popup with how user information is used by

the site) and a close option at the bottom. If the information exceeds the screen size, scrolling is enabled. The popup is handled by a separate Popup component.

Loading:

Used in the Chatbot itself to display a loading animation, signaling to the user that a text response is being generated by the Chatbot. With future additions to the web application that are resource heavily, such as contains large amounts of images, the loading component can be used between these pages as well.

LanguageSelector:

Used in the Chatbot itself to provide users with the ability to change the language of interaction. Both the name of the language and a corresponding flag image of it are displayed together. The selected option applies both to the text output and audio output if the text to speech option is enabled. The language is changeable anytime through the conversation. This is customizable and future languages can be added through finding the flag image and adding it to the flagList array and filling in the name of the language in the language select section.

The flag images are linked through URL from <https://flagpedia.net/download/api>.

5. Scrum

The success of the Museum Mate chatbot may be credited to the adherence of the Scrum methodology. The team made sure to follow the scrum process, specifically the sprint cycle, which includes bi-weekly scrum meetings facilitated by the scrum master, sprint reviews after each sprint, and sprint retrospectives after each sprint. Below is a complete guide on all sprint meetings, and what was discussed or handled.

Sprint	Date	Meeting #	Type of Meeting	Issues Discusses
1	Jan 24	1	Daily Scrum	<ul style="list-style-type: none"> Product backlog created Requirements defined User stories created Sprint 1 backlog created
1	Jan 26	2	Daily Scrum	<ul style="list-style-type: none"> Discussed JavaScript as major language used to code application in Fine tuning research discussed GPT funding and research Understand individual tasks for sprint
1	Jan 31	3	Daily Scrum	<ul style="list-style-type: none"> Discussed google as best option for admin login Plan for linking database and chatbot Planned for database population
1	Feb 2	4	Daily Scrum	<ul style="list-style-type: none"> Demonstrated basic running front end for input and output

1	Feb 7	5	Daily Scrum	<ul style="list-style-type: none"> • Visited NOTL museum and retrieved their database for web scraper search •
1	Feb 9	6	Sprint Review	<ul style="list-style-type: none"> • Checked all work completed against product backlog, determined team was on track and goals were being met
1	Feb 14	7	Sprint Retrospective	<ul style="list-style-type: none"> • Discussed team can improve on communication in our discord channel • Analyzed idea to allocate for tasks in future sprints to complete more backlog items in shortest amount of time
2	Feb 17	8	Daily Scrum	<ul style="list-style-type: none"> • Sprint 2 backlog created
2	Feb 21	9	Daily Scrum	<ul style="list-style-type: none"> • Decided on mobile version for mobile web browsers
2	Feb 23	10	Daily Scrum	
2	Feb 28	11	Daily Scrum	<ul style="list-style-type: none"> • Prompt based finalized as method for chatbot • Used web scraper for real data database population •
2	Mar 2	12	Sprint retrospective /Sprint review	<ul style="list-style-type: none"> • Checked all work completed against product backlog, determined team was on track and goals were being met
3	Mar 7	13	Daily Scrum	<ul style="list-style-type: none"> • Sprint 3 backlog created
3	Mar 9	14	Daily Scrum	<ul style="list-style-type: none"> • Generated floor plans for directions • Decided on A star algorithm • Decided on written instructions format
3	Mar 14	15	Daily Scrum	<ul style="list-style-type: none"> • Refined chatbot to accommodate directional component • Chatbot features discusses such as text to speech, speech to text, translations, microphone access, etc
3	Mar 16	16	Sprint Review/Sprint Retrospective	<ul style="list-style-type: none"> • Checked all work completed against product backlog, determined team was on track and goals were being met
4	Mar 21	17	Daily Scrum	<ul style="list-style-type: none"> • Sprint 4 backlog created
4	Mar 23	18	Daily Scrum	<ul style="list-style-type: none"> • Discussed more efficient string search for databases
4	Mar 28	19	Daily Scrum	<ul style="list-style-type: none"> • Created refined map to be

				<ul style="list-style-type: none"> used by users to navigate Plans for UI updates
4	Mar 30	20	Sprint Review/Sprint Retrospective	<ul style="list-style-type: none"> Checked all work completed against product backlog, determined team was on track and goals were being met besides visualization of directional component
5	Apr 4	21	Daily Scrum	<ul style="list-style-type: none"> Discussed removal of directional visuals and added maps for direction to museum
5	Apr 6	22	Daily Scrum	<ul style="list-style-type: none"> Sprint 5 backlog created
5	Apr 11	23	Daily Scrum	<ul style="list-style-type: none"> Made google form for user test batch Created test cases
5	Apr 14	24	Daily Scrum	<ul style="list-style-type: none"> No meeting due to exams
5	Apr 18	25	Sprint Review	<ul style="list-style-type: none"> Checked all work completed against product backlog, determined team was on track and goals were being met

6. Testing

6.1 Automated System Testing

Automated System testing was conducted by the developers where in a script was written once again that navigated to the website where the app was deployed. Script then opened the chatbot and was ready to receive input to which it would generate a response. This input/output test was conducting by taking a text file of GPT-names that was used as a way for the chatbot to identify certain artifacts within the database. The artifacts were fed to GPT and GPT was ask to summarize the artifact description in 4 to 5 words which then was stored in the database as the GPT-name. These GPT-names that were generated were located in a Json file which with the use of the script extracted the name and populated to the chatbot input field. Once the script is started with the specification of the number of times that the test should run the script console logs the inputted GPT-name and the response from the chatbot if the response matches the input GPT-name artifact we know that the test was a success.

The following is an example of a test that has succeeded:

```
George Ball's 1835 Farm
Yes, I can provide you with some information about George Ball's farm located in present-day Virgil, Niagara on the Lake. This property of the 1,000 acres granted to his father following the American Revolution and George built a large brick home ca. 1820. It is noted on Black Swamp Road (today highway 55). The home still stands and is located at 421 Hunter Road and fondly known as Locust Grove. Is
```

The following is an example of a test that has failed:

```
William Wilson Deed.
I'm sorry, but I don't have that information at the moment. Is there anything else I can help you with?
```

With the use of the following loop it goes through the array and checks if any of repeated elements are present and if there isn't it proceeds to console.log the element in question. As seen above it was evident that when the chatbot failed to identify the GPT-name with an artifact the response contained the word 'sorry' and so for the following if statement we used the .includes method to verify that if there was a sorry within the message it had failed and the variable called "test" would increment by one after each response (figure 12).

```
if (content !== '' && !uniqueContents.includes(content)) { // Check if content is unique
  console.log(content);
  if (content.includes('sorry')) { // Check if content contains the word 'sorry'
    test++; // Increment test by 1
    console.log(`Test: ${test}`);
  }
  uniqueContents.push(content); // Add content to array of unique contents
}
```

Once the test was completely set up we ran the script and specified for it to run for 50 elements and as the counter for the amount of failed cases increased it stopped on the 50th run with the amount of failed runs totaling to 10.

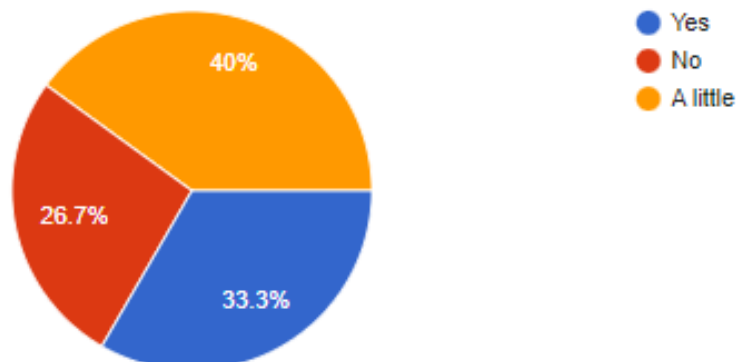
```
Test: 10
Hi, great to meet you! It looks like you're interested in James R. Benson and Mary Ann Ingersoll's marriage certificate from 1836. This certificate is part of the museum's permanent collection and can currently be viewed in the third floor exhibit hall. Let me know if you need help finding your way there, or if you have any other questions about the museum exhibits!
Test: 10
```

6.2 User Testing

In user testing, a batch of users were gathered by the scrum master and developers. This was a crucial stage in our testing phase as it allowed the team to receive feedback and responses on whether the system fulfills users' goals. The idea in this testing phase was to give the batch of 30 users 10-20 minutes to use the application and its various features. Once the application was used by the users, they were to complete a survey highlighting their experiences and providing the development team with their feedback on what can be improved in the application to better serve their purpose. Some of the results from the survey are below:

Is the website visually appealing?

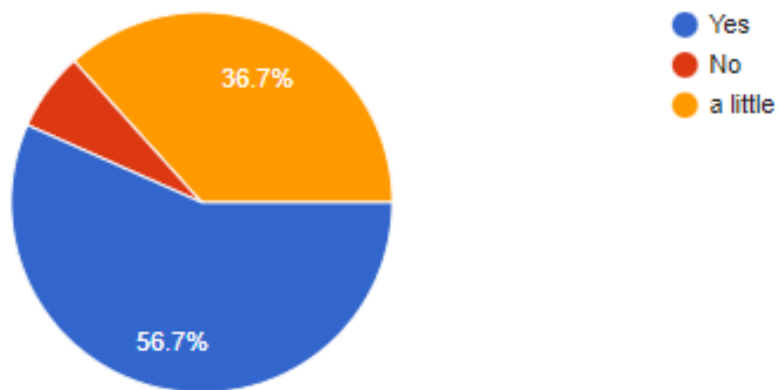
30 responses



As shown above, the original webpage that was used for testing had the majority of the users not pleased with the visual aspect as only 33% agreed that the website was visually appealing while the other 66% thought it was slightly appealing or not appealing at all. This test was conducted in order to measure the appearance of the front end and visual component. With this test we learned that more effort should be put into the visual component of the front end and make the color scheme appeal to the user.

Is it easy to navigate the website?

30 responses



The next question was if the website was easy to navigate, this question was used to identify if the website was user-friendly, and if all the features were easily located. This test had the majority of responses that resulted in a yes. Only 6% of users found it not easy to navigate and 36% believed it to be satisfactory. Through this test, we learned that the key components to a website and one that is successful lie within the simplicity and the ability for the user to access all that the website has to offer.

Did you encounter any bugs

30 responses

Opening the website using a cellphone, the chat bot doesn't fit to screen. Other than that no issues

On mobile chat doesn't fit and directions does not work.

No bugs

speech does not turn off

image ouputed for one answer but not for another answer

the directional page doesnt load

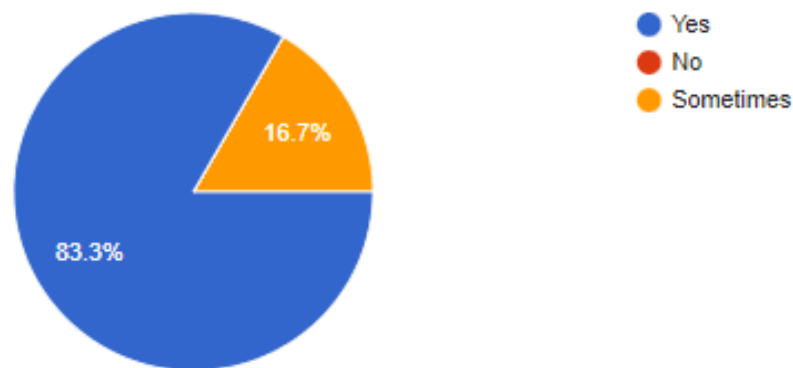
Unwanted users can authenticate with the google sign-in which would allow them to edit and change data

Doesn't work on mobile

The question that asked the users if they encountered any bugs gave a range of different answer from the users and so the team had to look through and use this information in order to better improve the website. The first bug that was encountered was the web page not fitting to the screen of the mobile device, and this was solved with the use of viewport that adjusted the size to the size of the user's screen. "The directional component was not working" was another bug as it was still in development preceding the delayed implementation, this was fixed as more resources were allocated towards the development. The speech button once clicked would not turn off until the text was over, this was fixed with the use of the toggle button the once clicked twice it would turn off. Unwanted user was able to login into the admin page, this was changed as the rules for accessing the database by writing to the database was restricted to certain users.

Does the chat bot respond with legible information?

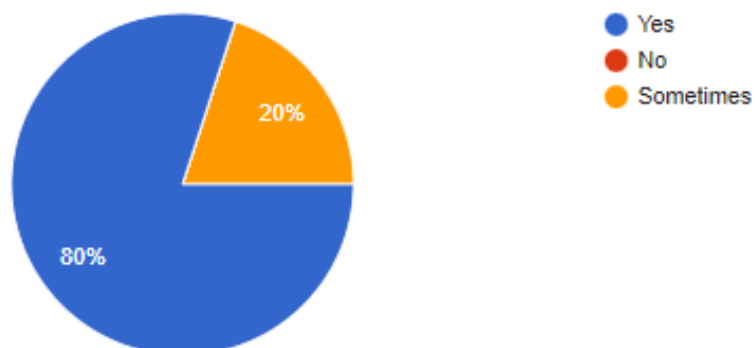
30 responses



The image above asks users if they received a satisfactory response from the chat bot after they had asked it a question. This question is significant in determining whether any bugs exist in the event where the chatbot outputs an incorrect, or insufficient answer to the user which is not understandable. If the chatbot were to output an unclear answer, we know bugs exist. Since 83.3% of users understood the response generated from the chatbot we know that it is doing its job in providing legible answers to the users.

Is the information provided by the chat bot relevant to the question?

30 responses



The question above was significant in understanding whether the output provided by the chatbot was relevant to the question that the user asked it. If the chatbot outputs an answer which has nothing to do with the question that was asked, it is an incorrect answer, meaning that bugs may exist. Since no user experienced any irrelevant answers, this was an issue that did not need to be further investigated. However, those that said sometimes, were provided with answers that was all the chatbot has knowledge of in the scope of its database related to the museum.

What would you improve?

30 responses

modern layout (buttons with icon rather than text)

site theme or colour maybe?

make an easier way to identify translation rather than asking at the beginning of every conversation

Have the chatbot be able to change topic easily

The home page could be better maybe more information about the museum?

have more museum information at the home page

Some user interface elements could be changed to make the site a little more appealing like resizing or layout

ability to access chat bot every where like a customer service chatbot

Make the UI more accessible

What would you improve?

30 responses

The screen fit issue as mentioned above

Navigation between the pages and some styling

Better colours/contrast

image sizes output when not needed

modern icon images, brighter colour scheme

Probably the design

the UI needs to be a little more colorful and vibrant

Spelling errors in the website, capitalization and some spacing issues.

the design and the full functionality of the webpage

What would you improve?

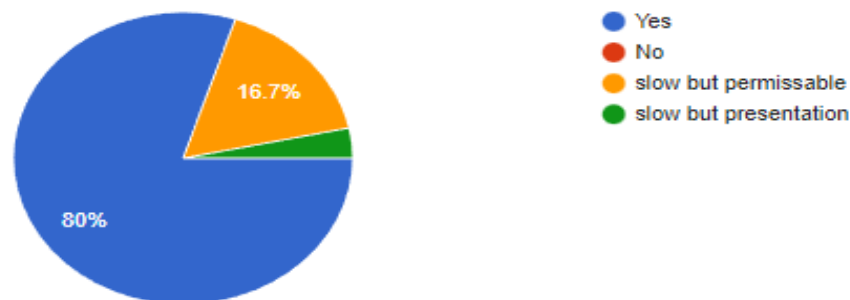
30 responses

Make the UI more appealing
remove the chatbot page it serves no purpose
The chatbot page seems kinda useless since it's available on other pages as well
Chatbot sometimes misinterprets my question. Sometimes difficult to figure out what is viewable and not viewable
More detail in the pages, it looks a little empty right now
make the interface user friendly
UI needs to be modernized for a younger audience. I would like to have seen a list of things that are available to get information on.
Fix on mobile so it can be used in the museum as well. Also image loading times.

The above images are specific ideas in order to improve the application. The biggest feedback received was related to front-end development as users wanted a user-friendly and appealing website. To create a modern layout, more icons with symbolism were incorporated rather than buttons with words or abbreviations. A change was established in the color scheme to navy blue and off white to provide a contrasting, consistent, and brighter layout. A bug was found in the image quality of some responses which was fixed by reducing image size, which ultimately improved its quality. A homepage was created with all significant museum related information with the ability to access the chatbot from all tabs in order to provide easy access to the chatbot. Moreover, several of the user responses were taken into consideration and improved on in the final version of the application. This will be shown below with images of the new system.

Were the loading times satisfactory?

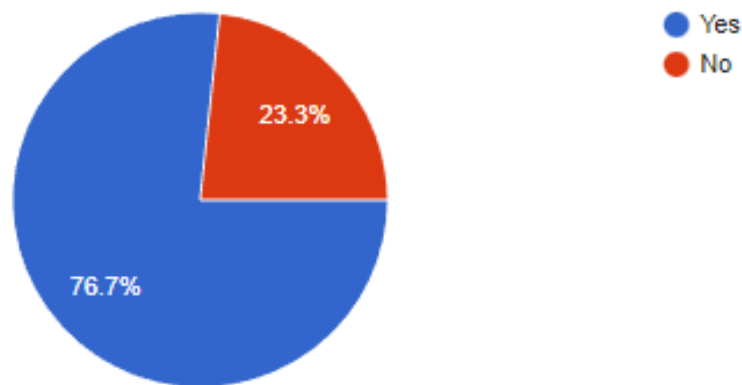
30 responses



The question above was for the use of the chatbot and the navigation of pages. The majority of users agreed that the loading times were satisfactory while 20% of the users said it was slow but permissible. Following this conducted test, we proceeded to research on a method to make the chatbot faster and through this research we discovered that the reason for the wait times were due to the API calls to OpenAI which unfortunately we had no control over.

Would you use a directional tool to navigate the museum using the website?

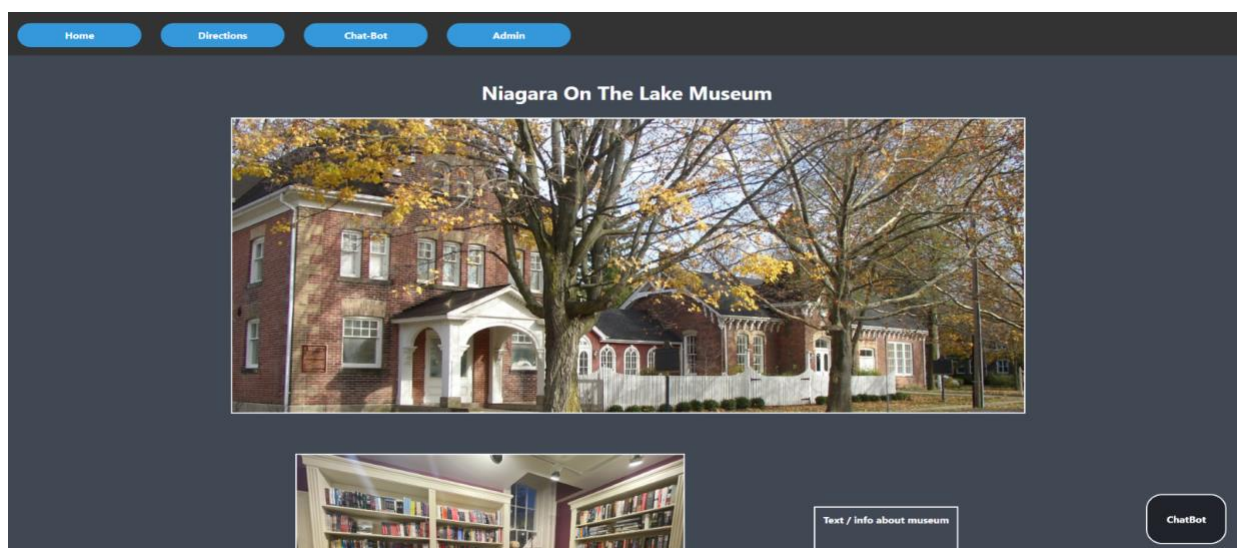
30 responses

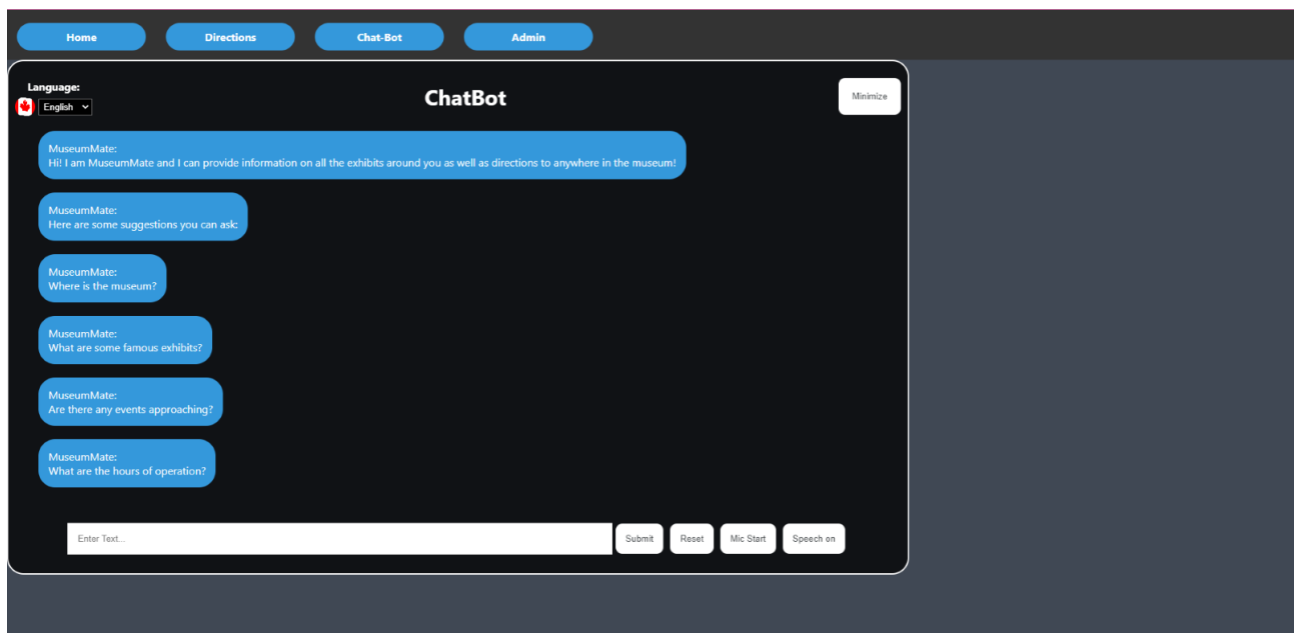
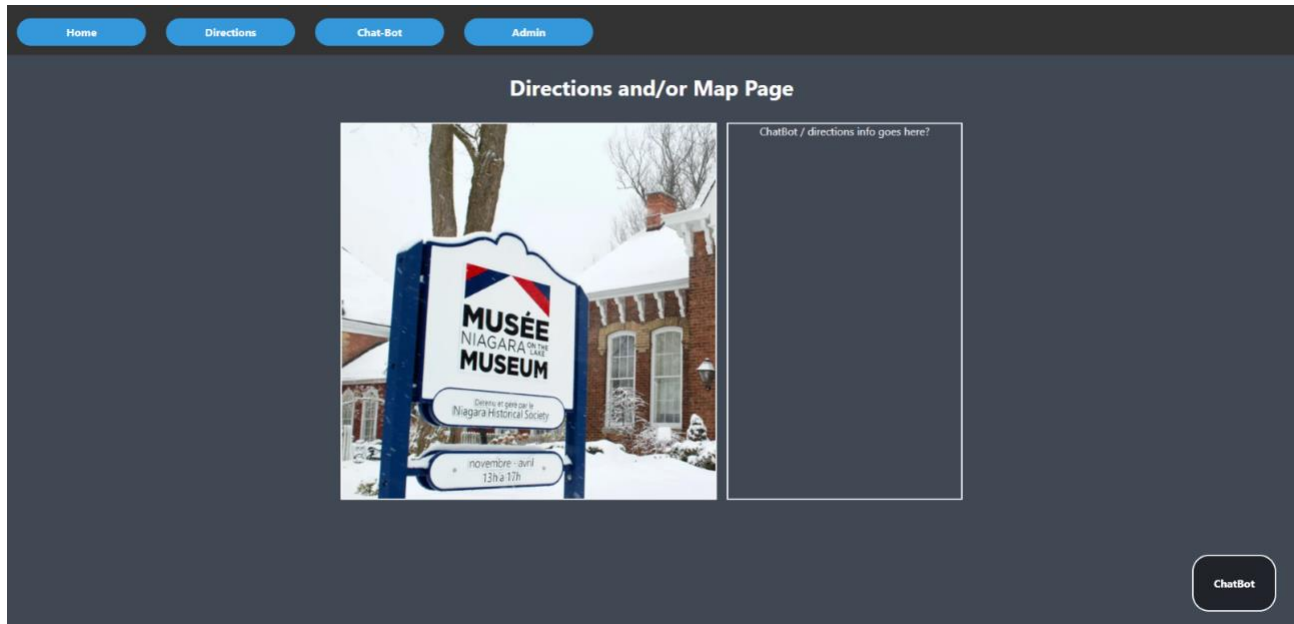


The question asked if a user would use a directional tool to navigate the museum using the website as its tour guide. Majority of the people that responded suggested that this was a good idea and so the directional component of the app included the ability for the chatbot give directions from one exhibit to another in human language (Go straight, then turn left)

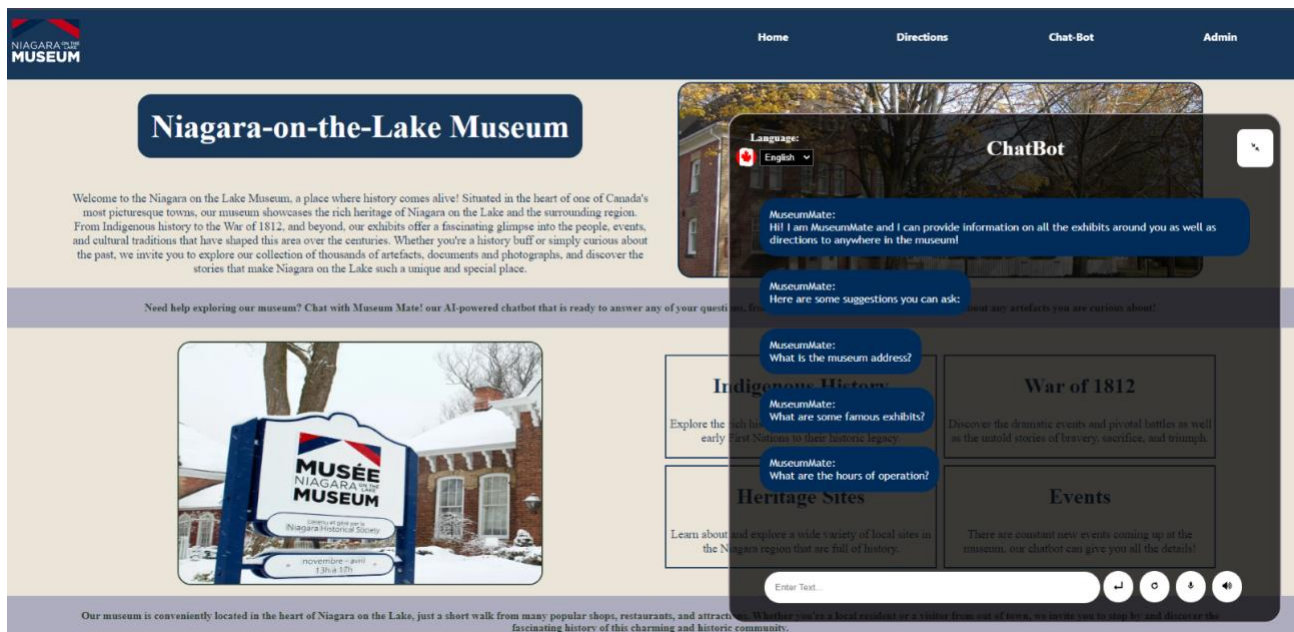
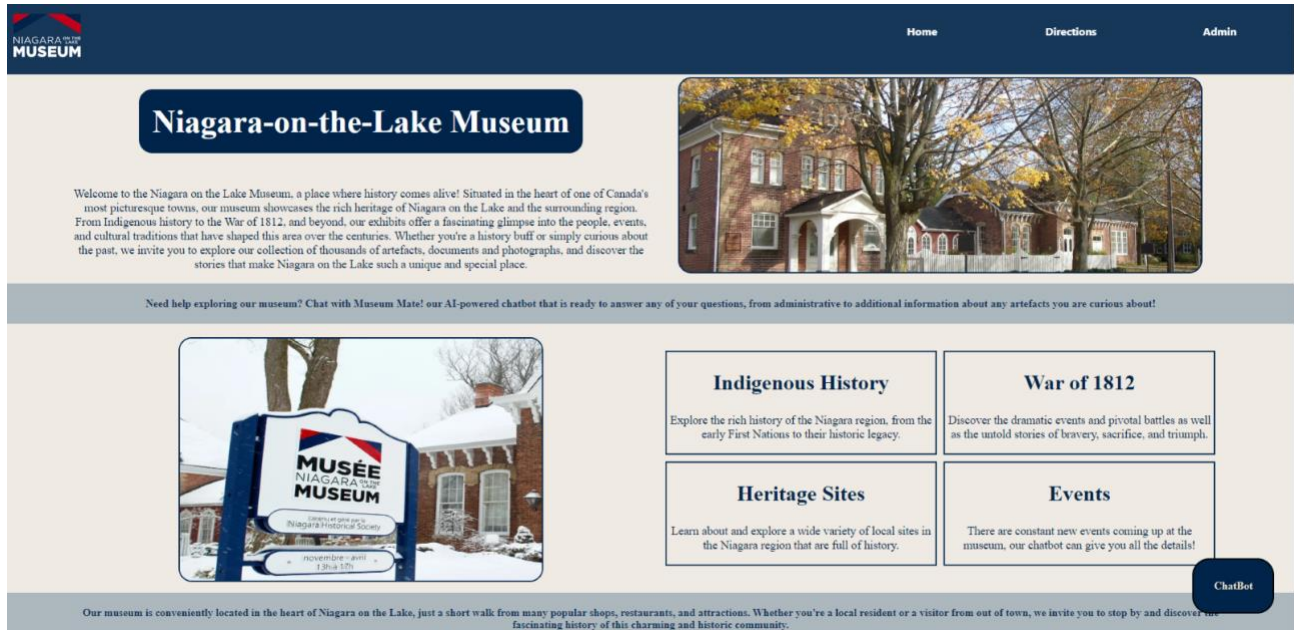
6.3 Improvements

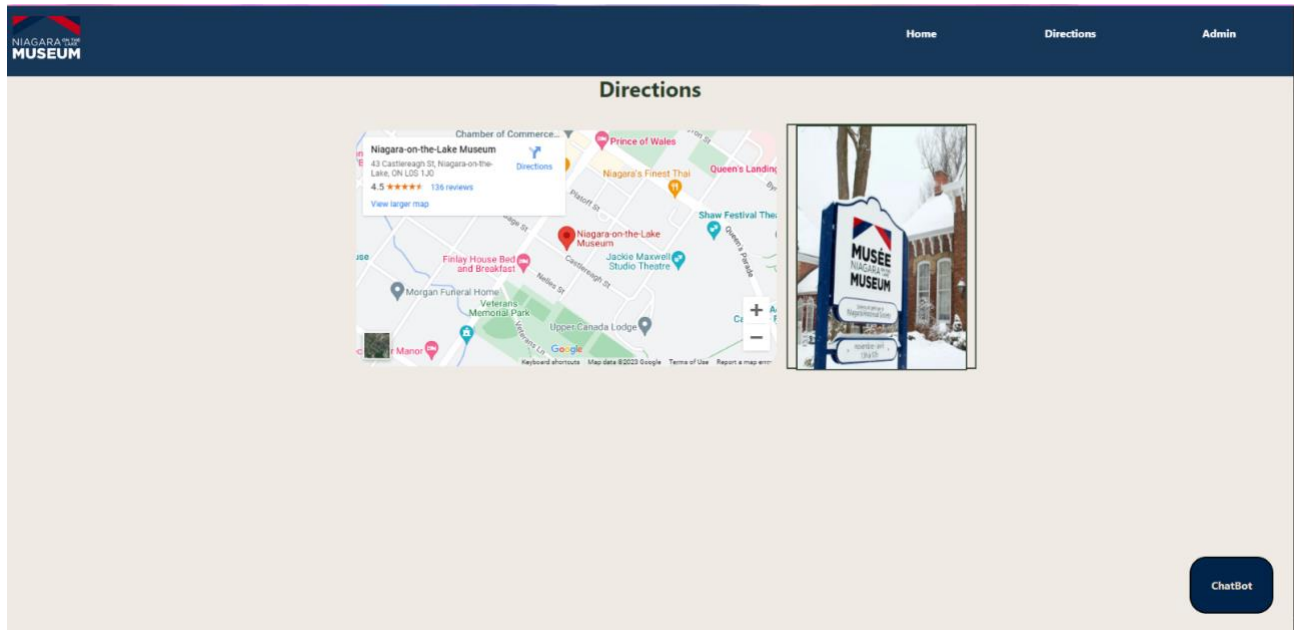
Improvements can be demonstrated from the user responses in the comparison between the old system and new system. The following are images of our previous system





The following are images that have improved our web application after testing:





7. Final Thoughts

The website is currently deployed. You can access it at:

<https://museum-mate.vercel.app/>