Technological Feasibility Final

Team Clean Carbon

October 24, 2022

Sponsor: Allie Shenkin

Mentor: Vahid

Members: Curtis McHone, Richard McCue, Shayne Sellner,

Justin Stouffer, Jonathan Bloom

Table of Contents

1. Introduction

Over the past 20 years, climate change has become a growing concern as scientists have seen increasing amounts of carbon dioxide and other greenhouse gasses in the atmosphere. Large companies have recently started to invest in combating climate change by offsetting their carbon footprint and reducing their carbon emissions where possible. Companies that want to offset their carbon emission can do so by purchasing carbon credits. Purchasing carbon credits allows large companies to invest in a plot of land that is viable for reforestation, or in a current growing reforestation project that has carbon credits for sale. These plots of land would be granted carbon credits based on the amount of carbon dioxide that the forest will uptake from the atmosphere. Thus, this uptake of carbon dioxide from the atmosphere offsets the emissions that these companies produce on a daily basis.

The carbon market is a relatively new market, and therefore there are not many tools available to the average consumer. In order to get estimates about reforestation statistics like carbon uptake, costs, etc, a consumer has to contact the individuals who manage the reforestation plot. This is incredibly inefficient, so our sponsor Allie Shenkin has sought to make tools that increase the efficiency of this process. Allie Shenkin is an Assistant Research Professor at NAU SICCS, as well as an Associated Scientist in Oxford's Ecosystem Lab. In addition to these tools, Allie and his colleagues have discovered a new climate cooling service, which aims to increase the profitability of the carbon market. It does this by increasing the number of carbon credits a plot of land would produce by up to 30%. Allie's current tool helps solve this problem by allowing consumers to get estimates for their plot of land using the data from his new scientific finding; however, the tool is still lacking. Because this tool is dealing with massive amounts of data that computes on the Google Earth engine, it is quite slow.

Our solution to this problem is to make a consumer-friendly web based application that allows a consumer to get an estimate on the amount of carbon credits they can expect from any given plot of land. The web-interface will feature an interactive map that the consumer is able to draw on, specifying the exact area and dimensions of the plot of land they want to process, which gets converted into a shape file for processing. The consumer will also be able to upload shape files instead of drawing on a map, if this suits them better. As this requires computing with large datasets, a very powerful backend is going to be required. This system will implement a REST API that will communicate the shapefile input from the consumer and send it to the backend. The backend will then interpret and compute the amount of carbon credits that can be expected from the given area that will be reforested and return those estimations to the consumer. Our solution would expand on and improve Allie Shenkin's current system to predict carbon credits. Our system sends the computation to the backend rather than computing natively on the Google Earth engine. This improves the consumer experience tremendously by speeding up the computation and making the consumer interface more attractable.

This document is going to cover many of the different technical aspects of this project. The technical challenges we are expecting will be outlined, as well as the main and alternative solutions to these challenges. Each solution will be analyzed and compared against the other potential solutions, to ensure that the best one is chosen. These solutions will be proven feasible as well. As integrating all of these solutions together to create a seamless and elegant software system, a technology integration investigation will also be conducted and included in this document. A final conclusion will wrap all of the pieces together that prove that this software system is feasible.

2. Technological Challenges

We as a team have outlined the main technological problems that we expect to face while developing this software system. These challenges are as follows:

2.1 Front End, Web Interface Requirements

A consumer is going to need a way to access the global prediction system in an easy to use way, so a web interface is going to be used for this aspect of the software. These are some of the challenges that we expect to face when implementing the web interface:

- A zoomable and scalable map should be present that the user can interact with
    - User is able to draw a polygon on the map designating the plot of land that they wish to process

- Ability to upload a shape file instead of drawing a polygon on the map
- Ability to send the polygon/shape file to the backend for processing.
- Ability to receive the results back from the backend, then display them to the user
- Responsive and fast while maintaining a positive user experience

2.2 Backend, Big Data Computing Requirements

As mentioned in the introduction, the current solution to this problem is not efficient enough, so computing and processing of the users input takes far too long. The desired solution to this is using the high performance computing cluster, Monsoon, although this might not be possible. Even if Monsoon is not chosen, the backend should be able to overcome these challenges:

- Creating an efficient and working global prediction system
    - Must be written in python
    - Raster layers need to be stored in the appropriate format
    - Processing times must be kept to acceptable levels, which will be determined by our sponsor
- Backend must be able to receive uploaded files and polygons from the web interface
- Backend must be able to communicate the computed results back to the web interface for display
- If Monsoon is chosen, we must use the different types of nodes on Monsoon to maximize the efficiency of our global prediction system
    - Login node, will be used for regular linux use (checking logs, scheduling jobs, etc)
    - Computation node, the global prediction system will be computed on these to minimize the processing time
    - Data transfer node, will be used to move large files more efficiently than the login or compute node
- Stretch Goal Challenges:
    - Building a module on Monsoon that can run the prediction
    - Further performance upgrades, tiling rasters
    - Develop an ArcGIS plugin

2.3 Communication, REST API Requirements

In order for the front and backend to communicate with each other, we need some sort of an API. Our project sponsor has decided that we will be implementing a REST API, so the web interface and the backend are able to interact with each other. Here are some of the challenges that will come with the API implementation:

- The backend must be able to communicate with the web interface and vice versa
- The API must be able to handle large data files efficiently
- Secure Encryption for security purposes

2.4 Database, User Account System Requirements

This system needs a user access control system, which means that user information must be stored somewhere. We are going to use a database to store this information. Some challenges that we expect are:

- We need to store user access information such as usernames and password
- All information stored in the database must be secured, as much of it is sensitive
- “Useful System” challenges
    - Store a user’s previous queries
    - Allow a user to manage their own account information
- “Stretch Goals” challenges
    - Allow tracking of user activity, so usage based billing can be enabled

2.5 Mapping and Feature Handling

The issues we anticipate running into with the mapping is how to handle the functionalities of using a map. Every good map has pan, zoom, and scale features. Some of the better mapping software even implements a polygon feature where you can draw and analyze parts of the map. For our project, we have to implement the following:

- Minimum Viable Product
    - The UI will present a zoomable global map to the user
    - It will prompt the user to draw or upload a polygon

- Stretch Goals
    - Further performance upgrades such as tiling rasters
    - Ability to select countries from a dropdown list instead of a polygon
    - Ability to upload shape files (.shp) instead of drawing polygons

2.6 Base Map API

Finding a base map that suits the needs of this project that implements some key features is going to be a challenge. Many companies have maps for the public to use but some block the use of additional libraries and plugins. Our challenge will be finding a mapping API that allows for external functionalities and expansion to implement certain necessary features.

3. Technological Analysis

Each challenge that was described above is going to be analyzed and solved in this section. Every challenge will be introduced, with multiple alternative solutions to each challenge. These solutions will be analyzed and compared against each other, until the best solution is clear. This chosen solution will then be proven feasible.

3.1 Front End, Interface

Our sponsor has requested a useful and user friendly web interface that is able to efficiently run our program. For this purpose, our group plans to use a combination of HTML, CSS, JavaScript and a web framework of some kind. Using these utilities in combination will allow us to easily communicate from our web page to our REST API.

3.1.1 Front End Interface Desired Characteristics

When designing our interface, we have a number of features we wish to implement. These features include a custom and secure login page as well as a dynamic map that users can draw on to create unique shape files that can be used for analyses.

- Minimum Viable Product
    - A Login page

  - A Map that can be drawn on to create shape files
  - Our interface should also be able to connect to our REST API for communication

- Stretch Goals:
  - Be able to view computation progress in real time
  - Our interface would allow for premade shape files to be sent and analyzed

3.1.2 Front End Interface Alternatives

An alternative solution to a web interface could be to create a mobile app interface using SWIFT and Xcode. This would change the platform our program would be accessed on to IOS /Apple. However, the functionalities would stay the same regardless of which interface is implemented. One downfall to making a mobile application would be that users would have to own an Apple product to download and use this program.

3.1.3 Front End Interface Feasibility

Our most difficult challenge in implementing this would be getting our interface to successfully communicate and connect to the REST API. However, python has many built in features that help send and receive requests making the process much simpler. Also in order for us to use all of these programs we would have to download some existing libraries. The most complicated to get working would be Node.JS, however once we download a package manager like NPM the process becomes simple and we can download the libraries through the command line.

3.1.4 Front End Interface Chosen Approach

Below is a table that compares and contrasts different aspects of the technologies for the front end interface.

Figure 3.1.4

Name	Linux OS (1 = yes, 2 = no)	Big Data Computing (1 = yes, 2 = no)	Cost (1 = Free 2 = Paid)	API Support (1 = yes, 2 = TBD, 3 = no)	Ease of Use (1 = easy, 2 = mediocre, 3 = hard)	Flexibility of hardware/software (1 = partial, 2 = full)
HTML/ CSS/	1	2	1	3	1	1

Node JS	1	2	2	1	3	2
JavaScript	1	1	1	1	2	2
Swift/ XCode	2	2	2	1	2	2

As a team we would like to implement a front-end interface that uses a combination of HTML, CSS, JavaScript and Node JS. This would allow our team to have full control over how our website works and how it communicates as well. Although it would be nice to have an IOS application on the app store, our program will still be able to be ran on a mobile device through the internet. This is why we have decided to stick with a website based front-end interface.

3.2 Backend, Big Data Computing

This project demands running a global prediction system that uses massive amounts of data. These results will need to be fed to the front end, so they will need to be computed hastily, so the user is able to have a responsive and pleasant experience. This means that a high performance computer is required.

At the time of writing this section of the document, the backend is still up in the air. Our project sponsor's original vision was to use Monsoon, but Monsoon may not support API's, or real time computing. Because of this, we are not going to be choosing a backend in this paper, as it is blackboxed at this time.

3.2.1 Current Backend's Issue

As stated in previous sections, the current prototype that is being used is simply too slow. The current global prediction system that is implemented is not efficient enough. This results in a clunky, unresponsive, and unpleasant web interface, so this is going to have to change.

3.2.2 Desired Characteristics of the New Solution

The main characteristic that we desire in our new solution is that the global prediction system is being efficient and fast. The user should be able to get their computed and processed data within an acceptable time, which will be determined by our sponsor. Other desired characteristics for our solution include being accurate, maintainable, and extendable. As our project sponsor may continue to develop this software after our project is over, we need to ensure that carrying on the development of the software is seamless and problem free. Our results also need to be accurate and correct, as to maintain the integrity of the software.

3.2.3 Backend Alternatives

As mentioned in previous sections, the backend is still up in the air. The two solutions that have been proposed to us by our project sponsor so far are Monsoon and AWS.

Monsoon is the high performance computing cluster, or supercomputer, made available to researchers at NAU in 2014. Monsoon is mainly used for intensive research computing projects, and not so much real time computing, which may pose issues to this project.

AWS, or Amazon Web Services, is a web server hosting service provided by Amazon that makes cloud computing with big data simple. A user is able to rent out an AWS instance, or server, to do almost anything that they want to do. In our case, we could use AWS to run the global prediction system, as well as using our API to bridge the gap between the front and backend.

3.2.4 Backend Analysis

Both Monsoon and AWS are able to fulfill the needs of this capstone project, but each of these options have their own pros and cons. The main concern of Monsoon is not being able to host our API for real time computing, while the main concern of AWS is that it can be quite expensive to rent an AWS server.

As Monsoon contains extremely powerful hardware components, as well as sophisticated software components, it will be able to cut down the computing time compared to the current prototype. Monsoon's hardware details are located here.

Monsoon also has multiple different node types such as the login node, the computing node, and the data transferring node. Each of these nodes has their own purpose and speciality. The login node is a simple node that allows a user to gain access to Monsoons command line. These login nodes will just be used to do basic actions such as checking job status, checking logs, scheduling jobs, and writing the software. The computing nodes will be used to actually run the global prediction system. These computing nodes have more hardware dedicated to them compared to the normal login node, which speeds up the computing time. Finally, the data transferring node will be used to transfer large files, such as the raster files. Some of these files can be extremely large, so using these data transfer nodes can improve the efficiency of our data transferring.

The final aspect of our Monsoon implementation is the communication from Monsoon to the web interface and vice versa. Monsoon uses the linux operating system, so being able to set up our REST API should not be a big deal, if the maintainers of Monsoon would allow real time computing. Currently, this is the biggest concern of using Monsoon versus other options, like AWS. Monsoon is also free to use for research purposes here at NAU, so this is a major positive.

AWS is a cloud computing service that is hosted by Amazon. AWS supports the linux operating system, just like Monsoon. AWS can also efficiently compute with big data, which is necessary for this project. AWS also has many different options on the type of server that we could rent, giving us more flexibility. Because of this flexibility, finding a server that provides all of the needed services for this project is not a big issue.

Both Monsoon and AWS have been tested by some of our group members, so they are each viable options. While we have not done big data computing in either of these so far, smaller demos have been done, which proves the feasibility of both of these options.

3.2.5 Backend Chosen Approach

As stated in the previous sections, we are still unsure of which backend is going to be chosen for this project. This section of the analysis is going to be blackboxed, as we are still waiting on our project sponsor to choose the desired backend. Both Monsoon and AWS provide the services that are necessary for this project, but they do have their own pros and cons. The table below compares both Monsoon and AWS, and their pros and cons.

Figure 3.2

Name	Linux OS (1 = yes, 2 = no)	Big Data Computing (1 = yes, 2 = no)	Cost (1 = Free 2 = Paid)	API Support (1 = yes, 2 = TBD, 3 = no)	Ease of Use (1 = easy, 2 = mediocre, 3 = hard)	Flexibility of hardware/software (1 = partial, 2 = full)
Monsoon	1	1	1	2	2	1
AWS	1	1	2	1	2	2

The main difference between AWS and Monsoon is that AWS is paid, while Monsoon is free. AWS server instances can be quite costly, so this is the main downside of using AWS. The main downside of Monsoon is that it may not allow API's and real time computing. Our project sponsor is going to be choosing the backend for this project.

3.2.6 Proving Feasibility, Backend

One of our group members is currently working on an undergraduate research project that uses Monsoon, so we have already proven some of the feasibility of Monsoon. This member has been able to login to Monsoon using the login node. They have also been able to access the computing nodes. These basic tests show that accessing and using Monsoon is feasible. Later in the semester, we will need to create a very basic API to ensure that the API is feasible on Monsoon. We are also going to need to create a rough outline of the global prediction system on Monsoon as well.

During previous computer science classes, our group members have been able to use and experiment with the free tier of AWS. While this is not going to be the exact server instance that we use if we use AWS, having this experience in AWS proves that it is feasible for this project. After doing lots of research, we have discovered that AWS also supports big data computing, which will be used during this project.

3.3 Communication, REST API

Our sponsor has requested that we create a REST API that will facilitate communication between the backend and the user. This API will need to be located in the backend to allow people to tie directly into the service we are providing without the need to use the web interface.

3.3.1 Communication Current Issue

As stated in previous sections, the current prototype that is being used is created and set up to communicate within Google's ecosystem. However, in order to improve the speed and efficiency of the solution we need to have our own backend and that creates the issue of how it will communicate with the front end to return the data and results to the user. We can solve this issue by developing a REST API that will facilitate the communication between the backend and the user / web interface. In order to create a useful REST API we need to choose the language that we are going to use to create it and the framework that will assist us in development.

3.3.2 Desired Characteristics

In order to have a useful REST API we need to make sure that it can be run on the backend so that the user can have the option to tie directly into our service or use our provided web interface. The API is also required to have the ability to transfer shape and raster files between the user and the backend.

3.3.3 Rest API Language Options Analysis

Node JS

Node JS is widely used to build fast, highly scalable network applications. Node JS has a lot of uniformity, readability, and consistency in the way it displays and renders the code. With the way this language is created, it allows developers to write API's quickly with relatively little challenge. This language suffices all the needs of the development of API's in a scalable manner. This will support high performance and robust routing and http helpers such as rendering, setting headers, etc. In the team, we have some experience in using Javascript with Node JS from our web development classes we have taken. However, it would be a language that we would need to learn more of to implement the REST API.

PHP

PHP is a server-side scripting language that was created for the development of web applications. It can also be used as a general-purpose programming language. This language can be inserted into the html code of the web front end or it can be separated from the html as its own executable. This allows a bit of flexibility with this language for the development of this REST API. The members in our group have some experience in PHP, but in the grand scheme of things it is very minimal. This would lead to us needing to learn more about the language and how it can be used to implement the API.

Python

Python is a powerful high-level, object oriented programming language that is known for its ease of use and high readability. It has a wide range of applications such as web development, scientific and mathematical computing, and desktop graphical user interfaces. Being such a clean and relatively simple programming language, it allows the programmer to focus on the problem rather than the syntax. Everyone in our project team also has sufficient knowledge of the programming language and experience in using it to implement previous projects.

SQL

SQL is a strong programming language that can be used to create and search databases. SQL is a relatively simplistic language but still gives the writer the opportunity to write any type of table search (simple or complex). Because of these abilities, SQL is one way we can interpret information between our REST API and database.

3.3.4 Framework Options Analysis

Flask

Flask is a lightweight web application framework designed to make the process of getting started easy. Flask allows for quick scalability as well. Flask-RESTful is an extension to Flask that provides support for creating and maintaining REST API's built using Python and/or SQL. Being a framework, Flask encourages best practices in creating an API with minimal setup.

Express

Express is to be used alongside Node JS. Express is a minimal and flexible Model-View-Controller (MVC) framework that includes a powerful collection of features for web and mobile application development. When it comes to writing APIs in Node JS, Express is more or less the best suitable framework.

Laravel

Laravel is an open-source web-based application structure that contains exquisite and expressive grammar that can ease the development of web applications and make it more available and captivating. Laravel is known as the top PHP framework when it comes to developing PHP based applications.

Django

Django is known as one of the best tools for developing REST API's. It is a framework for developing web applications using the Python programming language. This framework makes it easy to develop a working API in a relatively small timeframe and there are many tutorials on its website. This framework also comes with many built-in features that ease the development of an API. It also contains a built-in functionality that allows you to browse your API that is styled automatically with a clean and modern design. The design also contains multiple pages so that the data can be browsed more easily.

Figure 3.3

Name	Ease of use (1 = difficult 5 = easy)	Previous knowledge (1 = none, 5 = experienced)	Readability (1 = difficult 5 = easy)	Best Framework for the language	Ease of Use of framework (1 = difficult 5 = easy)	Total Score (X / 20)
Node JS	3	2	4	Express	4	13
PHP	3	3	3	Laravel	4	13
Python	4	4	5	Django	5	18
SQL	3	3	4	Flask	4	14

3.3.5 Language and Framework Decision

When comparing the programming languages above we concluded that we should use Python. This is due to the fact that it is a language that everyone in the team has experience with, and it is the language that our backend will be written in. This commonality as well as the many useful libraries it contains lead us to choose Python as our programming language. In choosing Python, we also need to choose a framework that will allow us to create the API with many features and ease of use. Looking at the options for Python based frameworks we concluded that Django is the framework that we are going to use as it is well respected and highly praised in industries. In addition, all the features it offers will assist us in creating the best REST API for our project.

3.4 Database, User Account System

A simple database management system is going to be required because we will need to implement a user access control system. In order to store user data, we are going to need a database.

3.4.1 Current Issue

Along with the process of calculating carbon credits, we need to create a database to store user data along with their possible queries. To implement the storage and access of user data, there are many different options we can use; however, some database systems are better suited for what we need than others. Three options we are considering for storing data are MongoDB, mySQL, and PostgreSQL.

3.4.2 Desired Characteristics

The database we choose to store our user data must have certain qualities/features that are required for our product. It must be able to efficiently store and access user data, provide ease of use in regards to granting user permissions, store data securely (encryption), as well as providing a way to scale the size of the database smoothly. There are other undefined characteristics that may be helpful also, such as spatial queries and data structure types.

3.4.3 Analysis

Each one of these database management systems differ from one another and offer unique advantages. MongoDB is a non-relational (noSQL) database system that stores and represents data in the form of JSON-like documents, which is much different than the standard table and row format, used by mySQL and PostgreSQL. A main benefit of using PostgreSQL is its flexibility in allowing users to store information in collections with no enforced schema, and how it is easily understandable to anyone with programming experience. MongoDB supports unstructured, semi-structured, or structured data; this will be very helpful when we begin implementation of a database system, because we can figure out as we go what kind of structure we want to use to hold our data. Along with data storage, MongoDB utilizes a role-based access control model that grants different users certain roles, which gives them permissions over particular database features. It is also encrypted with TLS (Transport Layer Security) and has encryption of data at rest. These security features make it an optimal choice when discussing user roles, as our sponsor will need to grant permissions to each individual user. MongoDB is also very easy to scale- with their "sharded cluster" implementation it allows the database to horizontally scale both read and write performance to cater to applications of any scale. If and when we will need to scale up our database, we can also upgrade to the MongoDB Enterprise level, which includes other features such as advanced security options (auditing, log redaction, Kerberos, LDAP), additional storage engines (encrypted and in-memory), and advanced management tools (Cloud/Ops Manager).

The MySQL database system also provides a compelling way to store/access data. MySQL is the standard RDBMS system, storing information in tables that consist of columns and rows. It is highly structured and must match the schema in order to be implemented correctly. While this is not as flexible as storing data in JSON files, it does offer a higher level of security due to the rigidness of the structure. To start, like MongoDB, we will be using the open source mySQL community edition server to implement our first iteration of a successful database. While not as scalable as MongoDB, mySQL does offer a similar "sharding" method that can enable databases to scale horizontally. However, this method requires more maintenance efforts because of the normalization that occurs with relational database tables. MySQL is not JSON oriented and uses structured query language, so it will not be as easy to understand at first when compared to

MongoDB, but all five of our team members have taken a database class centered on SQL, so it should not be too difficult. While not as robust as the enterprise edition of MySQL, the community edition does offer viable layers of security. It offers at-rest and in-flight encryption using SSL (Secure Socket Layer) which establishes an encrypted link between a server and a client. The at-rest data encryption features file data, redo logs, and undo logs which offer a layer of encryption while the data is stored. All of this will act as sufficient security and encryption for user data while the software is being tested to ensure that the database works. Once we are finalizing our product, we will most likely switch to a paid version of mySQL, that being the enterprise edition, similar to MongoDB's enterprise edition. It offers more features than the unpaid version, such as enterprise monitoring, backup, security, scalability, and high availability. Its encryption service is also much more secure, including:

- Asymmetric Public Key Encryption/Decryption
- Symmetric Public and Private Key Pairs
- Public/Private Key Generation
- Digital Signature Verification and Validation
- DSA, RSA, and DH-type Encryption Algorithms
- Transparent Data Encryption

Along with encryption, MySQL offers different plugins for auditing the database. Depending on what we will need to audit (database, server, user, etc.), we can use different plugins for our needs. ClusterControl and Cloud SQL automates security audits across the database, which would be helpful and beneficial in tracking user data, preventing hacker attempts, and providing an automated authorization service. Depending on how important high levels of encryption are in storing the user data, as well as how the data is stored, mySQL will be able to provide all of the support we need to properly store user data, their queries, and personal information.

The final database system our team is considering is PostgreSQL. PostgreSQL, like mySQL, is a RDBMS; however, one key differentiating factor is that postgreSQL uses objects, making it an object-relational database. With it being an ORDBMS, postgreSQL makes it possible to extend data types to create your custom types, while still adhering to the SQL structure. PostgreSQL supports foreign keys, stored procedures, joins, and views in several languages, and also includes various data types and supports the storage of large objects, such as pictures, sounds, as well as videos. PostgreSQL also features high extensibility, meaning future growth will be an easy task. Another feature that makes postgreSQL an attractive option is the fact that it can easily integrate and function in frameworks like Django, which is our framework of choice. PostgreSQL offers many of the same security features as mySQL, however it is ACID compliant, which ensures that database transactions are processed reliably. Compared to mySQL, PostgreSQL is best suited for systems that require execution of complex queries, or data warehousing and data analysis, meaning we will be able to efficiently analyze the data that would potentially be stored from user queries. Finally, a main factor that sets postgreSQL apart is its ability to execute spatial queries with the postGIS extension. This is a big differentiating factor because this will allow us to store spatial data, create and store spatial shapes, determine routes, and calculate areas and distances with the data sent back from user queries. Overall, postgreSQL is a very compelling option to use as our database backend with all of the features that it offers.

	MongoDB	MySQL	PostgreSQL
Database Type	Non-relational Document DBMS	Relational DBMS	Object-relational DBMS
Data Structure	JSON files	Relational Tables	Objects
Encryption?	Yes	Yes	Yes
Scalability?	High	Low	Medium
Spatial Queries?	Yes	No	Yes
Compatible Frameworks	Laravel, Django, Angular, Flask, Express, Spring boot	Flask, Express, Laravel, Django, .NET based	Django, Drupal, Cake, Mambo, Joomla, .NET based

3.4.4 Chosen Approach

Based on the different offerings listed above, the best choice for our product is the PostgreSQL database management system. It offers both relational and object based structures, excellent security and scalability, as well as an efficient and intuitive spatial query extension. It is also compatible with the Django framework, which is our chosen framework for this project. Overall, it will be perfect for the needs of our project, and will provide sufficient resources for the storage of user data and their queries, all while keeping security a top priority.

3.4.5 Feasibility

The feasibility of our project successfully using postgreSQL to manage user accounts and information is very high. Due to postgreSQL being highly compatible with Django, and its implementation of spatial queries, it is a very good option for our database system. Overall, implementing postgreSQL with our front end and backend is very feasible because of how flexible and compatible it is.

Queries and personal information related to billing and other are considered stretch goals for this product.

3.5 Mapping and Geographical Libraries

In order to use a base map and all the features that come with a great online map, we will use a javascript mapping library. This library will allow the consumer to use a map on our website just like any other site like google maps or bing maps.

3.5.1 Desired Characteristics

This library will implement the scrolling, zooming, and layers, as well as the polygon drawing. As specified by our sponsor, and for general functionality, our web map should/must be able to scale and zoom across the globe. This will allow the consumer to research and plan projects all across the globe rather than just focusing on one country or part of a country.

3.5.2 Analysis of Libraries

We have explored several different libraries to use and considered Openlayers or Polymaps. Both are open source javascript libraries that allow for easy raster and vector data manipulation as well as handling the shape files. These libraries both have very good documentation that will make it streamlined for us to implement into our webpage. Our plan for these libraries is to use them to handle the display and functionalities of the map on the page as well as implement the shape files and polygons to grab data that we will in turn send to the backend for processing and carbon credit prediction. Both Openlayers and Polymaps are very similar, but will be explained further in the next sections.

3.5.2.1 Openlayers

Openlayers is a javascript library for mapping and geographical functions. This library provides several basemap layers in both vector and raster data tiles. Openlayers provides several functions for handling raster and vector data as well as handling polygon drawings. This library also has extremely good documentation. Thus, it will be fairly easy to implement for our needs. One downside to Openlayers is that they implement their own styling methods as opposed to typical CSS styling.

3.5.2.2 Polymaps

Polymaps is also a mapping and geographical library for javascript. Polymaps advertises that they handle mainly vector data and SVG (scalable vector graphics) files, but it also handles raster layers and data. Polymaps does also handle polygon functions and data handling. It implements standard CSS styling. Therefore, implementing it into our webpage will be very intuitive.

3.5.3 Chosen Approach

While our sponsor Allie will be providing us with raster layers and data to use, if he would ever like to handle vector data in the future he would be able to with both of these libraries. Since vector data is not specified at the moment, the possibility of this will be blackboxed. As of now we will implement Openlayers as our mapping and geographical library. Figure 3.5 shows the libraries compared in a table.

Figure 3.5

Name	Raster handling (1 = yes, 2 = no)	Vector handling (1 = yes, 2 = no)	Styling (1 = CSS 2 = proprietary)	Bing Compatibility (1 = yes, 2 = with wrapper, 3 = no)	Google Compatibility (1 = yes, 2 = with wrapper, 3 = no)	Documentation (1-5 scale, 5=best)

Openlayers	1	1	2	1	2	5
Polymaps	1	1	1	1	3	3

3.6 Base Map API

Regardless of the mapping library we choose we still have to choose a base map to display. The two main ones we are considering are Google maps API and Bing maps API. Both implement their own features and have geolocation features. Allie, our sponsor, runs his prototype prediction software on Google Earth engine, which works but it is extremely slow because of having to wait in Google's computation queue.

3.6.1 Analysis of Map API

3.6.1.1 Google Maps

Google maps API is very restricted on what you can and cannot do because they restrict the use of external libraries and plugins. This means that we wouldn't be able to use the above mentioned mapping libraries without some extra work around. Although, Openlayers does include and implement a wrapper for the use of google maps.

3.6.1.2 Bing Maps

Bing maps on the hand works natively with Openlayers and Polymaps. Bing maps advertises that they have high accuracy geolocation. With what we are having to do, we want the most accurate geolocation system to provide the best prediction of carbon credits for the consumer. Bing maps API is free for the developer but when Allie does turn this into a business it would have to be switched to the enterprise license.

3.6.2 Chosen Approach

Openlayers does include some base maps that allow for basic services but if we want to implement the geolocation from Bing or Google. For the reason of Google blocking the use of external libraries we are going to use Bing or the built in maps that Openlayers provides.

Figure 3.6

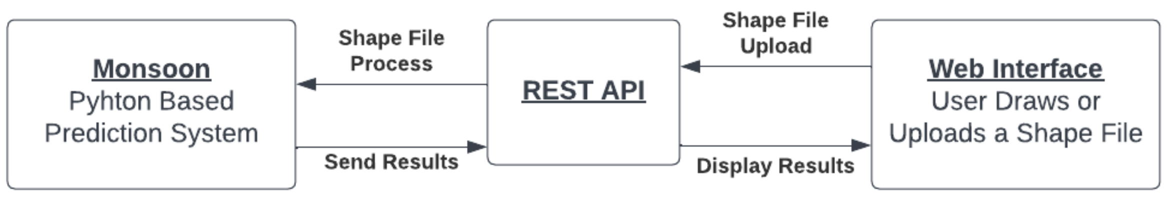
Name	Geolocation (1-5 scale, 5 = best)	Compatibility (1-5 scale, 5 = best)	Recognizability (1-5 scale, 5 = best)	Expandability (1-5 scale, 5 = best)
Google Maps	3	1	5	1
Bing Maps	5	4	3	5

4. Technology Integration

When we integrate all of the different parts of our software system, we are expecting to have three main software components that will interact with each other. These three are the front end web interface, the backend, and our REST API to allow communication between the front and backend. The front end will allow for a user to draw their desired plot on a map containing the plot of land that they wish to process. The API will then send the data to the backend so that the global prediction system may determine the amount of carbon credits for the given plot of land. Once the prediction system is done computing, which must be kept to reasonable levels, the backend will return that data to the API. The API will then send the computed results back to the web interface for the consumer to view and use. The consumer can then use that estimate or continue planning different areas for development. Figure 4.1 shows how our system will be implemented.

Figure 4.1

- The general architecture of our envisioned system.



5. Conclusion

The problem of climate change is not one that is restricted to certain parts of the world, or certain populations; it is a problem that affects everyone around the globe no matter where you live. As people are becoming more aware of the threat that climate change poses, many have pointed fingers at big companies such as Microsoft, Facebook, Amazon, etc. due to the large amounts of carbon they contribute to the atmosphere. In response, these companies have turned towards the market of carbon credits, where they can buy “credits” that offset their carbon footprint. While this is a legitimate and proven way to reduce the carbon footprint of these companies, it is not as appealing as one would think. The amount of money they have to invest compared to the time it takes to make a return on their investment is about seven years, leaving a large window where they are “in the hole” on their investment. This is the problem our team and Dr. Shenkin are aiming to solve; with the discoveries made by Dr. Shenkin and his team, along with our software, we will be able to increase the profitability of carbon credits by up to 30%. Along with this, we will be providing a clean and user-friendly interface to calculate the amount of carbon credits one would earn per plot of land, which uses a web interface front end that connects to our backend. With the new accessible software combined with the extra 30% companies would make purchasing carbon credits, we hope to incentivize large and small companies to buy more carbon credits to offset their footprint, therefore reducing the amount of carbon in the atmosphere, and reversing climate change.

The software our team, Clean Carbon, is developing will act as a way for companies to see firsthand how much carbon credits they would receive when outlining a certain plot of land. They will first access the website we will develop, and either log in to their account, or create an account for use. This will be made possible using HTML and CSS to structure our website, along with using MySQL to host and interact with our database of users. We will build our prototype on the community version of MySQL, and later upgrade to MySQL enterprise edition if needed. If possible, we will also implement a way to store previous user queries so they can refer to them later on. Once they have accessed their account, they will be brought to a page that will allow them to outline a specific area on a map to calculate their carbon credits. They will do this by using the Bing Maps API or the map integration that the OpenLayers API provides. Once they have outlined their chosen area, that information will be sent to the Monsoon supercomputer on campus to be interpreted and calculated. It will then be sent back to the user in the form of usable data, including how many carbon credits they can expect to obtain. This transaction will be made possible by the integration of a REST API, which allows us to connect our front end with our backend. All of this will result in the user receiving useful information about how many carbon credits they can expect to get by investing in a certain plot of land in a potential area for reforestation.

Our plan to integrate Dr. Shenkins research with our software will create a path towards a more sustainable future. Providing users with a straightforward, simple, and efficient way to calculate their carbon credits will undoubtedly incentivize more companies to invest more into reforestation. Along with this, they will also be benefitting from the new research by Dr. Shenkin and his team that allows for 30% more carbon credits per plot of land. Both of these advancements will act as a catalyst for a large increase in reforestation, which will in turn limit the amount of greenhouse gasses going into the atmosphere and slowly but surely reverse climate change.