

| **Sponsors:** | Northern Arizona UniversityUSDAUS Forest Service |
| --- | --- |
| **Clients:** | Dr. Jamie S. Sanderlin, Dr. Ana Miller-ter Kuile, and Dr. Kiona Ogle |
| **Mentor:** | Scott LaRocca |
| **Team Members:** | Alyssa Ortiz, Andrew Ortega, Payton Watts, and Tyler Chapp |

**November 1, 2024**

**Table of Contents**

[**Introduction 3**](#_x5xzf5a1bik6)

[**Technological Challenges 4**](#_t6vitu1rrdfq)

[**Tech analysis 5**](#_rhy025yyxtr)

[Front end 5](#_iatsxc32v6jx)

[Back end 6](#_48lw4j6coqfh)

[Mapping & Data Visualization 10](#_niw7maqz1r1e)

[Hosting & Version Control 13](#_uzcyloqanh0)

[**Technology Integration 16**](#_fg2l35dbdmlj)

[**Conclusion 16**](#_s03vy2gtzcm4)

[**Resources 17**](#_i3pbsvpcs6qo)

#

# Introduction

## Big Picture

Anyone who has lived in Northern Arizona for some time knows that every year around the beginning of Fall, the smell of smoke in the air becomes a frequent occurrence. But beyond the fires that are easily noticeable from Flagstaff and other towns and cities, any drive across part of the Western United States often has many portions of land that have been very noticeably scarred by a once-raging wildfire. While the protection of human homes is a priority, there’s an often-forgotten group whose homes are typically completely destroyed by the fires. The wildlife, who live in the vast swathes of land that are affected every year, are frequently displaced or even killed by these fires. As climate change further affects our wilderness and leads to more wildfires, the ability for forest management organizations like the USDA Forest Service to predict how these animals are affected becomes more and more imperative. The key to tracking these effects lies in one type of animal in particular- birds. From the days of miners using canaries to test the air, to signaling deforestation in the jungle, birds have been a reliable litmus test of environmental conditions. This project hopes to use large datasets of bird presence data across various regions in the Western United States to understand how past and present forest management combined with fire has affected each ecosystem.

## The Current Solution

 The issue with the current solution to tracking and interpreting bird data is that there really is no standardized, easy to use method available. Currently, if scientists or managers with access to the ten-year bird presence data want to make any predictions or conclusions, they have to carefully peruse the appropriate datasets to find the right information to use. Then, they have to manually analyze and visualize the data by their own means. Due to these challenges in just accessing and modeling, it’s very hard for the forest scientists and managers to then communicate any issues they detect to land managers, conservation organizations, and other interested parties who could make a difference. What this project aims to achieve is to improve the data retrieval process and streamline it into a simple, easy to understand and use data analysis and modeling program that also provides the data displayed as a download.

## Project Solution

The project solution, FireFlight, is a web-based application designed to assist forest service managers and researchers in visualizing bird community response to wildfires and forest service management strategies. Using data collected over the past ten years, this tool will assist in decision-making by visualizing historical trends, assessing current conditions, and predicting future ecosystem recovery.

Key features will include an interactive map showing bird community and wildfire data and the ability to visualize trends across local, regional, and national scales. Real-time data analysis will provide insight into how forest service management strategies impact bird communities and overall ecosystem health. The tool will need to generate CSV or other geospatial files that will provide users with flexibility in further analysis and research. Regular communication with stakeholders will ensure the application's design aligns with the user's needs.

In order for this proposed solution to become a reality, the team would need to focus on the possible challenges that can be faced. In the subsequent sections, it will be discussed what technologies can be used. It will also be discussed in how beneficial each technology will be in terms of the project.

# Technological Challenges

As we begin developing FireFlight, there are several key technological challenges the team needs to address:

1. User-Friendly Interface: Ensuring that FireFlight provides an intuitive and easy-to-navigate interface is crucial, particularly for non-technical users. Designing a system that presents complex data clearly and understandably will require careful attention to the user experience.
2. Data Visualization: One of the key challenges is displaying complex bird and wildfire data in a format that is easy for users to interpret. The interface must effectively convey data trends across different scales, from local to national, while avoiding overwhelming users with too much information at once.
3. Scalability: FireFlight will need to handle large geospatial datasets across multiple scales of analysis. Ensuring the system remains responsive and reliable when working with these massive data inputs is a critical challenge that requires efficient data processing and system infrastructure.
4. Exportable Results: The application must enable managers to export their results in flexible formats, such as CSV or other geospatial files. Ensuring that the export process is smooth and reliable while maintaining data accuracy, is another import challenge of our project.

# Tech analysis

## Front end

### Introduction:

* Front end development focuses on the part of the website that the user will see. They will see the tabs, the pictures, the links, etc. Each of these aspects of a website are essential to working properly in order for our data and information to be articulated properly to the user.

### Desired Characteristics:

* One of the desired characteristics is that our project looks as professional as possible. With a professional appearance, it will allow us to be seen as more credible. This is a result of maintenance being kept up on, and the information being accurately displayed to the user.
* Another characteristic would be user functionality. No one wants to have difficulty trying to navigate a website to reach their desired objective. With a focus on user functionality, It would draw in people to use our site.

### Alternative:

* One such approach that can help us attain these characteristics is Bootstrap. The team found out about Bootstrap from our team member Alyssa, who has had prior work experience with the product. Bootstrap is one of the most popular CSS frameworks that allows for ease in developing responsive sites. It also allows for ease in use of the site from a mobile device. Prior to being open-sourced code, it was known as *Twitter Blueprint*, since it started out from some developers who worked at Twitter in the 2010’s([1](#_i3pbsvpcs6qo)).
* Another tool that could be used to help attain user functionality and website appeal would be HTML. HTML works well with Bootstrap and is used as a framework for websites. While it is viewed as a building block to the web, HTML focuses on text of a site and links. Each team member has used HTML in previous web design courses and other practices.
* An additional tool that can be used is Javascript. Javascript works well with HTML and Bootstrap. The use of Javascript can help build upon the functionality of the site that HTML provides. In addition, it can also set objects up for Bootstrap to manipulate on the site, in terms of placement and design. We have some experience using Javascript in web development courses in our education.

### Analysis:

* The framework and usages for this project were determined based on a collective agreement. It was decided that HTML would be used as the framework for the website. It was also determined that Bootstrap would be implemented to customize the website's usability(CSS). The team feels comfortable writing code in Html and CSS, using Bootstrap for our CSS. While some members are still learning Bootstrap, tutorials have been suggested so that each team member can properly use Bootstrap to display the CSS that we want for our website.

### Chosen Approach:

* HTML and Bootstrap were the top contenders. HTML would be used for the text and the links of the website. Bootstrap would be used for the layout and design of the website.

| Technology: | Desired Characteristics: |
| --- | --- |
| Bootstrap | Helps build responsive sites, easily applicable to HTML framework |
| HTML | Framework for websites |
| Javascript | Not as clearly defined functions as HTML |

### Proving Feasibility:

* The decision of using Bootstrap and HTML has already been implemented into our website. The functionality of the two has performed the way that was envisioned. It has met the desired goals of the team, with respect to files being easily accessible, images displaying, and subsections being separated from each other. There are intentions to set up additional parts to the website and use similar layouts. The same CSS will be implemented, this way the website can stay uniform.

## Back end

### Introduction:

* The back end development will involve working on mainly the server-side code that works in the background between our programs and the website. Thus, the main challenge here will be efficiently connecting our working data-modeling program to our product’s website, to create a functional, intuitive web application for our intended users to interact with.

### Desired Characteristics:

* The ideal solution seems to be using the Python-based high-level web application framework **Django**. Part of what makes Django so great for taking care of this project’s back end concerns is its immense versatility and wide array of tools it offers. Running off of a web server, one important thing that Django provides is an object-relational mapper(ORM), which allows us to interact with database models in ways similar to SQL, such as adding, deleting, modifying and querying data objects. Another part of how Django functions is its Model View Controller(MVC) architecture. The model part is our access to the relational database, the view portion is for viewing our web application’s template, and lastly, the controller pertains to the logic that decides what should be brought out of the database and onto the web view, along with getting user input. All in all, Django’s main purpose is to efficiently create a web framework for Python programs, which could prove very useful to this project as the majority of code is planned to be done in Python.

### Alternatives:

* Django may be the solution that seems most optimal for the project, but it’s certainly not the only possibility.
* In terms of other Python-based web frameworks, another big name is **Flask**. Flask would also be relatively easy to implement, if not easier than Django. This is due to the various tradeoffs it makes compared to other web frameworks. Technically considered a microframework, Flask is a much more bare-bones approach to a Python web framework. On its own, it has no database abstraction, and potential scalability issues with high traffic. However, extensions do exist for ORMs and such, but there could be potential issues relying on lots of third parties for features.
* Outside of Python, the project could also adopt an **Express** web framework, which makes use of Node.js. Like Flask, Express tries to be minimalist and lightweight. But unlike Flask, Express provides far more solid grounds for detailed web development, with features like HTTP utility methods, routing, and middleware support for intercepting and manipulating requests.
* Another non-Python web framework option we could choose to use is **Ruby on Rails**. Like Django, Rails also follows the idea of utilizing MVC architecture, allowing easy integration of database information into the web view. It also has many other tech standards it follows that ensure a structured approach to development. Rails is also quite fast, but this benefit also comes with a decent amount of overhead.

### Analysis:

* The team’s analysis of each backend method was largely based on a specific set of criteria that would ensure the best functionality once implemented. One aspect that we looked for in the following methods was flexibility. The team desires an approach that isn’t specifically crafted just for large or small scale projects, as this project’s back end is not the main focus, but also not negligible. Another criteria we judged by was scalability. This project could see a lot of traffic weighing down on its web framework, especially between the server and website when it comes to taking in user inputs and replying with data and models, and we would like to be sure our chosen method can handle that. Lastly, we were concerned with ease of use. The project’s focus is on the data and modeling/analyzing it, and the team can’t afford to spend too much time struggling with an unintuitive backend that may require excessive add-ons that require even more learning to use.
* Django is a web application framework that was heavily vouched for by the project’s team lead as a reliable, intuitive, and efficient platform for backend management with Python. This decision was further influenced by the fact that Django is very friendly towards working closely with data and implementing it into the web application, which will be an essential part of the finished product. It also provides significant scalability, and comes with a lot of flexible, useful features as-is, instead of requiring the installation of additional features like some web frameworks, which is also a benefit. The main issue with Django is that it will require studying quite a bit of Django-specific documentation to be able to use it, as while the framework is built around and implemented using Python, Django introduces a lot of its own unique features to Python that are needed for successful implementation of the backend.
* Flask is probably the most well known Python-based web framework, second only to Django, and is probably the most influential. The most influential feature of Flask that drew the team to consider it is its well-known lightweight structure. Compared to Django, getting a Flask framework up and running is a lot less of a major undertaking. However, this comes at the cost of sacrificing easy use of databases with it, as it comes with no database-handling capabilities. Flask also risks not being able to handle large amounts of traffic, which goes against our team’s desire for scalability. Some of Flask’s issues can be solved using extensions, but as mentioned before, that’s not a very desirable trait to the group, as it often requires lots of installation and reading documentation for something that could only make a minor improvement.
* The Express framework is another option we’ve considered. It’s based around JavaScript, which isn’t necessarily preferred by the team, as when it comes to object-oriented languages, most members are versed in Python instead. However, it does offer a flexible and lightweight approach, without a lot of the same pitfalls as Flask, albeit still needing add-ons/middleware to fulfill certain needs.
* Ruby on Rails is a unique framework with a lot of different benefits but also hindrances. The main issue is that it’s in Ruby. On one hand, Ruby is said to be a very streamlined and efficient coding language, often requiring less lines of code to do the same task as C, for example. But on the other hand, most if not all of the team aren’t very well-versed in Ruby, if at all. But beyond the language it’s built around, Rails has a lot of other important facets to it. Unlike Express and Flask, Rails is like Django in that it comes with many features built into it by default. There’s a lot less worrying about installing and adjusting to middleware. Also like Django, Rails provides quite a bit of scalability. The issue with Rails is that it comes with a decent bit of overhead, and will require the team not just learning a new coding language but also Rails’ method of implementing it.

### Chosen Approach:

* Django offers the best balance of built-in abilities to flexibility and scalability. Rails offers a wide array of abilities and a MVC-based structure too, but none of the team know Ruby, and beyond that, Rails may prove a bit too complex and bulky for what is needed of a web framework for this project. Express is another top contender, with flexibility and a lightweight build. However, the team isn’t as experienced in JavaScript as they are in Python, and learning a whole framework built around it would be difficult, especially if the team needs to build upon its limited built-ins. Lastly, Flask offers a very lightweight and easy approach, at the cost of having next to no built-in features and potential scalability issues.

| Technology: | Desired Characteristics: |
| --- | --- |
| Django(Chosen) | Lots of features, flexible, scalable, data-oriented |
| Ruby on Rails | Lots of features, scalable |
| Express | Flexible, Lightweight |
| Flask | Lightweight, ease of use |

### Proving Feasibility:

* Aside from the other backend code for interpreting and modeling the data, which will be done in just Python itself, the Django backend implementation will most likely be tested later into the project, once the other essential building blocks have been tested. However when the time comes to try out Django, the team will be using it essentially as a bridge between our datasets and the programs that interpret them, and the frontend, in particular the product website. This will prove how well it can function at the task of combining the elements of the program and website into a single well-functioning web application.

## Mapping & Data Visualization

### Introduction:

* To get meaningful insights from wildfire data and ecosystem health, mapping and data visualization play a critical role. To achieve this, the tool will need to include an interactive and user-friendly visualization, enable easy integration with complex data sources, and streamline the embedding process.

### Desired Characteristics:

* The most important characteristic of data visualization technology is the creation of interactive map features. The tool needs to support interactive features like clicking on points of interest, zooming, panning, and other similar map transformations. This enhances user experience by allowing them to explore the data dynamically to get meaningful insights.
* Another important feature is this technology needs to allow for easy integration with the Python ecosystem, as the system’s back-end is built in Python. This ensures a smooth workflow and easy data integration within a single environment.
* Layering and customizability are significant requirements for this project. The ability to customize maps by adding multiple layers such as heatmaps, markers, and polygons is essential. The tool also needs to be able to analyze at different levels of analysis such as local, regional, and national. Tools that offer high customizability are ideal for data visualization requirements.
* Another crucial feature of the chosen technology is the support and performance of processing multiple, large geospatial data sets. Given the size of tens of years of collected data, the tool must efficiently handle large amounts of data without significant performance issues. The tool must also handle a wide range of geospatial data formats to allow flexibility in researchers' use of the final product.
* Lastly, the product will be presented as a web-based application so our chosen data visualization technology must support this.

### Alternatives:

* Google Earth Engine API is a robust platform for sourcing and analyzing satellite and other geospatial data. It is incredibly powerful for data processing but requires some familiarity with Google’s cloud-based structure and API. Although it is powerful and could help solve the project’s data visualization problem, it may be unnecessary as a huge part of Google Earth Engine API is sourcing data from satellites. Most, if not all of the data required for the team’s product will be provided to us by the clients.
* Mapbox GL JS is a powerful JavaScript library that allows for the creation of highly customizable and interactive maps. It offers many options for layering, visualization, and data processing, but requires a more complex integration with Python than another framework that is also in Python.
* Kepler.gl is a browser-based tool that helps visualize large geospatial datasets. While it is a great tool for data visualization, it does not include any data processing capabilities. Can be thought of as primarily a front-end framework that cannot process data and lacks the flexibility needed for server integration with Python workflows.

### Analysis:

* To evaluate each alternative, we tested them on the following criteria: customization, handling of large datasets, Python integration, and the ease of web embedding. By testing each of these tools with web-based tutorials and small demo projects, the team was able to discern which technology was the absolute best choice for handling mapping and data visualization.
* Folium is a Python-based library that can be naturally integrated into a Python development environment with ease. This allows the team to quickly create interactive maps and embed them into a web-based application without needing to implement complex workarounds. It supports other Python libraries that assist with dataset processing like GeoPandas which will make it easy to work with data provided by the client.
* Google Earth Engine API platform is designed to handle massive geospatial analysis and is particularly strong in processing satellite imagery. However, the project does not require the high-level data processing capabilities that GEE offers. Most, if not all of the geospatial data required for our project will be readily available from the clients. This means that the strength of this technology is not relevant to our project.
* Mapbox GL JS offers the highest level of customization options in creating interactive maps with layers and data points. However, it requires integration with Javascript which adds complexity to a Python-based system. This means handling more complex communication between the back-end and front-end when not in the same language. This slows down the development process and puts some of the development time and focus on integration rather than focusing on solving the problem.
* Kepler.gl is excellent for visualizing large geospatial datasets quickly and efficiently but is primarily used as a front-end visualization tool. Kepler.gl does not offer any data processing capabilities and this limits its practicality in our project. Because of its lack of data processing capabilities and also weaker integration with Python making it a less appealing choice for our mapping technology.

### Chosen Approach:

* Based on our research and evaluations, Folium was selected as the best fit for the system. Its balance of simplicity, Python compatibility, and web embedding without sacrificing performance or flexibility makes it an ideal choice for our project. Although Mapbox GS JS provides a bit more customization options, Folium’s ease of integration and support with Python frameworks makes it the most practical choice.

### Proving Feasibility:

* To show validation of this choice, the team will develop a demo where Folium integrates client-provided geospatial data and visualizes it in a web-based application. Starting small, with one small region in New Mexico that has already been provided to us by the clients. Although this region’s geospatial dataset is relatively small compared to the full dataset, it still contains over five thousand data points and will ensure system performance can handle large amounts of geospatial data. This will further prove Folium is the best choice for our system.

| Technology: | Customization: | Large Data Handling: | Python Integration: | Web Embedding: | **Overall Rating:** |
| --- | --- | --- | --- | --- | --- |
| Folium (Chosen) | ⭐⭐⭐⭐ | ⭐⭐⭐⭐ | ⭐⭐⭐⭐⭐ | ⭐⭐⭐⭐ | ⭐⭐⭐⭐⭐ |
| Google Earth Engine | ⭐⭐⭐ | ⭐⭐⭐⭐⭐ | ⭐⭐⭐ | ⭐⭐⭐ | ⭐⭐⭐⭐ |
| Mapbox GL JS | ⭐⭐⭐⭐⭐ | ⭐⭐⭐⭐ | ⭐ | ⭐⭐⭐⭐⭐ | ⭐⭐⭐⭐ |
| Kepler.gl | ⭐⭐⭐ | ⭐⭐⭐⭐ | ⭐⭐ | ⭐⭐⭐⭐⭐ | ⭐⭐⭐ |

##

## Hosting & Version Control

### Introduction:

* Our tool needs to be able to be easily updated and accessed not only by us, but by everyone who will use it and maintain it in the future. Therefore, we need to find a hosting service that will be able to function outside of the NAU ITS services, so that it can be passed on in the future. In the development process, we also need to be able to quickly test and deploy our tool, and make sure that we can roll back easily if there are any issues.

### Desired Characteristics:

* The most important thing we need to properly host our tool is a service that we can pass along control to the clients after we are done developing everything. We need to have a hosting service that will be easily migratable to new users and developers.
* The tool must be easily maintainable. We need to be able to quickly push and roll back any changes that are made to the tool, to ensure timely updates and fixes.
* The tool must be easy to access by developers for development and maintenance. It should not be complex or difficult to access and deploy, so that every current team member (and future developers) are able to easily and quickly access the tools hosting service without outside assistance.
* Our end product should also be able to utilize automated deployment of the tool. This will be extremely helpful in reducing the workload of the development team, so we do not waste time on manual deployment when we can work on more productive tasks.

### Alternative:

* NAU ITS: This was presented as the default option for hosting our tool by our capstone instructor, Professor Igor Steinmacher. NAU’s Information Technology Services (ITS) provides web and server hosting for NAU affiliated projects. It is maintained by the ITS department of NAU. ([2](#_i3pbsvpcs6qo))
* Amazon Web Services (AWS): AWS was presented as a second-choice alternative by our capstone instructor, Professor Igor Steinmacher. AWS was created by Amazon, and is a cloud provider that has many services, such as hosting. There is a large variety of options for hosting to choose from, and different pricing tiers for each. ([3](#_i3pbsvpcs6qo))
* Microsoft Azure: Azure is a popular alternative to AWS, and was found through the recommendations of articles found online about hosting and deployment. Azure was created by Microsoft, and is a cloud and services provider, including hosting. There are many options to choose from for services, but if we decided to use their cloud service, we would also choose to use their version control and deployment service as well, called Azure DevOps. ([4](#_i3pbsvpcs6qo))
* GitHub: GitHub is a version control system based off of the open source Git. GitHub is the standard recommended by every instructor at NAU for collaboration and version control. It was created in 2007, and is used by almost every modern developer today for everything from collaborative coding to automatic deployment. ([5](#_i3pbsvpcs6qo))

### Analysis:

* Each alternative above was evaluated based on ease of use, how familiar our team members are with the service, and the level of support available for the service. To determine the ease of use, the documentation of each product was reviewed to determine the requirements for use, as well as how much time would be needed to learn how to use it. Our team discussed which products we are most comfortable with using, and which ones we did not have experience with. Discussing this helped us determine which products would be easier to learn and use for the team, as well as future developers of our project. The level of support was the easiest to determine, as each product clearly lists what support is and is not offered.
* Automation of deployment was only assessed for GitHub and Microsoft Azure, since these are the only services that we would consider for it. AWS is the easiest to access and maintain, as proven by our group's personal experience and official documentation. Azure would be more difficult to learn, as well as set up. NAU ITS is not easily migratable, and does not have reliable service or documentation. The documentation contains many errors, and high quality maintenance and service is not guaranteed. GitHub is very reliable, and is an industry standard that all of our team members are familiar and comfortable with using. It is free and based on open source code, so help is not paywalled (as it is with Azure DevOps). AzureDevOps would take time to learn, and is not as easy to use and access, as shown in documentation. Because of this, we have decided to go forward with AWS and GitHub as our hosting and version control systems.

### Chosen Approach:

* After analyzing the options, we decided to use AWS for hosting, and GitHub for version control and automated deployment. While NAU ITS was initially our first choice for hosting, it did not meet the key requirement of easy migration. We also would not have total control over the system, and we are not sure how well it would be maintained by NAU ITS. Microsoft Azure (and Azure DevOps) were considered, but our team is not familiar with these products like we are with AWS and GitHub. Both AWS and GitHub have been used by all of our team members before, and have extensive documentation to fall back on when we need help.
* The table below shows each product, and which of the desired characteristics it was found to have.

| Technology: | Desired Characteristics: |
| --- | --- |
| AWS (Chosen) | Easily migratable, maintainable, ease of access |
| GitHub (Chosen) |  Easily migratable, maintainable, ease of access, automation of deployment |
| Microsoft Azure | Easily migratable, maintainable, automation of deployment |
| NAU ITS | Ease of access |

### Proving Feasibility:

* To test our chosen alternative, we will link our AWS server to GitHub, which will enable us to automatically deploy. We will prove this by changing something in the GitHub repository, and then reloading the webpage hosting our tool to demonstrate the automated deployment feature. We will also ensure that we can quickly roll back these changes, using our GitHub linked to our server. We will demonstrate this in a similar way to how we demonstrated the quick and automated deployment.

#

# Technology Integration

All of these parts mentioned above work great individually, the trouble is getting them to work together well. The architecture of our system must be able to handle a variety of user inputs, and be able to fetch data and display it quickly and efficiently.

On the top layer, we have the User Interface, where the user can view a probability heat map for bird populations in a given area. Any options they choose for displaying data will be sent to Folium, and Folium will request the relevant information from our clients database. Folium will then render the information as a raster heat map, which it will overlay the map data, and display to the user. Our AWS server will be hosting our website, and we will use GitHub to push new code to our server, for both the front end and the back end.

# Conclusion

 As the issue of climate change’s effects on nature and the average person’s life become harder and harder to ignore, and wildfires become bigger and bolder each year, the ability to accurately predict the outcome of these events becomes more and more imperative. Birds provide a valuable insight into how climate change and wildfires affect ecosystems and in particular, animal communities. Currently, the serious lack of ease-of-use in methods of acquiring and modeling bird presence prediction data makes it incredibly difficult to take advantage of an invaluable natural litmus test for the aforementioned effects of wildfire on wildlife and ecosystems at large. This project seeks to simplify current data acquisition and modeling methods by streamlining them into a versatile and intuitive web-based application that can clearly map out the bird presence predictions in an area following a fire. This will be done with a diverse and expansive tech stack. The main Python program will convert comma separated value data sets into raster data, which can then be fed into our Django web framework backend, to be converted into real maps on the web application using the Folium library. The website will be created with HTML and CSS, supported by Bootstrap for easy CSS implementations. The server the website will run on will be an Amazon AWS EC2 instance, of which we will upload our code to. Code will be stored on GitHub, as it provides the best environment for collaboration and source control. This project has a long way to go, but solidifying the choices for the tech stack has ensured stable ground for moving onwards. With this basis, moving on to the design review phase and tech demo should proceed smoothly, thanks to the building blocks of the minimum viable product being outlined successfully.

#

# Resources

(1) <https://getbootstrap.com/docs/4.0/about/overview/>

(2) <https://in.nau.edu/its/servers-and-hosting/>

(3) <https://aws.amazon.com/>

(4) <https://azure.microsoft.com/en-us>

(5) <https://github.com/about>