5-2022

An Investigation Into, and the Construction of, an Operable Windows Notifier

Grey Hixson

AN INVESTIGATION INTO, AND THE CONSTRUCTION OF, AN OPERABLE WINDOWS NOTIFIER

An Undergraduate Honors College Thesis

in the

Department of Computer Science and Computer Engineering
College of Engineering
University of Arkansas
Fayetteville, AR
April, 2022

by

Grey Hixson

Abstract

The Office of Sustainability at the University of Arkansas identified that building occupants that have control over operable windows may open them at inappropriate times. Windows opened in a building with a temperature and air differential leads to increased HVAC operating costs and building occupant discomfort. This led the Associate Vice Chancellor of Facilities at the University of Arkansas to propose the construction of a mobile application that a building occupant can use to make an informed decision before opening their window. I have formulated a series of research objectives in conjunction with the Director of the Office of Sustainability and my thesis advisor, to determine what precursive steps I would need to take before I could begin development of the application. I was able to resolve each prerequisite research objective with the assistance of HVAC staff, the Entegrity Energy Partners. Following the conclusion of the preliminary investigation of the mobile application, I started development of the project. I constructed a mobile-forward web application to meet the needs of the Vice Chancellor of Facilities. This web application supports user authentication, organization and space creation, user lookup of an organization and space, and a notification system all while maintaining a modern user interface, thoroughly secure backend services and adherence to the Web Content Accessibility Guidelines [30] to support a quality user experience. I had the intent to support the deployment of the application with the Office of Sustainability, but due to the considerable time required to resolve several administrative details I was unable to release the project with the Office of Sustainability. Another student or developer could resume this project and release it. They could then research and iteratively refine to uncover the fruitfulness of such an application.

THESIS DUPLICATION RELEASE

I hereby authorize the University of Arkansas Libraries to duplicate this thesis when needed for research and/or scholarship.

Agreed 
Grey Hixson

Refused_____
Grey Hixson

TABLE OF CONTENTS

LIST OF FIGURES

1 Introduction

In November 2021, I accepted a project offered to me from the Office of Sustainability. The project description stated that the Associate Vice Chancellor of Facilities at the University of Arkansas, wanted a mobile application to help people know when to open and close their office and/or classroom windows.

1.1 Motivation

Windows that allow for someone to open and close them, operable windows, can be found throughout many buildings and rooms across the University of Arkansas. The issue with operable windows is that the building occupants that control them may open them at times when they should remain closed. If the temperature outside is different from the temperature inside, then the HVAC system will become strained as it works to close the gap between the two different temperatures. This added load on the HVAC corresponds to higher energy costs, a shortened life for the HVAC system, and thermostat malfunction. Occupants in the same building would also experience discomfort as their area becomes warmer or cooler as the HVAC system attempts to reconcile the temperature change from the newly added air.

Unregulated temperature change is not the only concern. A sudden change in temperature would not go unnoticed, but there are other factors that would go mostly ignored like air pollution and pollen count. If the air quality is poor for a particular day or the pollen is exceptionally high, it may not be immediately obvious to someone who just opened their window thinking they were just letting in fresh air. Added pollen and pollution to the indoor air would not only cause the staff to replace the filters in the HVAC system more frequently, but it would also cause unnecessary harm to the other inhabitants.

Final apprehensions that the Office of Sustainability raised were security

and other irregular occurrences. A window that an occupant has opened and forgotten about could allow easy access to an otherwise secure building whether it is a building intruder or unwanted gases and particulates.

1.2 Background

An occupant that opens a window at an inappropriate time or forgets to close the window is a problem that is solvable in several ways, and this thesis seeks to investigate the construction of an application that helps users identify whether it is appropriate for them to open their window.

As of the writing of this paper, there is no known application, publicly or privately available, that allows for a user to register an organization, create spaces within the organization and set parameters for the space, and then for another user to search for a space within an organization and be informed whether the organization recommends that they may open their window or if it should remain closed. The closest known application that shares similarity would be any form of weather application, as a user would at least be able to view a weather app and then make a more informed decision whether they can open their window or if it should remain closed from current weather conditions.

1.3 Research Objectives

Creating an original application while keeping stakeholders' interest in mind requires extensive research and communication with clients to ensure that the project can be accurately scoped and completed before the conclusion of my time at the University of Arkansas. The Associate Vice Chancellor of Facilities at the University of Arkansas initially conceived the project, but initial meetings were set up with the director of the University of Arkansas Office of Sustainability. The director, my thesis advisor, and I held a meeting during the late fall of 2021, and over the course of the semester, we discussed what objectives I would need to answer before development work could take place.

The goal of the project is to develop a mobile application that can help people identify whether they can open the window for the space they are in or if it should remain closed. The set of research objectives that I devised are as follows:

1. Determine the target audience.
2. Develop a list of the parameters involved in the calculation of the appropriateness of opening a window.
3. Decide what features constitute a functioning application that solves the problem.
4. Explore how a user will interact with the application.

Following the conclusion of the above research objectives, I then proceeded to resolve the next set of development research objectives:

1. Determine the platform on which the Office of Sustainability will release the application.
2. Decide what frameworks, libraries, or other technology will be involved in the creation of the application.
3. Figure out how to best convey to a user if they can open their window or if it should remain closed.
4. Investigate the possible channels of communication with the user and implement the most appropriate ones.

After I sufficiently developed and evaluated all features of the application to ensure quality and precision, I completed a final set of research objectives.

1. Calculate estimated operating costs of the application.
2. Finalize all features of the application in conjunction with the requirements of stakeholders.

3. Evaluate the different options the Office of Sustainability has for deploying the application.

4. Devise a strategy so others can maintain the application following my graduation from the University of Arkansas.

5. Determine the ways that the Office of Sustainability could extend the application.

6. Decide the work that the Office of Sustainability must conduct before the application can go live.

7. Assist the Office of Sustainability with the deployment of the application.

2 Investigation of Preliminary Research Objectives

Over the course of my fall 2021 semester, I met on a mostly weekly basis with the Director of Sustainability, my thesis advisor, and various parties who have an interest in this project. In these meetings we discussed and contemplated several objectives that are necessary to adequately scope out the project while keeping the interests of the Office of Sustainability in mind.

2.1 Interested Parties

My primary contact with the Office of Sustainability was the Director of Sustainability, and he arranged for other various parties to meet with us so we could get their insight into what I should consider when determining the openability of a window. The two main parties involved were HVAC staff from the University of Arkansas and partners from Entegrity Energy Partners.

2.2 Parameters to Determine Window Openability

I met with the HVAC staff on December 3rd, 2021, to discuss what parameters I would need to consider when determining the openability of a window. During this meeting, staff from HVAC and Facilities Management at the University of Arkansas pointed me to three main parameters that I should absolutely consider: temperature, humidity, and air pollution. What they consider affecting a building's HVAC system the most is in line with the United States Environmental Protection Agency as the United States Environmental Protection Agency states "[the] main purposes of a Heating, Ventilation and Air-Conditioning (HVAC) system are to help maintain good indoor air quality through adequate ventilation with filtration and provide thermal comfort [1]." Then it followed that these three variables held the highest weight amongst potential factors I considered when I

wrote the logic to calculate the openability of a window.

2.3 Essential Features of the Application

During these regular meetings, another objective was to decide the features that would be necessary to meet the wants and needs of the interested parties, clients, and end users.

The core functionality includes the ability for a user to search for an organization with the organization's name, and then search for a specific space that their window is in with the name of their space. This space could either be generic or specific; an example for the name of a generic space could be "Research Lab" to denote all research labs in an organization; an example of a specific name could be "JBHT" representing the building code for the J.B. Hunt Transport Services Inc. Center for Academic Excellence at the University of Arkansas. A user would not have anything to search for unless there were organizations and spaces created, so additionally the application would need to support a way for a user to register an organization, create spaces within the organization, and then manage them accordingly.

Naturally, user authentication would follow the demands of functionality for creating and registering an organization since the organization should only be able to be manipulated by the user who created it. A final want for the core functionality came from the initial description of the project entailing a desire that a user can sign up for notifications for a specific space and then for the application to notify the user when they can open their window. Finally, since the Associate Vice Chancellor intended for a mobile deployment of the application as users will primarily interface with the application on mobile devices, depending on the application platform selected, the application will need to support a mobile view.

2.4 Application Platform and Frameworks

It was a given that the application will need support for mobile devices. This specification allows for me to either develop a web application with a supported mobile view or a dedicated mobile application. I decided to pursue developing a web application over a mobile application. A web application provides several benefits over a mobile application.

1. A user can interface with a web application on any device with a browser.
2. A user does not need to download or install a web application.
3. A user does not need to update a web application.
4. A web application would be cheaper as mobile apps have to pay app publication fees.
5. I will only need to develop a web application for one platform opposed to the many mobile platforms that exist. (Although I will need to develop for cross-browser compatibility)

Following the decision to pursue the development of a web application over a mobile application, I had to determine what, if any, frameworks, and technologies I would use to build the application. Although it is possible to build a web application with only HTML, CSS, and JavaScript, most developers tend to opt towards using a client-side JavaScript framework. JavaScript frameworks make modern web development faster, and easier. It would have been a near herculean task to develop all the previously listed features without a JavaScript framework, and the result would be worse.

A more formal definition of a framework comes from The Mozilla Developer Network as they state that:

> A framework is a library that offers opinions about how software gets built. These opinions allow for predictability and homogeneity in an

> application; predictability allows software to scale to an enormous size and still be maintainable; predictability and maintainability are essential for the health and longevity of software [2].

The Mozilla project, which the JavaScript creator Brendan Eich co-founded [3], advocates for the use of JavaScript frameworks listing them as "an essential part of modern front-end web development, providing developers with tried and tested tools for building scalable, interactive web applications [4]." Overall, there was no debate in my head regarding the use of a framework or not. The harder decision was determining which framework to use.

There are four standout JavaScript frameworks that any developer could choose from: Ember, Angular, Vue, and React. None of the options would have been an unfitting choice for a project of this scale, but I settled with Vue primarily because it was the only one that I had experience with since I use it every day as a part of my employment as a software developer, and because I really enjoy working with it. Vue also pairs with Vuetify, a material design framework made specifically for Vue.js. I was able to use a JavaScript framework to build the application faster, and in a similar vein I was able to use Vuetify so that I could build the application's user interface and deliver a quality user experience with less effort than doing it with just CSS. Vuetify is a Vue UI Library [5] that contains custom Material Components. Having these Material Components on hand allowed for me to develop the application extraordinarily faster overall. The final piece of technology that I leveraged is Google Firebase. Firebase is a platform that offers extensive computing and development tools. The primary service they offer that I took advantage of is their fully managed backend infrastructure. I had to store the organizations, spaces, account settings, and notifications that users create somewhere in addition to the API keys that I used. Google Firebase offers a service called Cloud Firestore which is a "Serverless document database that effortlessly scales to meet any demand, with no maintenance [6]." Managing the backend is no trivial task, and so having Google manage it for me accelerated development even further. With the frameworks chosen, I then had to determine the service I would

use to manage the dependencies the frameworks would become for my project, and the dependencies the frameworks use.

To ensure that the dependencies that are involved in my web application remain updated and exist at a compatible version that their parent expects them to be, I used the node package manager. Node package manager "...is the package manager for Node.js. Node.js was created in 2009 as an open-source project to help JavaScript developers easily share packaged modules of code [7]." Vue, Vuetify, and Google Firebase all exist as node modules, so with the node package manager I can bundle them all into my web application to maintain them alongside any other dependencies I might add.

Altogether the three pieces of technology that I decided on to develop the application were Vue.js, a modern JavaScript framework, Vuetify, a material design framework built for Vue.js, and Google Firebase, a platform with an array of computing and development tools. This technology stack I had selected enabled me to build this application many times faster, and better than if I had opted to forgo dependencies entirely, so much so in fact, that it would have been impossible to develop it without them given my limited period.

At this stage I had completed all preliminary research objectives and I was ready to begin development work. This would also see a transition to my meetings with the clients and advisor focus around demoing the current state of the application. These demos would allow me to continuously hone and refine the app while ensuring I maintain the original idea for the app as it was first envisioned.

3 Development of the Application

Every developer has a different philosophy regarding the correct approach and order to develop an application. There is not one singular methodology that a developer can utilize for every project, but my personal philosophy is to prioritize writing the absolute best logic that I can for features that I am certain will persist throughout the life of the project. This makes me a better developer, makes it easier for others to work on the project, and it keeps my code clean, readable, and extensible. If I am not certain a feature will persist, I will rapidly prototype it so that I can fully grasp what the user experience will be, and if it is not satisfactory, change it. This strikes what I find to be a balance between speed and quality, even though a developer should always rewrite a rapidly prototyped feature, as a developer should always conduct writing code to the best of their ability if time allows it.

To develop my web application, I used Visual Studio Code in the beginning of the development, but I later switched to the fully-fledged JavaScript IDE JetBrains's WebStorm. I managed all the dependencies for my project with node.js and maintained a GitHub repository to track versions over time, and to keep backup copies of the program. Finally, I used ESLint which was setup to enforce Airbnb style guides to ensure that the code I write is not just valid JavaScript, but JavaScript that adheres to a professional set of standards.

3.1 User Interface

I constructed the user interface with Vuetify. Vuetify provides developers with all the tools they need to design and build user interfaces. These tools include custom Material Components that range from basic buttons to data tables, and styling and animation support. These custom components and styles allowed me to rapidly prototype different layouts to assess the user experience and to create a

modern user interface. Due to the time constraints of the project, I kept the theme of the Vuetify components mostly unchanged and stuck to their default theme. The only color I paid a conscious mind to change was the toolbar that displays the title of the web application to be the University of Arkansas's Razorback red paying homage to the University I attend.

Figure 3.1: Navigation Bar

Although I did not dedicate much time to the styling of the theme, I did spend considerable time on the layout.

Vuetify supplies developers with several tools for managing the layout of a page. The intent for the web application is for users to primarily use it on mobile devices, so I made sure to develop it with that intention in mind. I kept the sides of the pages clear and focused all content to the center of the screen. I also added the necessary logic to resize components and change the layouts depending on the size of the mobile device. To develop for mobile devices, I referenced the viewports of the smallest iPhone, which has a viewport size of 320 x 568 (Apple iPhone 5) and the largest iPhone, which has a viewport size of 428 x 926 (Apple iPhone 12 Pro Max) [8].

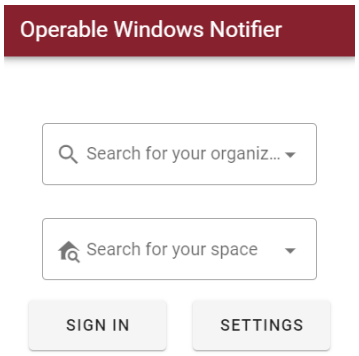

Figure 3.2: Pre-display 320 x 568 viewport

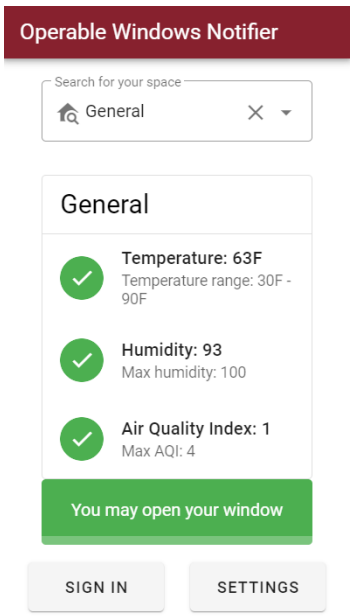

Figure 3.3: Post-display 320 x 568 viewport

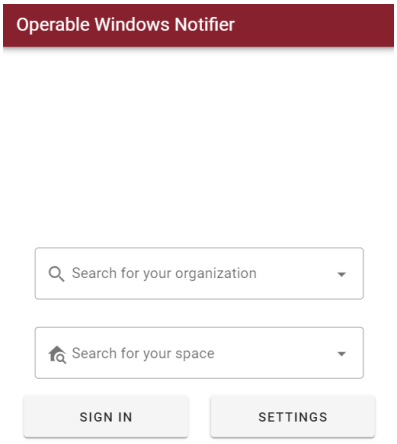

Figure 3.4: Pre-display 428 x 926 viewport

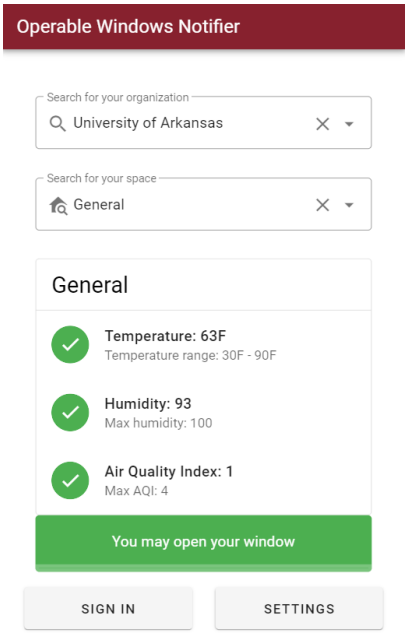

Figure 3.5: Post-display 428 x 926 viewport

Although this application was developed with a mobile viewport in mind, viewports were setup with all device sizes in mind ranging from an iPhone SE to a full-sized desktop monitor.

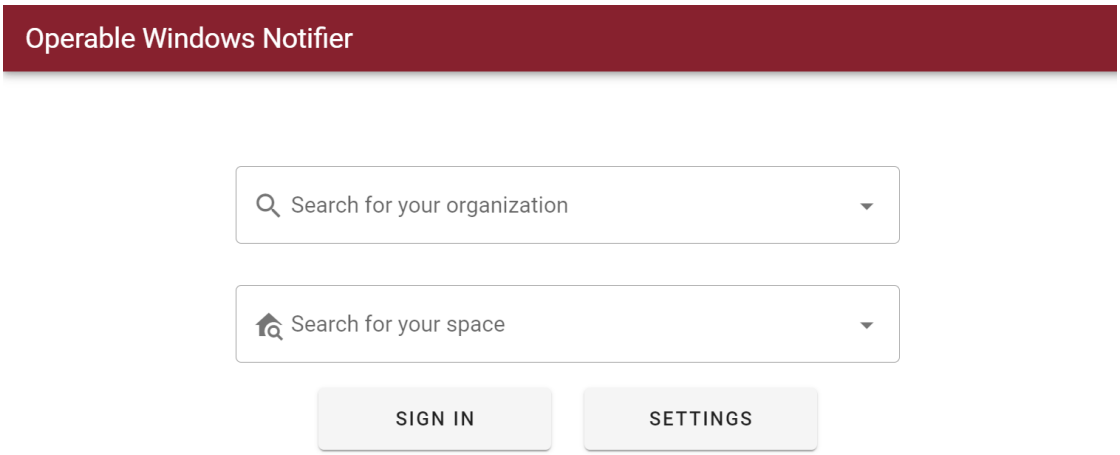

Figure 3.6: Pre-display desktop viewport

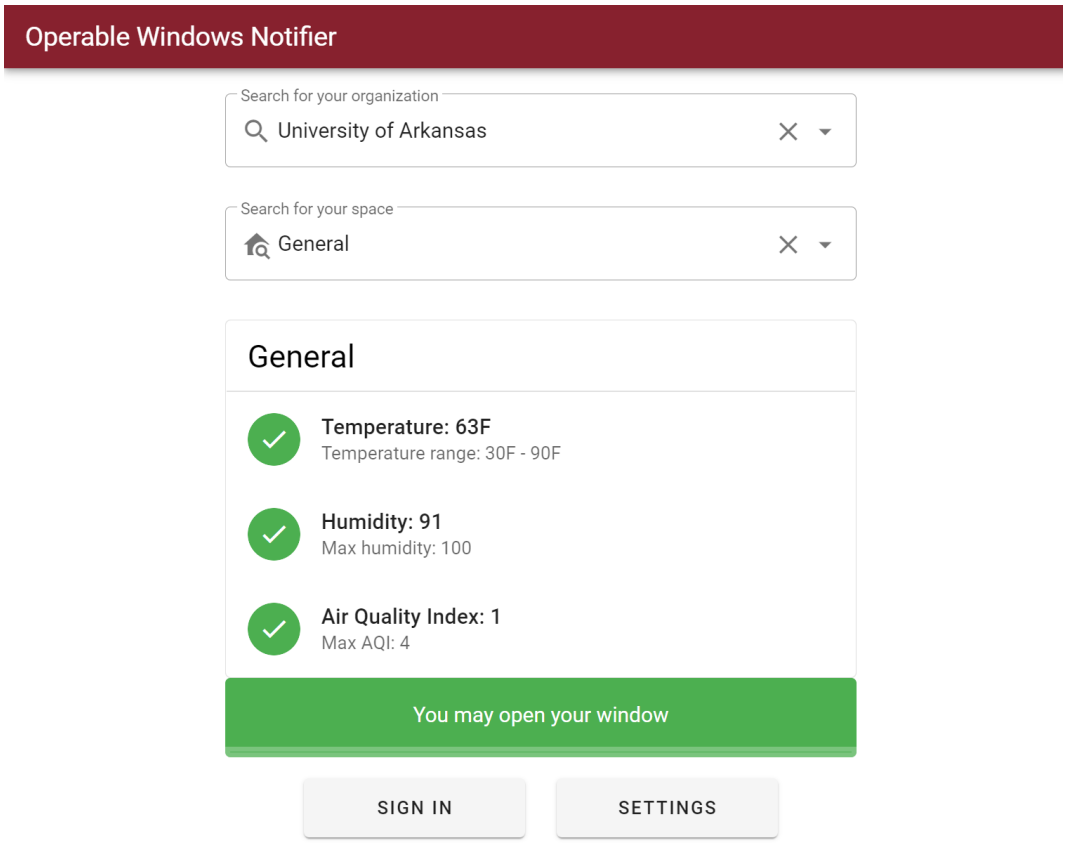

Figure 3.7: Post-display desktop viewport

3.1.1 Future Work

I was able to successfully implement views for functionality across the entire span of viewports, but users that have screen sizes on the smallest end are not able to interface with the application as easily as I should have made it for them. A future developer will need to add the necessary logic to dynamically scale the font-size down at a certain viewport size to improve their experience. Additionally, following the deployment of the application I will need to gather feedback to iteratively improve the design.

3.2 User Experience

Peter Morville, a pioneer in the fields of information architecture and user experience, devised the user experience honeycomb as a way "... to illustrate the facets of user experience [9]."

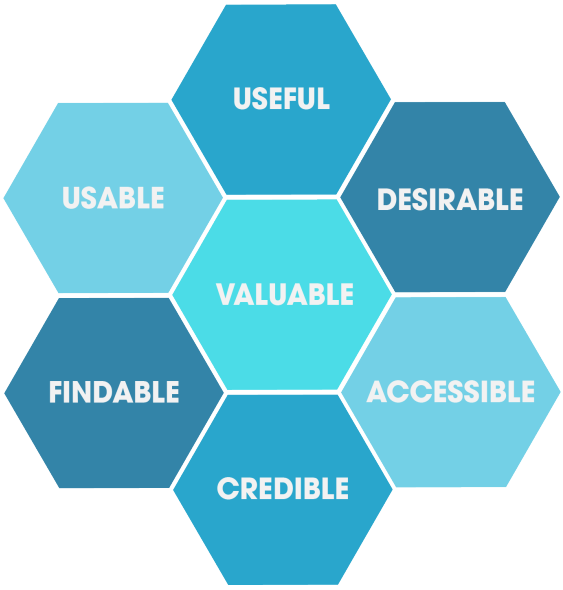

Figure 3.8: The User Experience Honeycomb

Throughout the development of this application, I paid mind to these seven user experience hexagons.

3.2.1 Usable, Findable, and Accessible

When surveying the usability of my application, I referenced Stanford's usability principles [10]. The list of principles are as follows:

1. **User Control:** The application will not automatically redirect users except when they attempt to enter a page that they do not have access to (e.g., going to the manage organization view when they do not have an organization

to manage). Users will only advance to a different page by selecting the appropriate navigation button.

2. **Recognition vs. Recall:** One step I took to lighten the load on users' short-term memory is giving users the ability to save their search selection so they will not have to recall the exact name of the space and/or organization they are in. This feature does not require recall except for authentication.

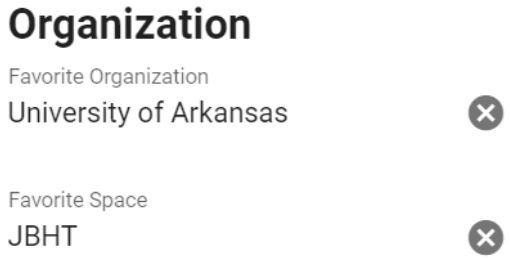

Figure 3.9: A saved organization and space

3. **Mental Model:** It is important that I model a user's window the same way that they do. Therefore, I have a user search for their organization and then the space they are in, opposed to having a confusing and large list of all potential spaces they could select.

4. **Clarity:** I attempted to use the simplest language possible to convey meaning to my users. For example, navigation buttons use one to two words (e.g., "settings," "sign out," "reset password"). Additionally, I have hints that appear around input fields, so the user knows what the field is expecting for them to enter.

5. **Aesthetic Integrity:** Due to the reusability of Vuetify Material Components I had a straightforward way to maintain the integrity of the aesthetics of my application. I kept the view as clutter-free as possible and sustained a centered view of components across all pages of the application.

6. **Simplicity:** As mentioned in Clarity and Aesthetic Integrity, my UI is clutter-free and is laconic with communication.

7. **Predictability:** All buttons perform the action the user expects, with prompts before major events like deleting an account. This ensures a user is entirely aware of what will occur because of an action they took.

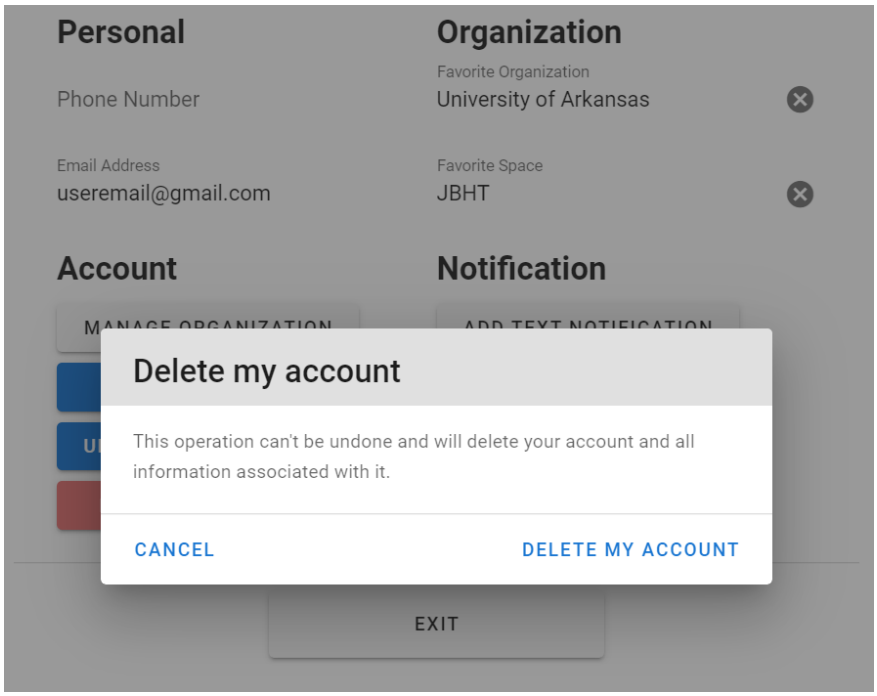

Figure 3.10: Delete account dialog

8. **Interpretation:** My browser and password manager can correctly identify my login form and autofill my information. The interface is then aware of what the user is attempting to do.

9. **Accuracy:** I used type inputs throughout forms and text fields to prevent form-breaking entries such as emojis in a field for updating a phone number.

10. **Error Handling & Prevention:** I wrote custom rules for all forms in the application so that a user cannot input something that form is not expecting

or does not allow, and to provide feedback to a user on why their input is incorrect. If all fields in a form are not valid, the application will make a user aware of the invalid fields and prompt them to correct it so that the user can correct the form and submit it.

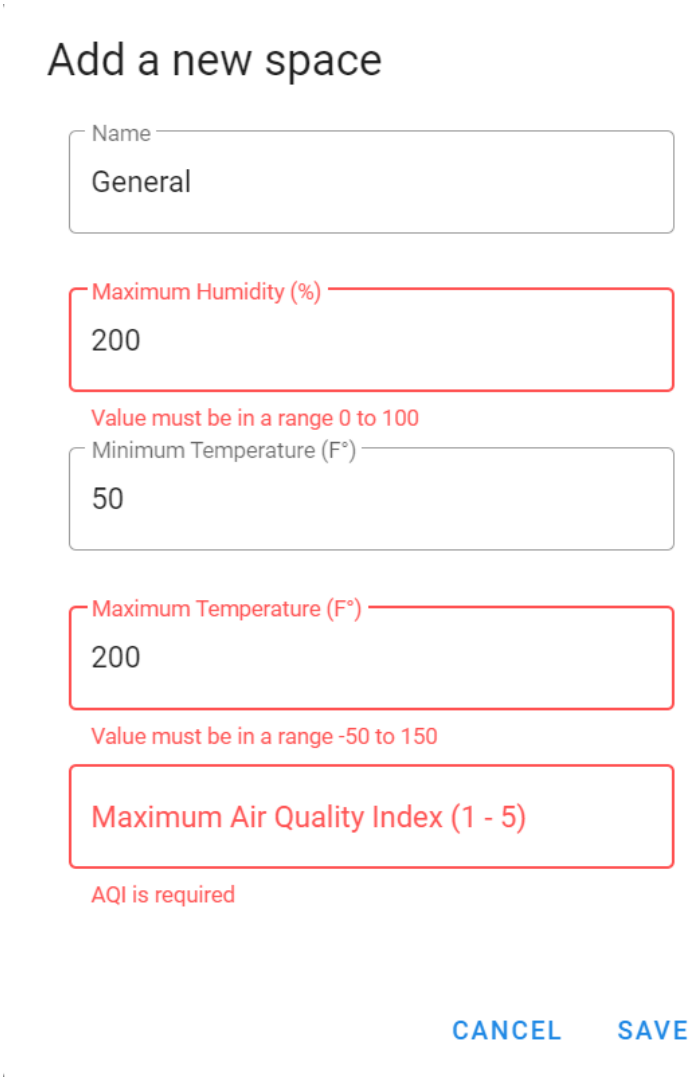

Figure 3.11: An invalid new space form

11. **Customization:** I minimized the complexity and distinct types of information, so there was little utility of customization that seemed appropriate or

worth the development time to spend implementing.

12. **Shortcuts:** Users can navigate back to the home screen from every page.

13. **Consistency:** The toolbar which displays the title of the page is in a consistent fixed location across all pages.

14. **User Support:** It is up to the discretion of my client how they would like to manage user support.

15. **Precision:** Users do not have to rely on workarounds or inefficient means to complete actions on the application. They can use the application in the way it expects and get an expected and appropriate output each time.

16. **Forgiveness:** I created several features to let the application be forgivable. A few of them include:

    - Delete and readd a mistakenly created notification.
    - Delete and recreate a misspelled or mislocated organization.
    - Update parameters for a space if it should be changed.

17. **Feedback:** On every page in the application, I created a custom component which contains an alert banner that the application uses throughout to display error messages if the application fails to complete an action, and a success message if the application succeeds in completing an action. If the application is unable to complete an action, I created an indicator to display to the user what is missing before a user can complete the form.

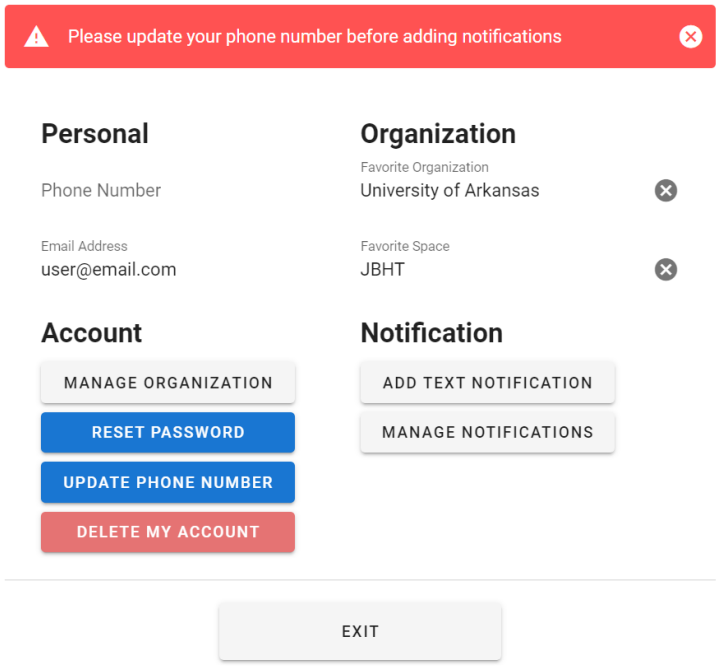

Figure 3.12: Example error banner

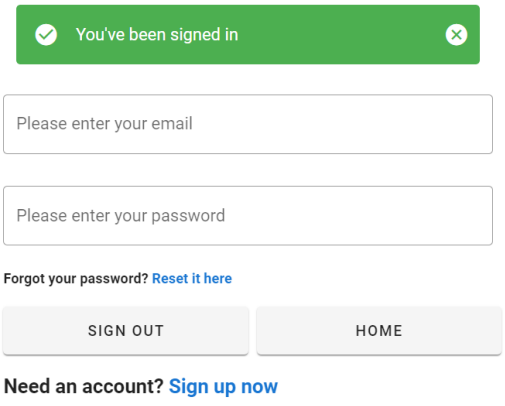

Figure 3.13: Example success banner

18. **Accessibility Compliant:** See Section 3.7 Accessibility.

3.2.2 Valuable, Desirable, Credible, and Useful

My application's success in alleviating the problems introduced in Section 1.1 Motivation is what will determine its value. Following the deployment of my application, the Office of Sustainability will ideally see the problems become resolved or made better, making my application valuable to them and to the end users. The deployment of my application has yet to be carried out, so this section has yet to be answered.

I kept the design of the application minimal, and straightforward. A visually aesthetic design for a website enhances its desirability for a user. Furthermore, I had limited development time to spend on the creation of this application, so I focused on desirable features. All features that I added were with the intent that it is a feature that a user will want to use and will use like notifications and saving of their search selection.

The credibility of my application immediately fails if a user checks to see if they can open their window, and upon seeing they should keep their window closed, notices the temperature listed is not at all what is in the area for their window. A user would then question the credibility of my application if it cannot do something as essential as finding accurate weather data. This credibility also extends to other areas of the application that do not work as expected. If application features fail, then the application will lose credibility with its users. To maintain credibility, I made sure to use an API for gathering weather information from a reputable source, and I performed extensive quality assurance testing of the application to guarantee that features will not fail, and if they do fail, to let a user know why.

There are two parties I must convince on the utility of my application. Building occupants who use my application to determine the openability of their window, and the Office of Sustainability. Throughout the meetings with my client, we determined a set of essential features as I described in Section 2.3 Essential

Features of the Application. Because all the features that my application includes constitute the initial project description, it is then up to the discretion of my client and end users to determine the usefulness of my application.

3.3 Account Management

3.3.1 Overview

A user can create an account by filling out the sign-up form with their email address, a password, confirming their password, and then submitting their password. If the user successfully submits the form, the application will create an account for the user, and then sign in the user.

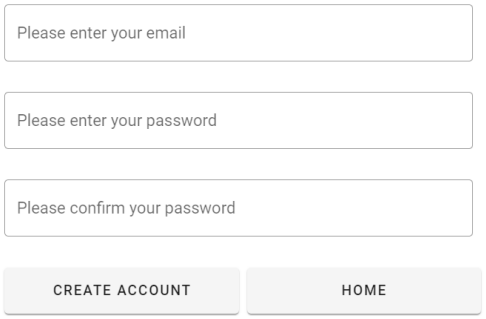

Figure 3.14: Account creation form

After a user creates an account and the application signs them in, or they sign in manually, they can access their settings and gain access to all the features for an authenticated user. These features include:

- Search preference save
- Autoloading of search preference
- Notification management
- Organization management

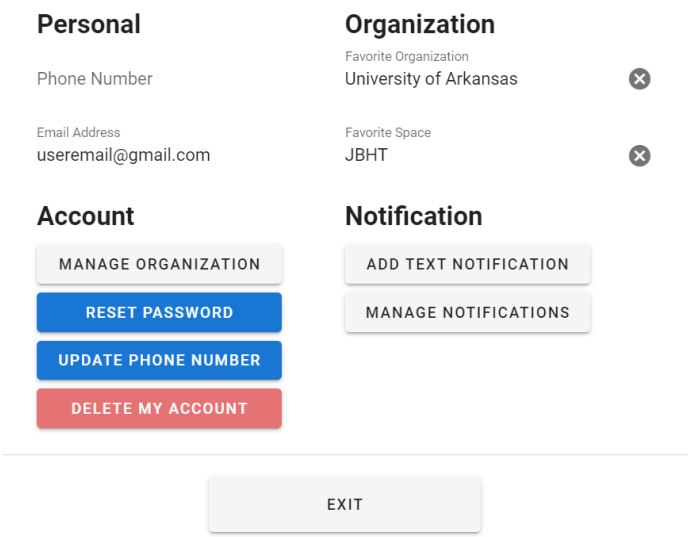

Figure 3.15: A user's settings

An authenticated user can also update their personal information and account settings. This includes:

- Resetting their password if they are signed in or signed out.
- Updating their phone number.
- Clearing their search preferences.
- Deleting their account and all information associated with it.

These features enable a user to eliminate their digital footprint if they so wish. Finally, if a user loses their authentication; they can sign back in by reentering their email and password.

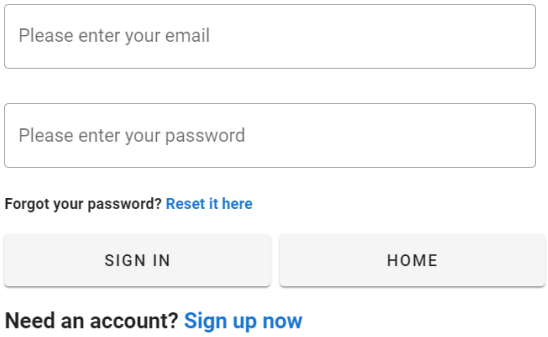

Figure 3.16: Sign in screen

3.3.2 Implementation

Account Authentication

Account management requires account authentication as well. Account authentication cannot be managed client-side and requires the use of backend services. The backend services that I used come from Google Firebase Authentication [11]. Google Firebase Authentication has extensive documentation regarding how to set up user authentication. The user authentication that I opted for is a standard email and password authentication. Standard email and password authentication strikes a balance between security, effectiveness, and ease of use. A user was only able to manage their account with email and password authentication after I had implemented the five functions listed below. This enables a user to manage their account without intervention from an admin. A user at minimum needs to be able to:

- Sign up for an account with an email and password.
- Sign into their account with their email and password.
- Reset their password if they are signed out.
- Reset their password if they are signed in.

- Delete their account.

I implemented these functions according to the documentation that Google provides for Web version 9 [12]. The only change that I made was to resolve promises with async/await instead of with promise chains. The application handles errors with try catch blocks and discards error messages and codes instead of displaying them to the user due to the potential for abuse. The only exception to this rule is during account creation, as a user may be unaware that they had already created an account so the application would need to inform them that an account already exists with their email.

Account Settings

Whenever a user creates an account, the application creates a document for them in the backend database, Cloud Firestore. This document contains all the user's settings which they can view and update which include the following:

- Organization and favorite space
- Registered organization name if they have one registered
- Phone number
- Email address

These settings are all tied to a user's authentication, so when they sign in, they will be able to view and change them. A user's authentication ties the user's settings to the user, so that adversaries are unable to read or update a document containing this information securing the user's data.

3.3.3 Future work

I have implemented all the essential tools for user authentication and the functionality that comes with it. At this stage, a future developer could add

functionality to enhance a user's experience. They could add different types of authentications such as phone number authentication. A user's phone number would then provide two utilities as it could serve as authentication in addition to receiving notification text messages. To circumvent a user needing to create an account at all, the application could use federated identity providers like Google Sign-in and Microsoft Login as OAuth providers. This takes the weight off the application owner to manage user authentication. Also, because the intent of this application is for its deployment to be at the University of Arkansas, and the school already uses Microsoft as their OAuth provider, any faculty member, staff member or student would already have an accepted login.

3.4 Notifications

3.4.1 Overview

One of the specifications for this project was to add functionality for notifications. Users can add notifications and then receive information for the window in the space they have signed up to receive notifications in. Initially the client and I considered email and text notifications. I abandoned email notifications soon into the project as I came to the realization that many of us have inboxes that are already too full, and a consistent stream of emails about our windows would only serve to worsen the problem. Additionally, because of limited time it did not seem like a worthwhile investment.

I carried out the implementation of text notifications. A user can go in and add as many text notifications as they like. A text notification has six parameters that are all required before a user can add one.

1. **Organization:** This is the name of the organization the space the user is searching for is a part of (e.g., University of Arkansas - Fayetteville)

2. **Space:** This is the space the user's window is in (e.g., Mechanical Engineering Research Lab)

3. **Notification send time:** This is the time the application calculates and sends the openability of a window to the user (e.g., 1:00pm).

4. **Repeated days:** These are the days the notification will repeat throughout a week (e.g., Monday, Wednesday, Friday).

5. **Start date:** The date the notifications will begin to send (e.g., January 2nd, 2022).

6. **End date:** The date the notifications will stop sending (e.g., May 17th, 2022). The application will automatically remove expired notifications from the database.

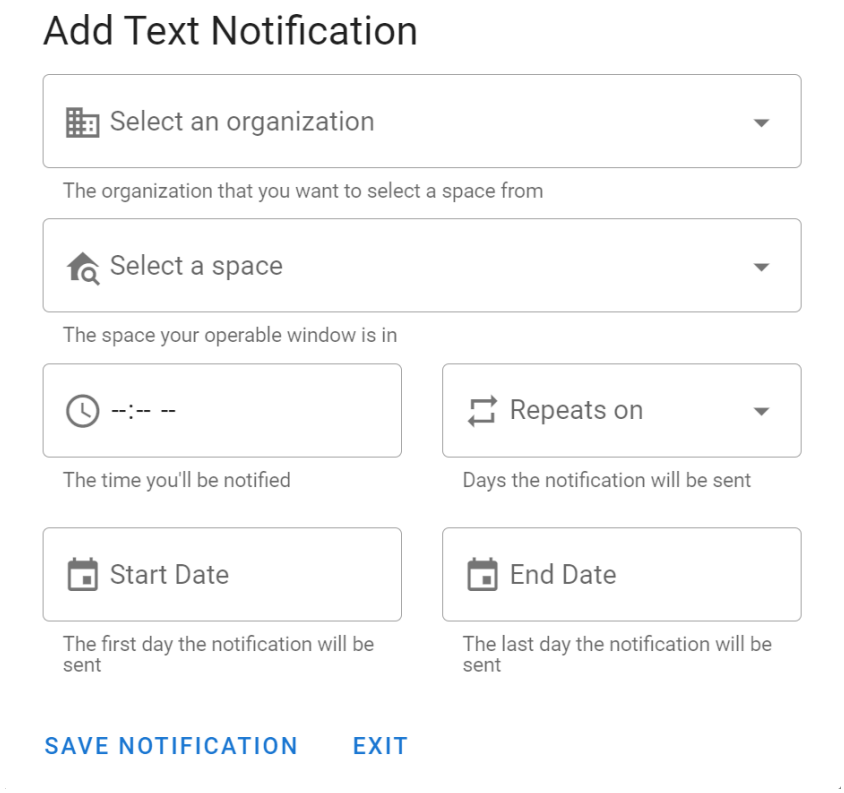

Figure 3.17: Add text notification

Finally, users can manage their notifications which allows them to view a list of all of their notifications and either enable a notification, disable a notification, or delete a notification.

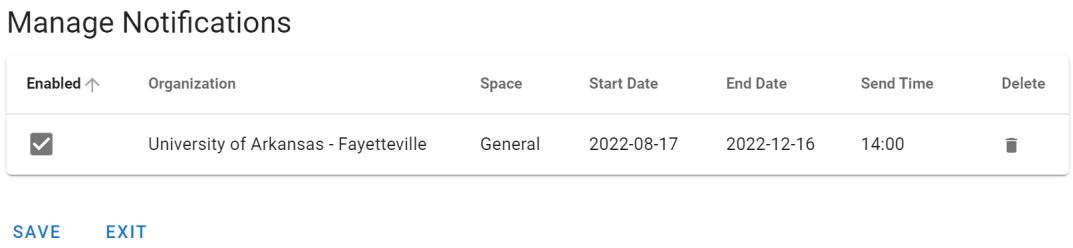

Figure 3.18: Manage notifications

3.4.2 Implementation

A notification system was the most complex piece I had to develop for this project. An application takes a notification through a long pipeline for it to go from creation to user delivery. Users create and manage their own notifications. Each user has their own notification document that stores their notifications.

The server executes a cron job every minute using Cloud Functions to check all the notifications and then sends the ones that the server has deemed ready to send [13]. One check in the series evaluates if the current date is past the notification's end date. If the notification has expired, then the server will remove it from a user's notifications document. If the notification passes a series of checks the server conducts to see if it should send it or not, it will be queued and sent. MessageBird is the API I implemented to send notifications and Firebase extensions provided the functionality for me to include it in my project in a simple manner [14]. This extension will create a notification message in a collection of documents and provides information surrounding the sent text.

3.4.3 Future work

Currently text messages are the only way to send notifications. The United States of America is where the deployment of the application will take place, which has extremely strict guidelines for SMS with many restrictions in place [15]. These restrictions and the recent 10DLC changes [16] make it so that I am unable to set up a functioning phone number that can be handed off to my client. Instead, I will have to collaborate with my client to register a phone number under their organization or use a number they have already approved for A2P SMS messaging. I knew this would be the case, so throughout development I planned for it as my current system of notifications that I have created is easily extensible to include an API for SMS messaging. This extensibility goes not just for SMS messaging but also email, and push notifications should my client decide to go that route.

Prior to the development process, one feature that we had discussed was giving admins the ability to send a notification to all users to prompt them to close their window for a given organization and space. A reason an admin may want to do this is because the window would need to keep irregular occurrences of a variety of events both natural and unnatural such as gas leaks, unmanaged waste, animals, intruders, and more outside. I did not implement this during the development process due to time constraints and because the decision of an internal or external deployment would heavily affect how I could implement the feature.

3.5 Organization and Space Management

3.5.1 Overview

A core feature to this application is the ability for a user to search for their organization and spaces associated with the organization. The creation, and management of spaces and organizations must be a feature accessible to users but can only function if organizations exist. This web application allows a user to first register an organization if they do not already have one. Once a user has successfully

registered an organization, they can go into the manage organization view and create, update, and delete spaces in addition to deleting the organization. I put into practice the parameters from Section 2.2 Parameters to Determine Window Openability when creating the feature for a user to add a space. An organization owner can set the maximum humidity, minimum temperature, maximum temperature, and maximum air quality index [17] for a space that the application then uses to calculate the openability for windows in that space.

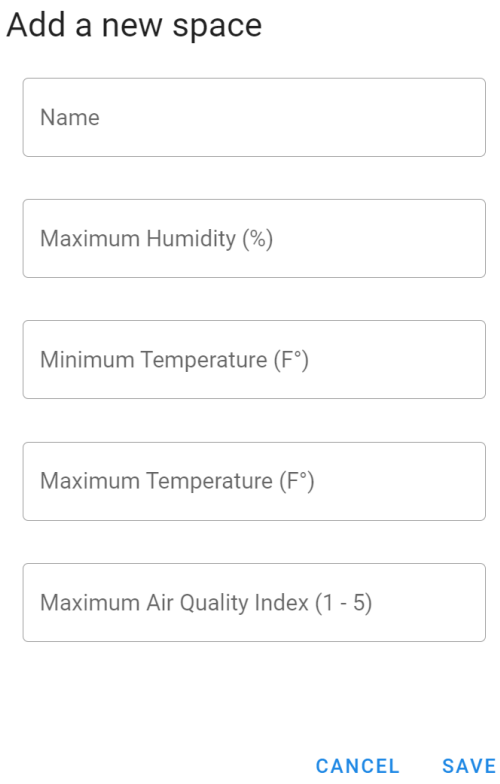

Figure 3.19: New space creation form

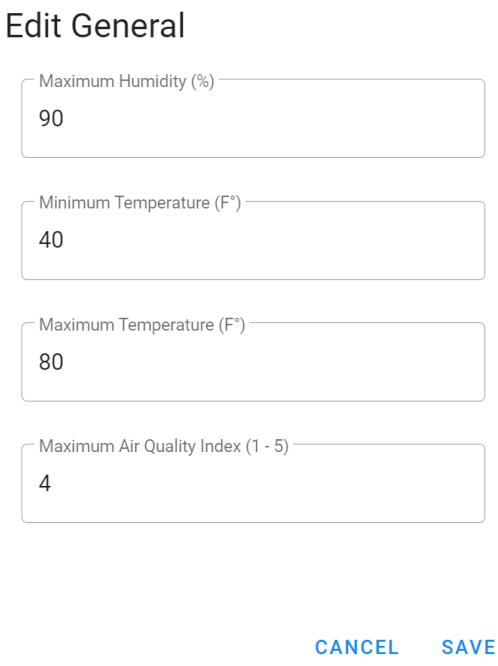

Figure 3.20: Space editing view

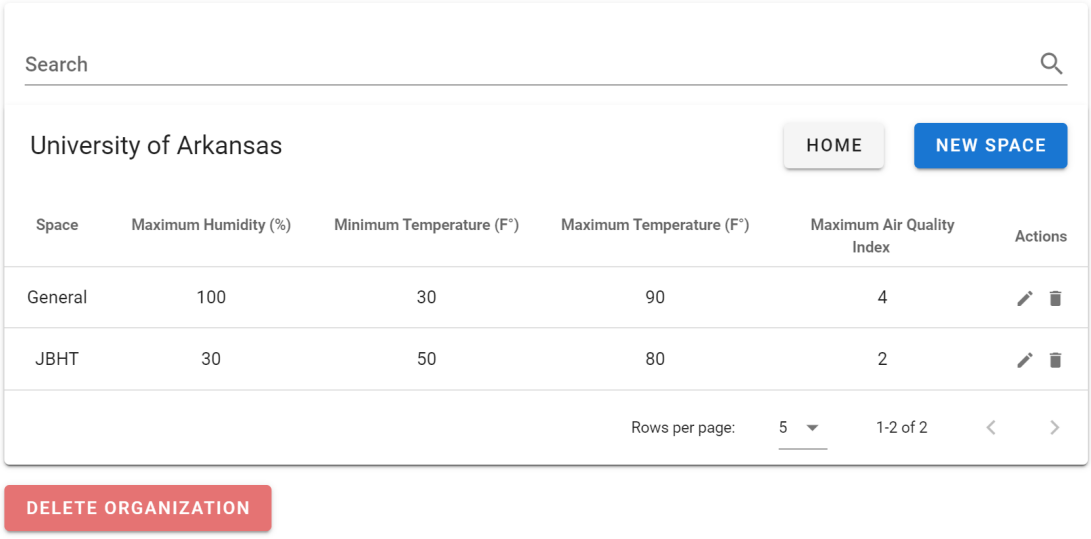

Figure 3.21: Space and organization manager

Once a user registers an organization any user can search for the organization and select it. A user can then select any space associated with the organization to view the openability of windows in that space.

3.5.2 Implementation

The application will prompt a user to fill out a form after a user creates an account and goes to register an organization. This form requires that the user provides an organization name, city, and state. When the user submits the form, the application will check to see if it already exists. If it does not exist, then the application will silently call the API that I use for my weather services OpenWeatherMap. If OpenWeatherMap returns weather data successfully, then the application will create the organization for the user as a document with the organization's name, city, and state as key value pairs and stored in the Cloud Firestore. Finally, to reflect a user's ownership of an organization the application will update the user's document.

Once a user successfully creates an organization, the user is then able to go into a separate view to manage it. A user can route themselves to the manage organization view in their settings as the button for "register organization" changes to "manage organization" if a user owns an organization. This application does this when the user first enters the settings page by using Google Firebase's functions for getting a user's authentication and checking if the state of it changed [18]. The manage organization view checks a user's authentication to ensure that they are the true owner of the organization when the owner created the organization, and if the user's authentication changes, checks to see that the user still owns the organization. The application will kick out a user from the managing an organization view if they do not own it. This is handled the same way as it is inside of the settings view. Inside of the manage organization view, the owner for the organization can update, add, or delete their spaces which the application will save in Cloud Firestore within a subcollection of the organization document.

3.5.3 Future work

The functionality for registering, and managing organizations is fully complete, as is the functionality for creating, updating, and deleting spaces. There are still features that a developer could implement that would provide utility to users and admins alike. The initial specification for this application includes a mobile friendly view for users. I have created a mobile friendly view for users to search for their window, but the view for admins to manage their organizations is functional but a developer could update it to provide for a better experience.

3.6 Security

An article by Invicti, a web application security firm, states that "You have to find every security flaw, but a malicious hacker only has to find one [19]." Due to the nature of the internet, web applications are incredibly insecure. The security firm Positive Technologies performed audits on client security in 2017 and had a shocking revelation when they discovered that "... high-severity vulnerabilities [were] found in 100% of tested banking and finance web applications [20]." Banking and finance applications should be some of the most secure and safeguarded web applications, but as the previous quote states it can only take one security flaw to compromise a system.

3.6.1 Overview

Securing a web application is difficult, and arguably impossible to do perfectly [21]. The web application that I have created does not contain especially sensitive data compared to financial banking web applications, or government web applications, so a data breach would be a minor inconvenience to a user at worst, although I would never undermine the creativity of a malicious adversary. Regardless, I took as many steps as possible that I could to secure the web application.

My web application is running with Google Firebase. Google Firebase published a security checklist that includes a series of steps to secure a web application

[22]. Because my client has yet to deploy the project, I took all the steps that I could before they do. The series of steps I validated and incorporated include:

- I tested my cloud functions locally by utilizing emulators.
- Abusive traffic cannot disrupt my only cloud function or use it to abuse users because the cloud functions run on a scheduled cron job.
- I created security rules for every resource location in my Cloud Firestore.
- I set quota limits on email-password authentication.
- I do not pass authentication tokens.
- I do not use anonymous authentication.

The above steps secure my backend, but to ensure that users are only able to read, write, update, and delete from places they should be allowed to do so, I routinely check their authentication whenever it changes or whenever they visit a different page to ensure they are only able to do that for which they have permissions. Finally, because I wanted to provide users with adequate feedback if an action they took is unsuccessful or successful, I opted to display generic error messages to prevent potential adversaries from taking advantage of an error code to exploit the system.

3.6.2 Implementation

Creating a secure application requires both proactively taking steps to ensure the application is impenetrable, and not accidentally making mistakes which cause an otherwise secure application to become vulnerable to adversaries, such as leaving API keys that control backend services publicly exposed on the client-side.

There are several measures I took to secure my backend services which I listed above. The earliest measure I took was placing the API key I use for OpenWeatherMap in the backend. If I left my API key client-side, then any

adversary could take the key and automate as many requests as they would like, which would completely shut down the main utility of my application as a locked-out key could not make requests to a service provider.

The second measure I took was to use a combination of authentication services provided by Firebase and Cloud Firestore Security rules to handle serverless authentication, authorization, and data validation. I must protect the data stored in my backend from both intentionally malicious adversaries, and users who do not mean harm but have found themselves accidentally accessing and mutating data they should not have. The first step for this is to allow users to create accounts which then serves as their authentication. I discussed this step in Section 3.3.2.1 Account Authentication. With account authentication, I can then get a user's authorization before a user takes an action that requires it. I was able to do this by creating a computed property in Vue within each page of the application that uses Firebase Authentication to get the current user's authentication [23]. Instead of passing this authentication around, the program will get the current needed authentication version. This guarantees that a user has not mutated their authorization before performing the action, and that it is up to date.

With authentication and authorization, I then wrote security rules in my Cloud Firestore to check if a user has the correct request permissions for an action [24]. When a user makes a request to a document or collection in the Cloud Firestore, the rules will check various parameters such as the user's authorization, and their authorization ID to know if the Cloud Firestore should complete the request. For example, all users both authorized and not authorized should have read access to all organizations and spaces, but only the owner of the organization should have write access to them. Only the user should be able to read and write the notifications that they created. A question then arises for server-side work as Cloud Functions will need to be able to read and write to users' notifications but are not the user. Fortunately, Cloud Functions are a server client library, and server client libraries bypass all Cloud Firestore Security Rules [25].

3.6.3 Future Work

In Section 3.6.1 Overview I listed a series of steps that I took to secure the application, but I left off many because many of the decisions are up to my client. There are several security measures that future developers must take after the deployment of the application or near deployment of it and include the following:

- Enabling App Check to protect backend resources from abuse [26].
- Creating alerts and monitors for resources nearing their thresholds and quotas.
- Configuring usage thresholds for all billable resources.
- Restrict the use of the API key for the Firebase App to the deployed website [27].
- Monitor quota and usage for login attempts so that logins cannot be brute forced.
- Enable OAuth2.0 providers

With all the above implemented, the application would be at an exceptionally secure state especially given the unlikelihood of a malicious adversary.

3.7 Accessibility

3.7.1 Overview

When developing a website, it is imperative that developers keep people with accessibility barriers in mind. I am no different. The World Wide Web Consortium defines web accessibility as "... websites, tools, and technologies are designed and developed so that people with disabilities can use them [28]." Digital.gov acknowledges that the "globally recognized guidelines for creating accessible digital experiences..." comes from the "World Wide Web Consortium" and that

"WCAG 2.0 outlines the principles, guidelines, testable success criteria, and techniques needed to optimize content [29]." WCAG 2.0 has a set of four guidelines broken into sections. The guidelines are as follows:

1. Perceivable

    1.1 Provide text alternatives for any non-text content so that it can be changed into other forms people need, such as large print, braille, speech, symbols or simpler language.

    1.2 Provide alternatives for time-based media.

    1.3 Create content that can be presented in different ways (for example simpler layout) without losing information or structure.

    1.4 Make it easier for users to see and hear content including separating foreground from background.

2. Operable

    2.1 Make all functionality available from a keyboard.

    2.2 Provide users enough time to read and use content.

    2.3 Do not design content in a way that is known to cause seizures.

    2.4 Provide ways to help users navigate, find content, and determine where they are.

3. Understandable

    3.1 Make text content readable and understandable.

    3.2 Make Web pages appear and operate in predictable ways.

    3.3 Help users avoid and correct mistakes.

4. Robust

4.1 Maximize compatibility with current and future user agents, including assistive technologies.

My web application does not have many features that require precision when developing accessibility features for, such as rich multimedia websites. This automatically passes my web application on many of the above guidelines such as guideline 2.3 "Do not design content in a way that is known to cause seizures." The primary guidelines I paid attention to were supporting full functionality with only a keyboard and supporting screen readers, in addition to developing as many helpful messages as I could regarding pointing users in the right direction, alerting them when actions were handled successfully, and when errors occurred.

Although my web application does not need to convey complex interactions to the user or involve complex interactions from them, I still had to pay attention to these guidelines throughout the development process so that I could affirm that my web application adheres to them.

3.7.2 Implementation

My application uses many development dependencies that not only check for JavaScript errors but also enforce specific coding guidelines. These development dependencies originated with the initialization of the Vue application through the Vue CLI where I chose to add a linter, ESLint, and pair it with a formatter, Airbnb [31]. These development dependencies also included a Vue.js accessibility plugin that monitors the accessibility of my application [32]. With all these dependencies, not only was my linter checking to make sure my logic was correct and formatted correctly, but that it also adhered to accessibility guidelines as found in the GitHub for the plugin [33].

3.7.3 Future Work

I have been the only user to evaluate my web application, so no user with accessibility issues has assessed that they are not only able to use it, but able to

use it as easily as a user with no accessibility issues would be able to. Following application deployment, the application will need further testing along with feedback from users with varying levels of abilities to gauge accessibility and refine it for those users.

4 Investigation of Postliminary Research Objectives

Following the conclusion of the development of my web application, I had to turn my attention to gathering the necessary data on operating costs, analyzing potential deployment options for my clients to take, and formulating a list of application modifications they could make that they may be interested in.

4.1 Estimated Application Operating Costs

The services that I use for my application are Google Firebase, the MessageBird API, and OpenWeatherMap. When estimating operating costs, I have assumed an internal deployment, as an external deployment would be more difficult to gauge as the popularity of the app is not easily predictable. Google Firebase and OpenWeatherMap both have generous free tiers that should meet the needs of the Office of Sustainability. The OpenWeatherMap API allows for sixty calls/minute and one million calls/month on its free tier [34]. sixty calls/minute is an unlikely exceeded rate with an internal deployment. The features that Google Firebase provides that my web application utilizes are Cloud Firestore, and Cloud Functions. During development, I did not use more than an exceedingly small fraction of any of the free tiers.

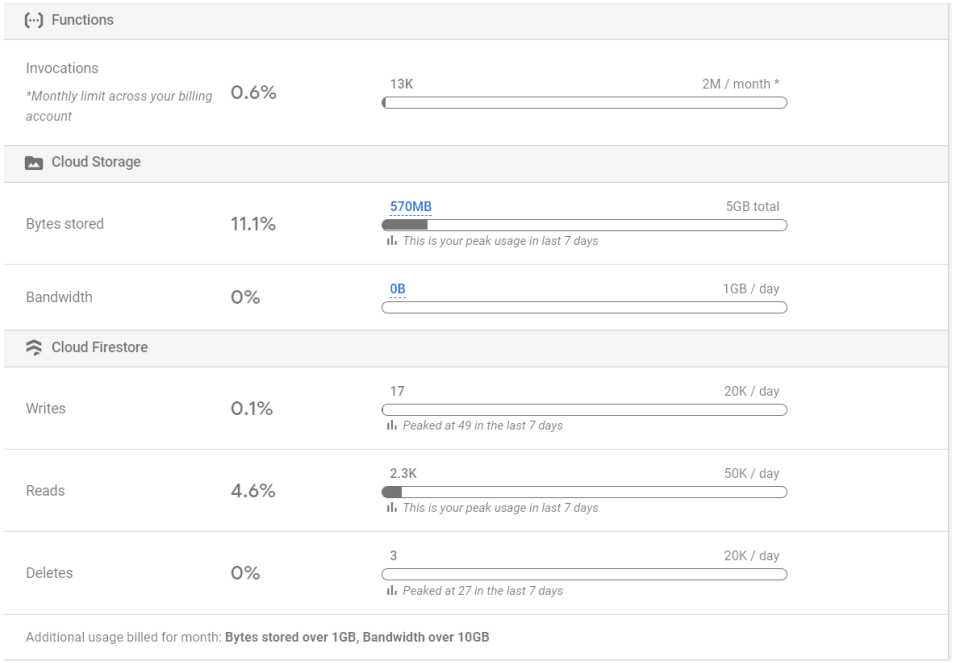

Figure 4.1: Operating costs during development

It is highly unlikely that the application exceeds any of these tiers, with the application most likely exceeding cloud storage, but at a pricing of \$0.026/GB the costs are miniscule [35]. Additionally, the clients will be responsible for the costs of CPU seconds and GB seconds as they relate to cloud functions, but Google only charged me \$0.05 throughout the course of development. Then, the only felt costs are the costs associated with sending text message notifications. It is unlikely my clients are going to be using the same text messaging API that I used, but the currently rated cost of sending an SMS is as low as \$0.006 per message [36]. Additionally, because the client will deploy the application in the United States there are associated carrier fees for sending messages ranging from $0.0020-0.0035$ [15]. Using this above information, I can estimate costs for each user for a month. Assuming a user receives three text messages a week, a user stores 100mb of data in the Cloud Firestore, then the average cost per user would be in the range of \$0.1086 - \$0.1266.

4.2 Application Deployment Options

The client must make several decisions and provide several resources as outlined in the future work subsections contained throughout Chapter 3 Development of the Application. These decisions involve security, such as setting billing quotas, and usage thresholds, and resources like a valid API key for handling SMS communication with users and billing for Google Firebase resources. Following a developer has completed the above precursory steps, the client can then decide the deployment strategy for the application.

4.2.1 External Deployment

I initially built this application with the intention of deploying it so that it would be publicly accessible, but initial issues have remained unresolved which are grounds for a concern with a public deployment. The primary issue at hand is the unmoderated organizations. Currently any user can create an organization so long as it has a valid city and state and a name that another organization has not already taken. This would then allow for a user to come in masquerading as another organization to trick users into searching for spaces in the organization they created. One solution to this problem could be a superadmin that moderates the registration of organizations. The issue with this is manual time it would take for a person to moderate the approval of organizations and admin accounts for those organizations. This would increase operating costs a significant amount. A second issue is the client would have to absorb all the operating costs for all organizations. Again, significantly increasing operating costs.

These issues are not insurmountable, and a public deployment does have its boons. For one, the volume of user feedback would greatly accelerate the application's utility. Additionally, because I created the application with a public deployment in mind, the development time spent on it would not be for naught. Finally, I created this application for the Office of Sustainability which would have more than just the interests of the University it represents at heart, as deploying

it publicly would serve to reduce the energy costs of each organization utilizing its decreasing global emission.

4.2.2 Internal Deployment

An alternative to this would be an internal deployment. This would remove all needs for a superadmin and would keep operating costs at a minimum. The chief issue with an internal deployment is that I did not write the app for an internal deployment. A developer would have to spend tens of hours to rewrite logic to accommodate for an internal deployment. Users would no longer need to search for an organization, they would not be able to create or manage an organization, they would not need to select an organization they would receive notifications from. To ensure internal use of the application only, a developer will have to add new logic for handling authentication and security as well. Although there are drawbacks to an internal deployment, the positives are a massive security increase assuming proper authentication and appropriately implemented security, and a more refined user experience as the application becomes constricted to the deployed organization.

4.2.3 Internal Deployment, Public source code

A third option that mixes the pros and cons of the two aforementioned approaches involves an internal deployment but with publicly available source code. This provides the benefits of an internal deployment as an admin will no longer have to arbitrate organization names, costs of maintaining the app will be at the lowest, application security will increase, and the application will have an improved user experience. If other organizations adopt the source code, then the application would benefit more than just the University of Arkansas allowing for an increased flow of user feedback.

The issue with this approach is that it would overall have the slowest deployment time because it would require the same extensive code rework required

for an internal deployment, in addition to creating the necessary documentation to supply to other organizations interested in deploying the application.

5 Conclusion

5.1 Results

Near the end of 2021, I began to investigate an implementation for a mobile application which was to guide an occupant's decision before they open a window. I then spent the following four months researching implementation strategies and developing the application. I was given freedom in adding or extending the application in ways that I thought would benefit the user, so at the end of the development I was able to complete all feature requests that were brought before me in addition to expanding on the utility of the application.

I had intended to coordinate a deployment of this application with the Office of Sustainability during the span of my thesis, but the limited availability of critical parties required that we reschedule and push back meetings that ultimately caused a delay too great for me to be able to acquire the information and decisions necessary to facilitate a deployment of the project during the thesis's timeline.

5.2 Future Work

This project has a tremendous amount of extensibility that I would hope another student or developer will pick up. The first step for the torch-bearer is to complete precursory deployment tasks and then decide on a deployment strategy with the client as I mentioned in Section 4.2 Application Deployment Options.

Following a successful deployment, a developer would be able to focus their time on addressing the future work sections outlined in Chapter 3 Deployment of the Application. During this time the client, developer, or other third party should conduct research on the effectiveness of the application following its deployment. Ideally, the University of Arkansas should notice decreased energy costs for HVAC systems, and overall comfort of building occupants increase amongst other factors.

The developer should then continuously hone and refine the application following user feedback until client and user alike are satisfied with the post-deployed application.

Bibliography

[1] "Heating, Ventilation and Air-Conditioning Systems, Part of Indoor Air Quality Design Tools for Schools," EPA, 16-Dec-2021. [Online]. Available: https://www.epa.gov/iaq-schools/heating-ventilation-and-air-conditioning-systems-part-indoor-air-quality-design-tools. [Accessed: 20-Apr-2022].

[2] "Introduction to client-side frameworks," Learn web development, 20-Apr-2022. [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Introduction. [Accessed: 20-Apr-2022].

[3] "About JavaScript ," JavaScript, 18-Apr-2022. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript. [Accessed: 20-Apr-2022].

[4] "Understanding client-side JavaScript frameworks," Learn web development , 18-Feb-2022. [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks. [Accessed: 22-Apr-2022].

[5] "A material design framework for vue.js," Vuetify. [Online]. Available: https://vuetifyjs.com/en/. [Accessed: 20-Apr-2022].

[6] "Firestore," Google Cloud. [Online]. Available: https://cloud.google.com/firestore. [Accessed: 22-Apr-2022].

[7] "About npm," npmjs. [Online]. Available: https://www.npmjs.com/about. [Accessed: 22-Apr-2022].

[8] "Viewport Sizes for iPhone," YesViz. [Online]. Available: https://yesviz.com/iphones.php. [Accessed: 22-Apr-2022].

[9] P. Morville, "User experience design," Semantic Studios, 21-Jun-2004. [Online]. Available: http://semanticstudios.com/user_experience_design/. [Accessed: 22-Apr-2022].

[10] "Usability principles," Stanford Improvement, Analytics, and Innovation Services. [Online]. Available: https://improvement.stanford.edu/resources/usability-principles. [Accessed: 22-Apr-2022].

[11] "Firebase Authentication," Firebase Documentation, 19-Apr-2022. [Online]. Available: https://firebase.google.com/docs/auth. [Accessed: 22-Apr-2022].

[12] "Get Started with Firebase Authentication on Websites ," Firebase Documentation, 22-Apr-2022. [Online]. Available: https://firebase.google.com/docs/auth/web/start. [Accessed: 22-Apr-2022].

[13] "Schedule functions ," Firebase Documentation, 19-Apr-2022. [Online]. Available: https://firebase.google.com/docs/functions/schedule-functions. [Accessed: 22-Apr-2022].

[14] "Send Messages with MessageBird," Firebase Extensions. [Online]. Available: https://firebase.google.com/products/extensions/messagebird-firestore-messagebird-send-msg. [Accessed: 22-Apr-2022].

[15] N. Rosenberg, "United States ," United States - Support & Help, 05-Apr-2022. [Online]. Available: https://support.messagebird.com/hc/en-us/articles/360018067337-United-States-. [Accessed: 23-Apr-2022].

[16] United States 10DLC FAQ, 22-Mar-2022. [Online]. Available: https://support.messagebird.com/hc/en-us/articles/208747865-United-States-10DLC-FAQ. [Accessed: 23-Apr-2022].

[17] OpenWeatherMap.org, "Air Pollution API concept," Air Pollution. [Online]. Available: https://openweathermap.org/api/air-pollution. [Accessed: 22-Apr-2022].

[18] "Manage Users in Firebase," Firebase Documentation, 22-Apr-2022. [Online]. Available: https://firebase.google.com/docs/auth/web/manage-users. [Accessed: 22-Apr-2022].

[19] R. Abela, "The dangerous complexity of web application security," Web Application Security can Become Dangerously Complex , 14-Mar-2022. [Online]. Available: https://www.invicti.com/blog/web-security/dangerous-complexity-of-web-application-security/. [Accessed: 22-Apr-2022].

[20] "Positive Technologies research: Banking and finance were the most vulnerable web applications in 2017," Positive Technologies research, 16-Apr-2018. [Online]. Available: https://www.ptsecurity.com/ww-en/about/news/banking-and-finance-were-the-most-vulnerable-web-applications-in-2017/. [Accessed: 22-Apr-2022].

[21] B. Rotibi, Can a web app ever be truly secure?, 14-Jul-2021. [Online]. Available: https://www.computerweekly.com/opinion/Can-a-web-app-ever-be-truly-secure. [Accessed: 22-Apr-2022].

[22] "Firebase security checklist," Firebase Support, 19-Apr-2022. [Online]. Available: https://firebase.google.com/support/guides/security-checklist. [Accessed: 22-Apr-2022].

[23] "Manage Users in Firebase ," Firebase Documentation, 22-Apr-2022. [Online]. Available: https://firebase.google.com/docs/auth/web/manage-users#web-version-9_2. [Accessed: 22-Apr-2022].

[24] "Get started with Cloud Firestore Security Rules ," Firebase Documentation , 22-Apr-2022. [Online]. Available: https://firebase.google.com/docs/firestore/security/get-started. [Accessed: 22-Apr-2022].

[25] "Structuring Cloud Firestore Security Rules ," Firebase Documentation, 22-Apr-2022. [Online]. Available: https://firebase.google.com/docs/firestore/security/rules-structure. [Accessed: 22-Apr-2022].

[26] "Firebase App Check," Firebase Documentation, 19-Apr-2022. [Online]. Available: https://firebase.google.com/docs/app-check. [Accessed: 22-Apr-2022].

[27] "Using API Keys ," Google Cloud, 22-Apr-2022. [Online]. Available: https://cloud.google.com/docs/authentication/api-keys#api_key_restrictions. [Accessed: 22-Apr-2022].

[28] S. L. Henry, "Introduction to Web Accessibility," Web Accessibility Initiative (WAI), 31-Mar-2022. [Online]. Available: https://www.w3.org/WAI/fundamentals/accessibility-intro/. [Accessed: 22-Apr-2022].

[29] T. Bonitto, "An Introduction to Accessibility," Digital.gov, 03-May-2021. [Online]. Available: https://digital.gov/resources/introduction-accessibility/. [Accessed: 22-Apr-2022].

[30] "Web Content Accessibility Guidelines (WCAG) 2.0," Web content accessibility guidelines (WCAG) 2.0. [Online]. Available: https://www.w3.org/TR/WCAG20/. [Accessed: 23-Apr-2022].

[31] "Plugins and Presets," Vue CLI, 17-Feb-2022. [Online]. Available: https://cli.vuejs.org/guide/plugins-and-presets.html#installing-plugins-in-an-existing-project. [Accessed: 22-Apr-2022].

[32] "Eslint-plugin-vuejs-accessibility," npm, 22-Jan-2022. [Online]. Available: https://www.npmjs.com/package/eslint-plugin-vuejs-accessibility. [Accessed: 22-Apr-2022].

[33] vue-a11y, "eslint-plugin-vuejs-accessibility," GitHub, 22-Apr-2022. [Online]. Available: https://github.com/vue-a11y/eslint-plugin-vuejs-accessibility. [Accessed: 22-Apr-2022].

[34] "Pricing," OpenWeatherMap. [Online]. Available: https://openweathermap.org/price. [Accessed: 22-Apr-2022].

[35] Firebase pricing. [Online]. Available: https://firebase.google.com/pricing. [Accessed: 22-Apr-2022].

[36] "Plans & pricing," Pricing - MessageBird. [Online]. Available: https://www.messagebird.com/en/pricing/api. [Accessed: 22-Apr-2022].