4.2 Design Exploration

4.2.1 Design Decisions

List key design decisions (at least three) that you have made or will need to make in relation to your proposed solution. These can include, but are not limited to, materials, subsystems, physical components, sensors/chips/devices, physical layout, features, etc. Describe why these decisions are important to project success.

One of the design decisions our group had to navigate was choosing which application(s) the end product would be accessible over – like, for example,whether that be a web-based, desktop, or mobile application, as this will ultimately heavily influence both the level of work required for implementation and the accessibility/useability of the project. Similarly, a decision involving the acceptable formatting of entered data files needed to be made, with complementary concerns as: on the one hand, limiting different acceptable file types would decrease complexity but , on the other hand, it would also decrease end user accessibility and ease of use.

When analyzing data and utilizing entered data for processing specific queries, necessary considerations needed to be made regarding the back end library and compiler, as these aspects can have an instrumental role in determining the maintainability, scalability, performance, and overall complexity for the application. Complementary to this, decisions for the frontend also had to be made regarding the certain primary concerns, as well as regarding the issues of ensuring compatibility between both frontend and backend.

A different category of concerns that needed a consideration related to the overall design pertained to the structure of the backend from the perspective of the types of queries the end user would utilize most frequently. These were also necessary for ensuring that the overall performance and scalability would not suffer due to particular types of queries.

While considering what functionality was necessary for the front end visualization, several decisions had to be made regarding what basic tools will be accessible to the user, such as pan, zoom, and move, as well as more advanced settings/configurable options such as selecting specific cuts/sections of the visualization, transparency and color options. The main trade off being again added complexity for user versatility and accessibility, but also performance while these tools are in use within the high level user interface.

4.2.2 Ideation

For at least one design decision, describe how you ideated or identified potential options (e.g., lotus blossom technique). Describe at least five options that you considered.

For the design decision of choosing the web application framework, we utilized the lotus blossom diagram/technique. Based on this, we were able to narrow down our search of which framework/language we planned to utilize. We identified several different potential candidates for what we could use on the frontend or backend framework through open brainstorming. We ensured that all possible ideas were heard from the entire team in order to thoroughly analyze and consider all types of options available.

Java- Containing a vast array of libraries, tools, different potential frameworks, and scalability, as well as several team members having experience with this language.

Javascript- Utilization on both frontend and backend would allow for simpler implementation, but by itself, with the team having extensive experience with this language, along with extensive resources and high performance, it's clear why this is one of our top options.

C#- Both incredible speed and scalability come at the tradeoff of limited libraries compared to other options and unfamiliarity with the group.

HTML- While offering great compatibility with modern web browsers, limited functionality and lack of scalability could offer unique challenges if utilized.

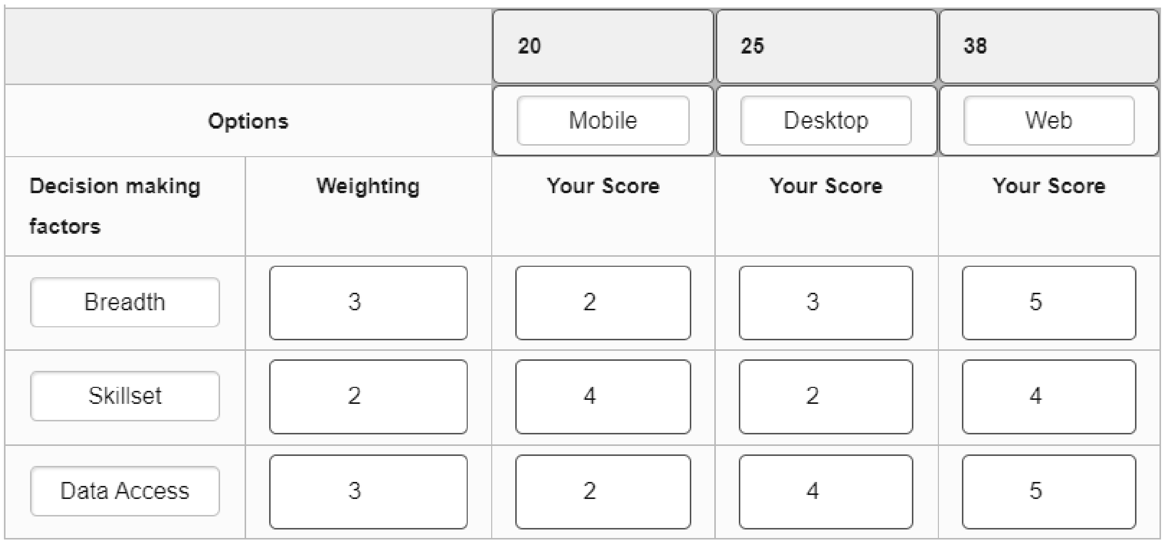
Python- While offering both clean syntax and the most extensive/available libraries, simplicity and flexibility, the drawbacks of lack of group experience with projects would prove to be extremely detrimental to progress given our agile methodology.

By exploring these different options via ideation techniques - i.e., the lotus blossom method, we were able to evaluate the pros and cons of each approach and make informed decisions aligned with our project goals and team capabilities.

4.2.3 Decision-Making and Trade-Off

The main ideated options were related to the application format, including mobile, desktop, and website. With these options, we were able to quickly narrow down our options to desktop or web-based, since a majority of our users will have data sets on their computers. Since data sets are relatively large, it would be impractical to utilize a mobile app as the primary format since most users will not have access to research data on their phones. Additionally, mobile applications for Apple and Android devices utilize different languages, so we would likely only have the breadth to choose one. This would be less accessible to diverse categories of users and limit the functionality of our application.

Between the desktop and website options, we determined that a website will be most accessible and effective since it can be accessed from any type of computer. It is also practical since users will not need to download software permanently and can instead access their visualizations from any device with an Internet connection. When looking at the skillset of our team, it was also apparent that either a mobile or web application would be most suitable for the skills we had developed. See Figure 1 below for the Weighted Decision Matrix accounting for the factors described above.



		20	25	38
Options		Mobile	Desktop	Web
Decision making factors	Weighting	Your Score	Your Score	Your Score
Breadth	3	2	3	5
Skillset	2	4	2	4
Data Access	3	2	4	5

Figure 1- *Weighted Decision Matrix for Application Format*

After determining the format we would use, we also had to take into account the languages, libraries, and compilers we would primarily use in our design. We ended up choosing SpringBoot for our backend on the Intellij compiler, since some of our team members utilized these tools in previous projects. For the frontend, we ended up deciding on Node.js and Javascript languages since they were most similar to other languages we had knowledge of and were compatible with our backend choices.

4.3 Proposed Design

4.3.1 Overview

The current design for our web-application consists of five primary subsystems and a few secondary or helper subsystems. The primary subsystems are as follows:

1. User Authentication/Registration - Allow users to create/login to their account to access private dataset previously uploaded.
2. Data Input & Management - Allow users to upload new datasets and specify the format to ensure proper parsing.
3. Whereabout Query Creation & Validation - Allow users to interact with their data and format whereabout queries.
4. Whereabout Algorithms - Performs calculations based on a submitted whereabout query.

5. Visualization - Plot data and whereabout algorithm to a window for the users to interpret.

Secondary subsystems include:

1. Database Query Engine - Given user information such as credentials and datasets, fetch and store results as required.
2. Whereabout Algorithm Result Packager - Take the raw results from the algorithm and package it in a way that is compatible with the visualization tool.

Most of the primary subsystems have both a user interface component seen by the user, and a hidden portion that operates behind the scenes. This is true for all except the whereabout algorithm, completely hidden, and the visualization, entirely visible subsystems. Together these systems work together to produce the web-application. Figure 2 shows several high-level components of different user-interface functionalities displayed to the user.

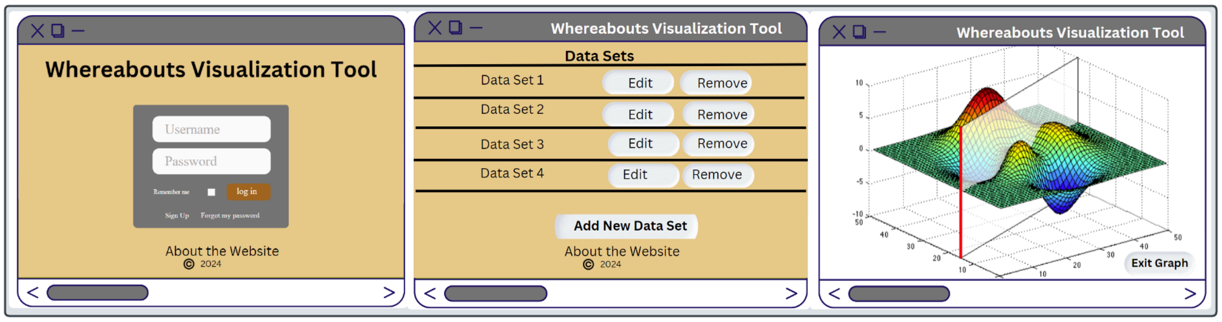


Figure 2- *Several High Level Functionalities; User Perspective (Visible)*

4.3.2 Detailed Design and Visual(s)

Describing the design in detail requires a more in depth view of how the subsystems, introduced in section 4.3.1, are interconnected and joint functionalities. Figure3 below illustrates the interconnections between blocks of which are color coded on the basis of the subsystem they belong to; see included subsystem legend. You will also see the before mentioned separation of visible and non-visible components denoted by the larger categorization of frontend and backend. Where frontend contains predominantly visible user interface elements, and backend comprises hidden functionalities or resources.

In further explanation of the diagram we are making two assumptions, firstly that all components falling under frontend and backend categories will be developed by the same groups of individuals, as mentioned in the task decomposition section of design document 3. Secondly all components in their respective front or back end category can readily pass information to one another. For these reasons we will not describe intra-categorical relationships in detail and will direct our attention to the inter-categorical relationships, the backend API.

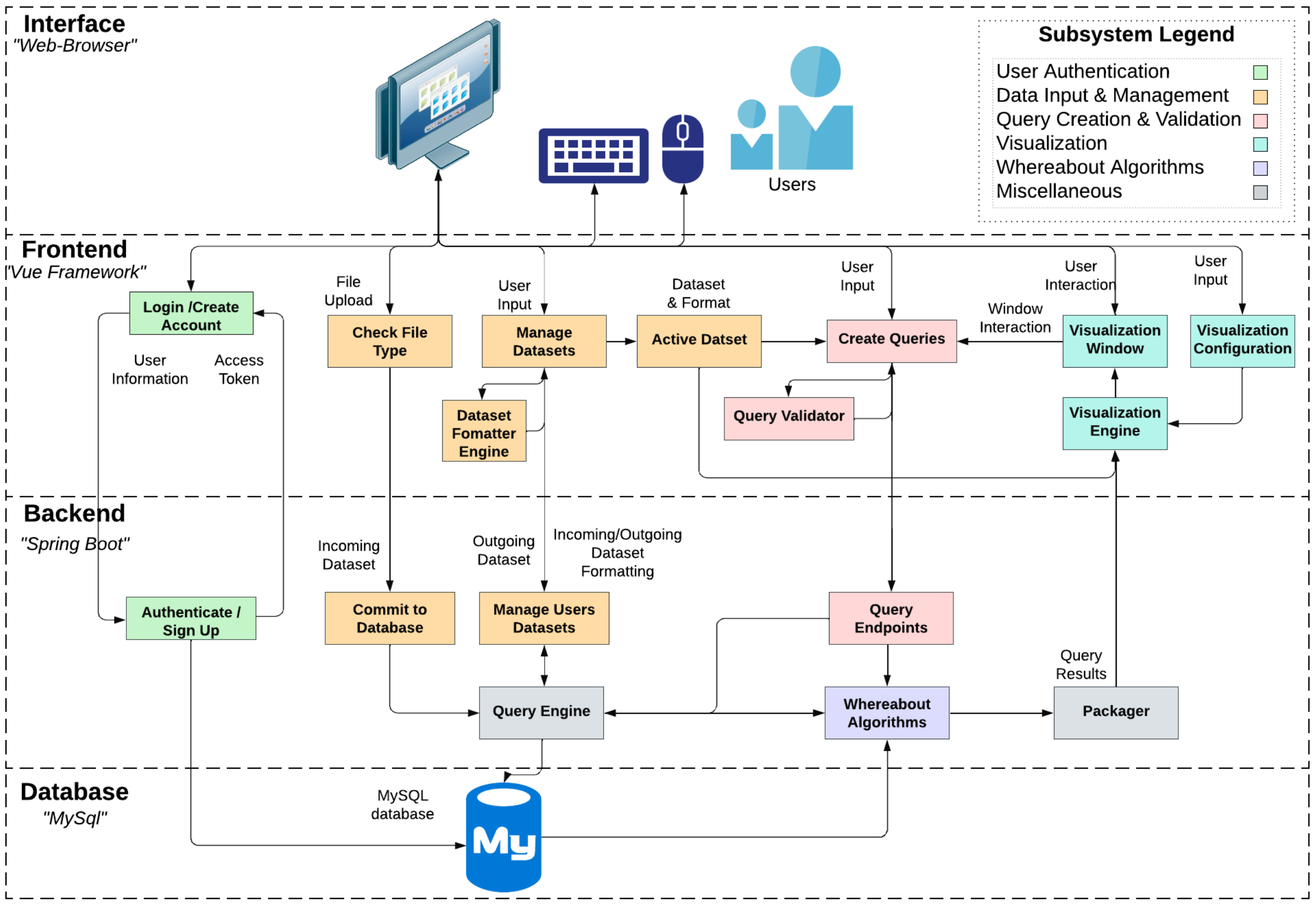


Figure 3 - *Systems diagram*

Referring again to Figure 3, there are essentially 6 lines or pathways that bind the front and back end together. Symbolically, they resemble HTTP request methods, and are used to request and send data between the front and back end. In order to effectively create a layer of abstraction, well defined data structures have been proposed for the 6 pathways. Seen another way, the 6 links can be grouped into three bi-directional links and related to the subsystems- they include user authentication, data input and management, and whereabout queries and results.

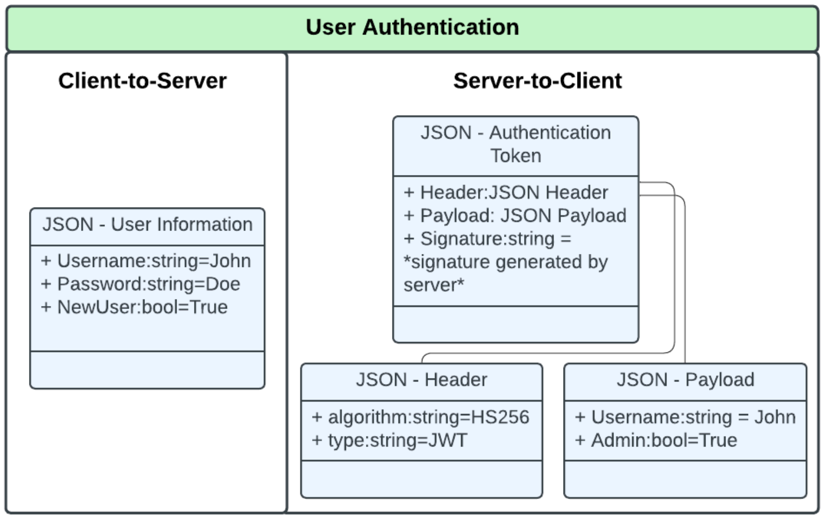


Figure 4 - *User Authentication JSON Objects*

Information passed over a link is structured using JSON objects, where each entry in an object comes in a "key" "value" pair. Additionally, objects can be nested within parent objects, allowing for complex data structures to emerge. Figure 4, shown above, gives the object details for the information being passed during user authentication. The user submits their username and password to the server. The server fetches user information from the database and if the credentials match the submitted input the server generates a unique signature and embeds it within the JSON authentication token to be returned to the client. In subsequent requests for information from the client requiring authorization to access, the client will provide token information.

Slightly more involved objects are necessary for the data input and management links. In reality there are three routes for information – the loading of the raw dataset in its entirety, the passing of the dataset format, both of which are moving from client-to-server, as well as a parsed dataset returned by the server to the client. Three routes require three JSON objects to carry this information, laid out in Figure 5 below.

The user uploads their dataset through interface interaction, which can be checked locally in frontend for initial verification; this information is passed using the "Raw Dataset" JSON object. The backend will store all information provided in the file. The user will next specify the layout of the dataset (i.g. What rows and columns contain relevant information for analysis, and how the application should interpret the time stamp). This will be stored alongside the initial dataset within the database.

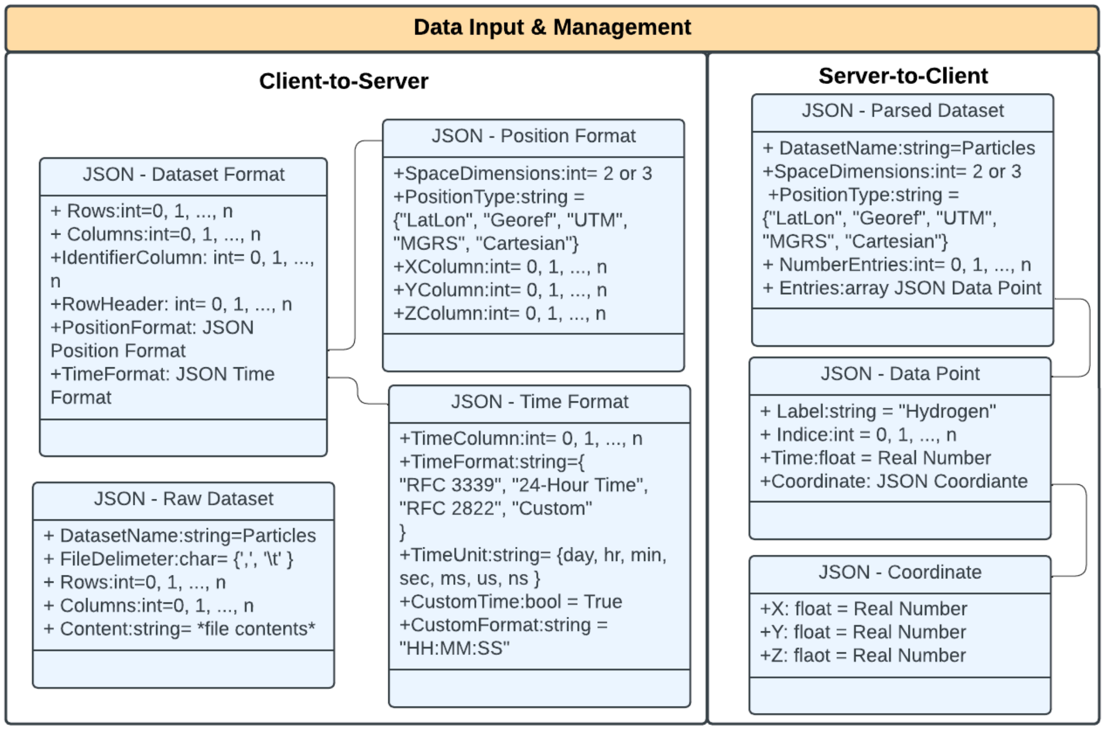

Data Input & Management
Client-to-Server
Server-to-Client
JSON - Dataset Format
+ Rows:int=0, 1, ..., n
+ Columns:int=0, 1, ..., n
+IdentifierColumn: int= 0, 1, ..., n
+RowHeader: int= 0, 1, ..., n
+PositionFormat: JSON Position Format
+TimeFormat: JSON Time Format
JSON - Position Format
+SpaceDimensions:int= 2 or 3
+PositionType:string = {"LatLon", "Georef", "UTM", "MGRS", "Cartesian"}
+XColumn:int= 0, 1, ..., n
+YColumn:int= 0, 1, ..., n
+ZColumn:int= 0, 1, ..., n
JSON - Raw Dataset
+ DatasetName:string=Particles
+ FileDelimeter:char= {',', '\t' }
+ Rows:int=0, 1, ..., n
+ Columns:int=0, 1, ..., n
+ Content:string= *file contents*
JSON - Time Format
+TimeColumn:int= 0, 1, ..., n
+TimeFormat:string={ "RFC 3339", "24-Hour Time", "RFC 2822", "Custom" }
+TimeUnit:string= {day, hr, min, sec, ms, us, ns }
+CustomTime:bool = True
+CustomFormat:string = "HH:MM:SS"
JSON - Parsed Dataset
+ DatasetName:string=Particles
+SpaceDimensions:int= 2 or 3
+PositionType:string = {"LatLon", "Georef", "UTM", "MGRS", "Cartesian"}
+ NumberEntries:int= 0, 1, ..., n
+ Entries:array JSON Data Point
JSON - Data Point
+ Label:string = "Hydrogen"
+ Indice:int = 0, 1, ..., n
+Time:float = Real Number
+Coordinate: JSON Coordiante
JSON - Coordinate
+X: float = Real Number
+Y: float = Real Number
+Z: flaot = Real Number

Figure 5 - *Data Input, Formatting, & Parsed Dataset JSON Objects*

Given the "Raw Dataset" and "Data Format" objects the backend will produce a parsed dataset object to be sent to the client. It will contain nicely packaged data points alongside additional information relevant to the visualization window.

The last of the links to be explained are for that related to whereabout queries and the corresponding results returned after completion. This portion may be the most obfuscated for the layman viewer, for it is related to the resources required by the whereabouts algorithms. We note that the JSON objects shown in Figure 6 are not final, they largely only support a single algorithm, "value" labeled "Cones" for "key"- Algorithm of JSON object "Query". Support for the "Bridgelets" algorithm can easily be adapted without needing to modify the priori by adding another nested JSON object "Bridgelets Information" within the "Query" object.

The "Cones" algorithm in its simplest form can be observed from the figures included in design document 1, but can be generalized to a series of positions in time. The output of the algorithm in this case will be chain of ellipses or “beads", also referred to as a “necklace”. The intersection of the area taken up by the necklace with the area of interest divided by the area of the necklace gives the probability of the object residing within the area of interest at a given time in between the first and last points of the necklace.

Given the premise of algorithm operation, the algorithm specific queries have JSON objects of their own; the “Query” object can be expanded on to support more query types and only two types are provided in the figure. The range query adaptation takes two series of time varied positions, finds their necklaces, and determines the range of possible distances separating the two objects during the specified time interval. This is why under the “Range Information” object you see two arrays of “Data Point”.

The contact query adaptation is nearly identical to the algorithm descriptor case. The “Contact Information” object translates contains a single array of “Data Point” to be used to create the necklace while the shape information, namely shape type, points, and radius for the circle case makeup the region of interest.

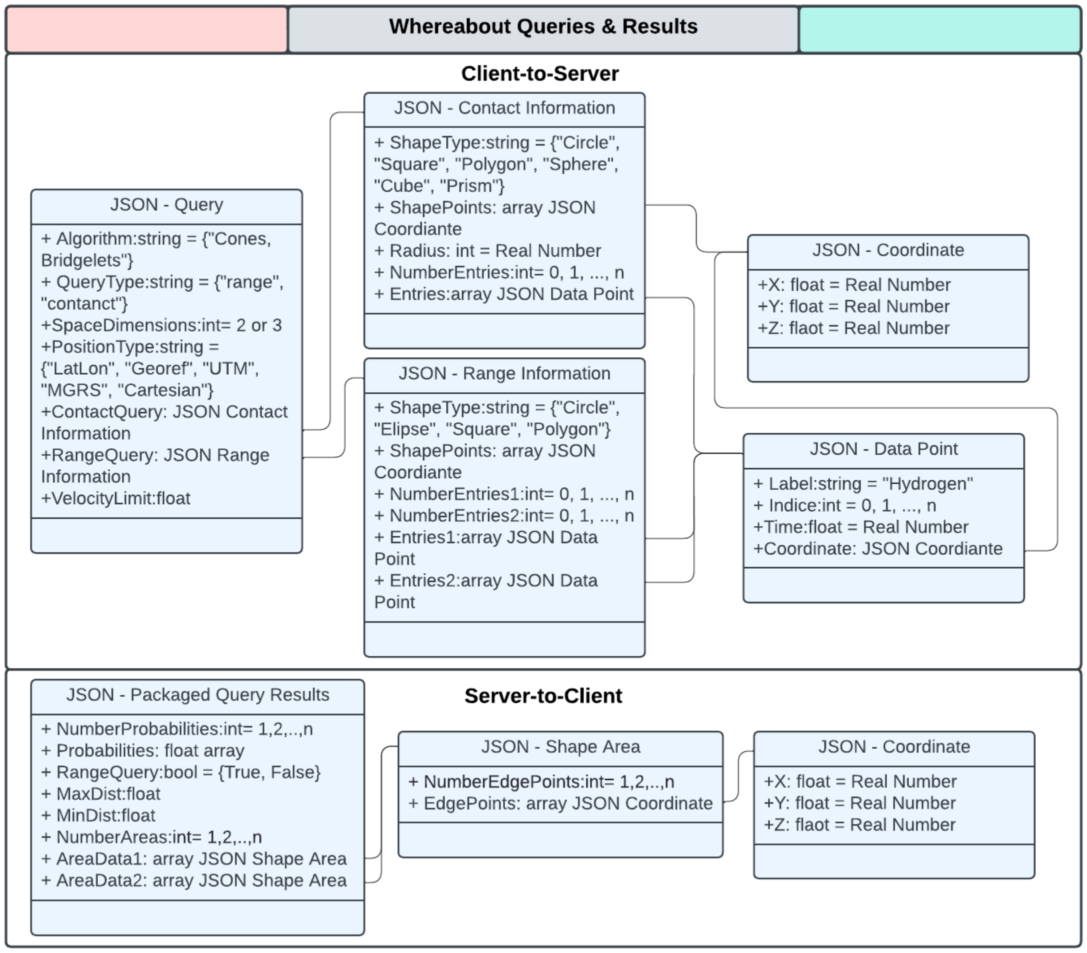

Whereabout Queries & Results
Client-to-Server
JSON - Query
+ Algorithm:string = {"Cones, Bridgelets"}
+ QueryType:string = {"range", "contanct"}
+SpaceDimensions:int= 2 or 3
+PositionType:string = {"LatLon", "Georef", "UTM", "MGRS", "Cartesian"}
+ContactQuery: JSON Contact Information
+RangeQuery: JSON Range Information
+VelocityLimit:float
JSON - Contact Information
+ ShapeType:string = {"Circle", "Square", "Polygon", "Sphere", "Cube", "Prism"}
+ ShapePoints: array JSON Coordiante
+ Radius: int = Real Number
+ NumberEntries:int= 0, 1, ..., n
+ Entries:array JSON Data Point
JSON - Range Information
+ ShapeType:string = {"Circle", "Elipse", "Square", "Polygon"}
+ ShapePoints: array JSON Coordiante
+ NumberEntries1:int= 0, 1, ..., n
+ NumberEntries2:int= 0, 1, ..., n
+ Entries1:array JSON Data Point
+ Entries2:array JSON Data Point
JSON - Coordinate
+X: float = Real Number
+Y: float = Real Number
+Z: flaot = Real Number
JSON - Data Point
+ Label:string = "Hydrogen"
+ Indice:int = 0, 1, ..., n
+Time:float = Real Number
+Coordinate: JSON Coordiante
Server-to-Client
JSON - Packaged Query Results
+ NumberProbabilities:int= 1,2,..,n
+ Probabilities: float array
+ RangeQuery:bool = {True, False}
+ MaxDist:float
+ MinDist:float
+ NumberAreas:int= 1,2,..,n
+ AreaData1: array JSON Shape Area
+ AreaData2: array JSON Shape Area
JSON - Shape Area
+ NumberEdgePoints:int= 1,2,..,n
+ EdgePoints: array JSON Coordinate
JSON - Coordinate
+X: float = Real Number
+Y: float = Real Number
+Z: flaot = Real Number

Figure 6 - *Whereabout Queries & Results JSON Objects*

The results of the array will be compiled into a JSON "Packed Query Results" object. This object contains entries containing resulting information from all algorithms and query types in the outermost portion of the structure. This is not extremely scalable and is subject to change to a format similar to the client-to-server query submission – where each query type has its own object. Regardless, the main contents include quantitative results such as probabilities and distances, as well as shape indices to be used by the frontend visualization engine ("Area Data" entries).

The visualization engine and window work together to view the data and algorithm results. The window is where visualizations take place while the engine is composed of logic driving the updating and management of the window. The window may contain a 2D map overlaid with data points and output algorithm necklaces or a 3D white space with equivalent volumes based on the dimensionality of the dataset provided by the user.

4.3.3 Functionality

The user would start by logging in or signing up and the front-end diagram for that process is shown in Figure 7, which goes over what the system would be doing when the user first starts the application. Figure 8 is the back-end side for the same part of the system. The back-end handles the checks and storing of user info as shown in the figure.

The next main functionality is dataset handling, and Figure 9, Figure 10, and Figure 11 all describe how datasets are handled in both the front-end and back-end. Figure 9 describes how the Front-handles new and existing datasets and how the user chooses what active dataset they want. Figure 10 shows how the back-end takes in and saves new datasets sent from the front end. Finally, Figure 11 shows how the Back-end sends saved datasets back to the Front-end for use.

The final large functionality is how the visualization works for the front-end and back-end. Figure 12 describes at a high level how the Front-end will work with visualization and what the user will go through when starting it. Figure 13 describes how the backend will handle visualization with the different choices for the user and that what will be sent back will be packaged then sent to the visualization tool and brought up with the users settings to be seen.

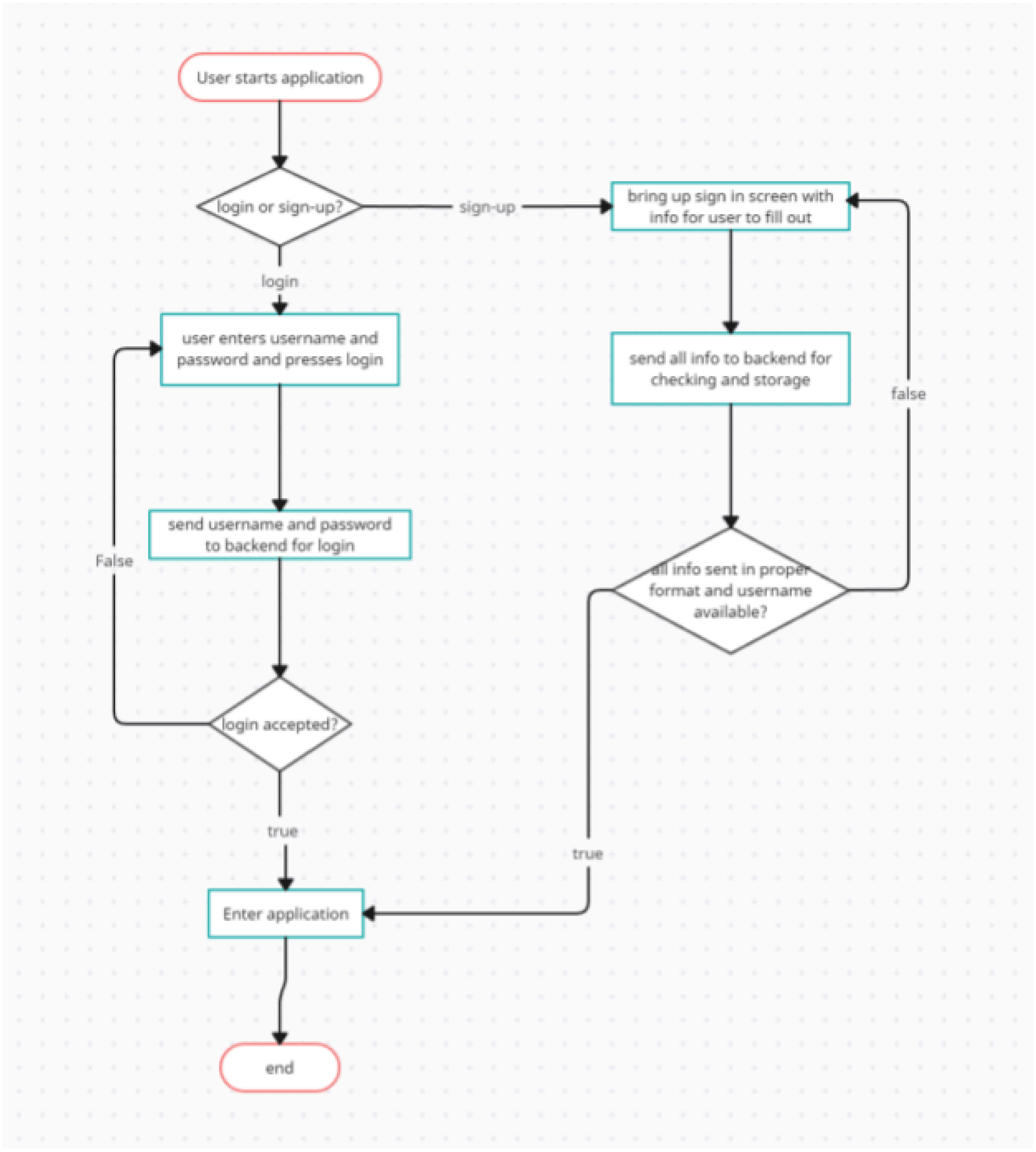

User starts application
login or sign-up?
sign-up
bring up sign in screen with info for user to fill out
login
user enters username and password and presses login
send all info to backend for checking and storage
false
send username and password to backend for login
False
all info sent in proper format and username available?
login accepted?
true
true
Enter application
end

Figure 7: Front-end login/sign-up

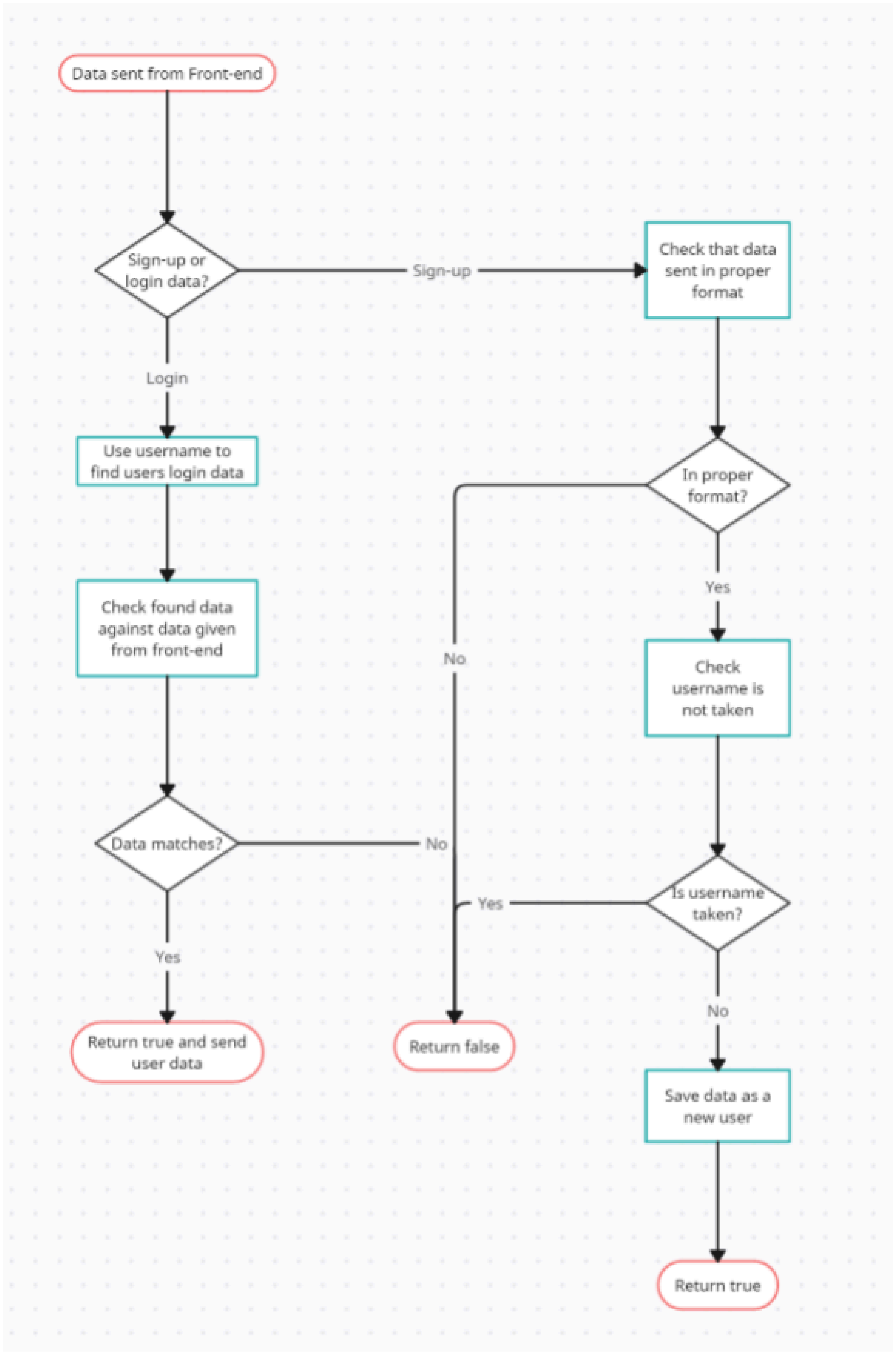
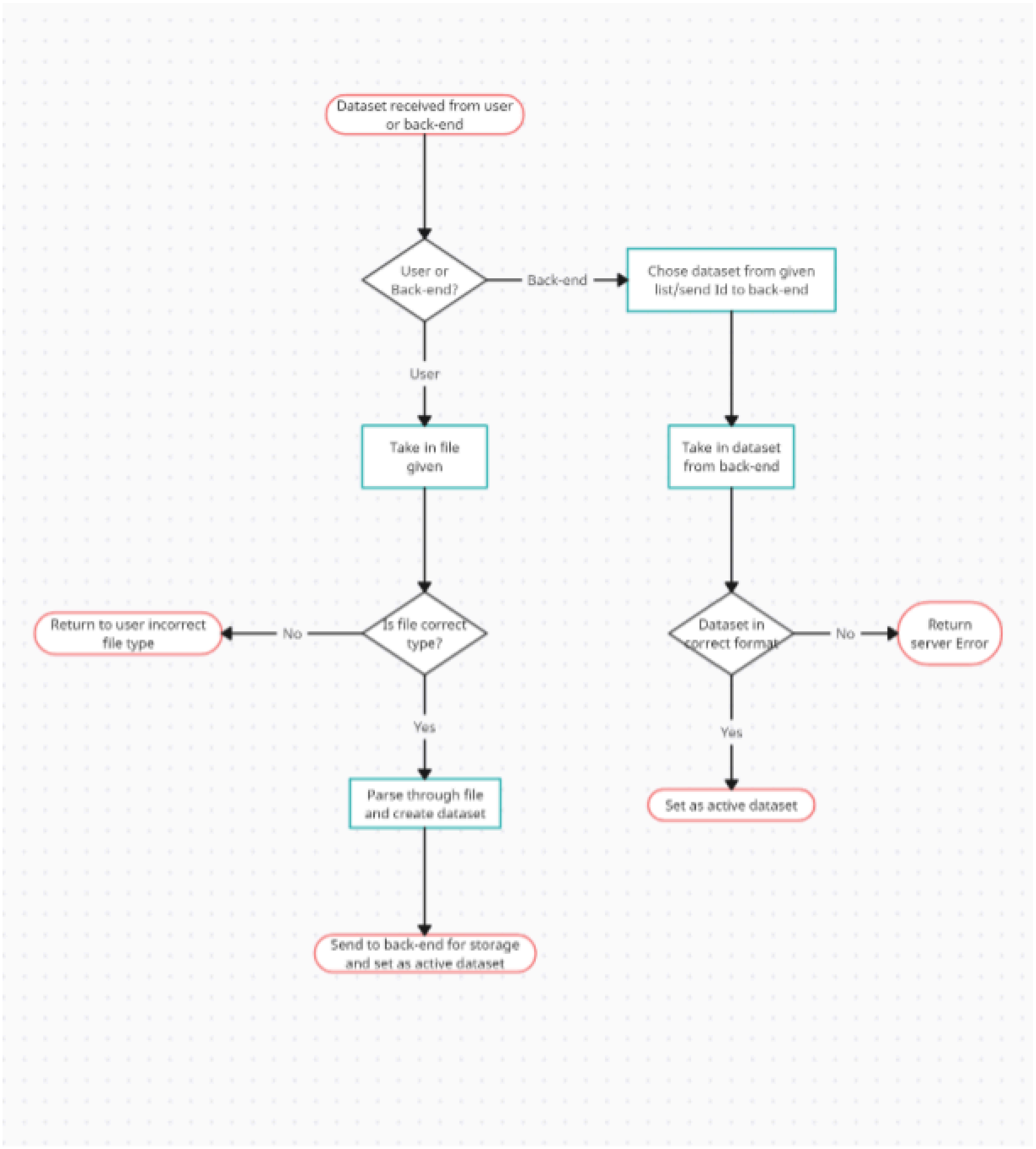

Data sent from Front-end
Sign-up or login data?
Sign-up
Check that data sent in proper format
Login
Use username to find users login data
In proper format?
Check found data against data given from front-end
Yes
No
Check username is not taken
Data matches?
No
Yes
Is username taken?
Yes
No
Return true and send user data
Return false
Save data as a new user
Return true

Figure 8: Back-end login/signup



Dataset received from user or back-end
User or Back-end?
Back-end
Chose dataset from given list/send Id to back-end
User
Take in file given
Take in dataset from back-end
Is file correct type?
No
Return to user incorrect file type
Yes
Parse through file and create dataset
Send to back-end for storage and set as active dataset
Dataset in correct format
No
Return server Error
Yes
Set as active dataset

Figure 9: Front-end file/dataset handling

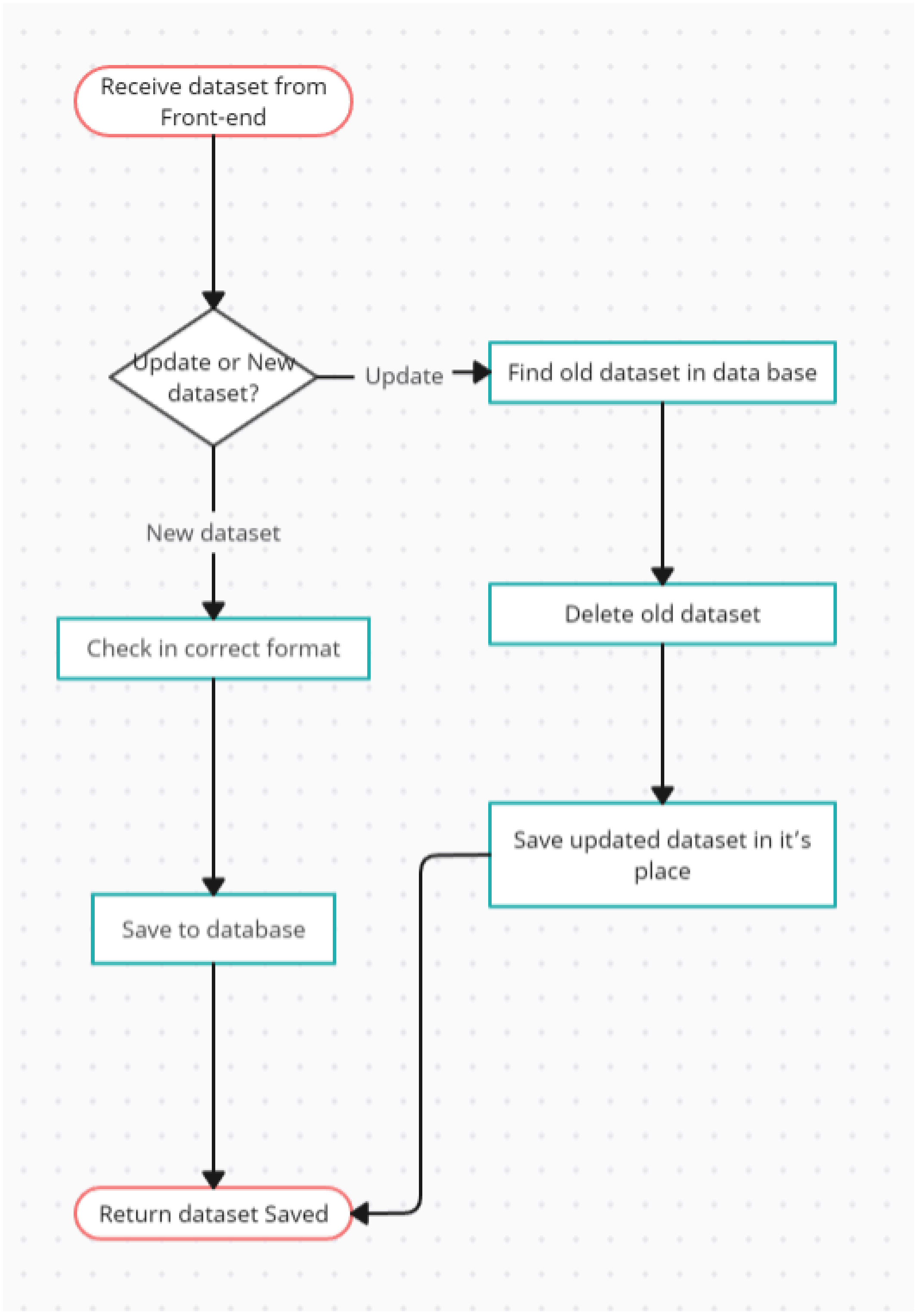


Figure 10: Saving/updating datasets

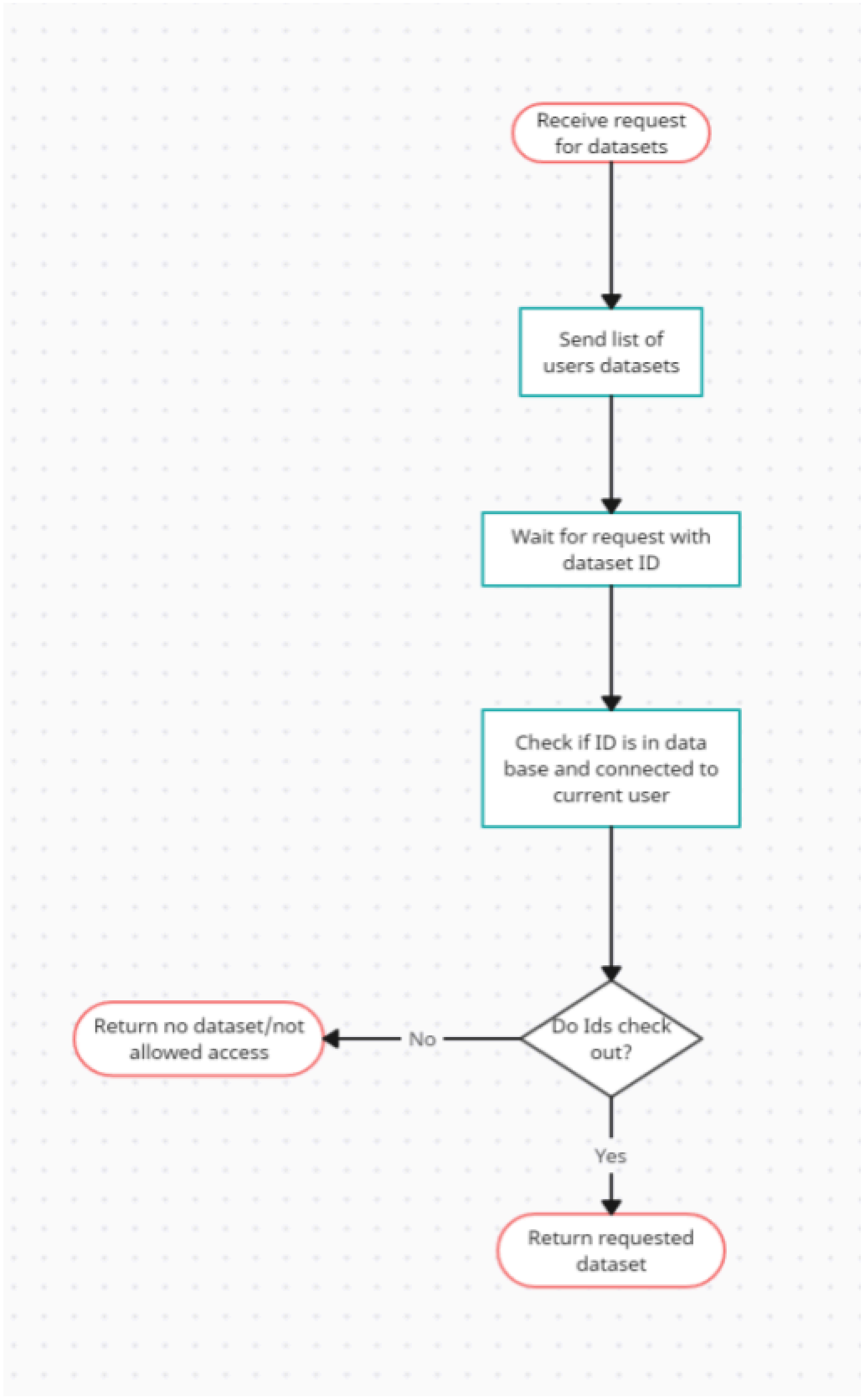

Receive request for datasets
Send list of users datasets
Wait for request with dataset ID
Check if ID is in data base and connected to current user
Do Ids check out?
No
Return no dataset/not allowed access
Yes
Return requested dataset

Figure 11: User requests dataset from database

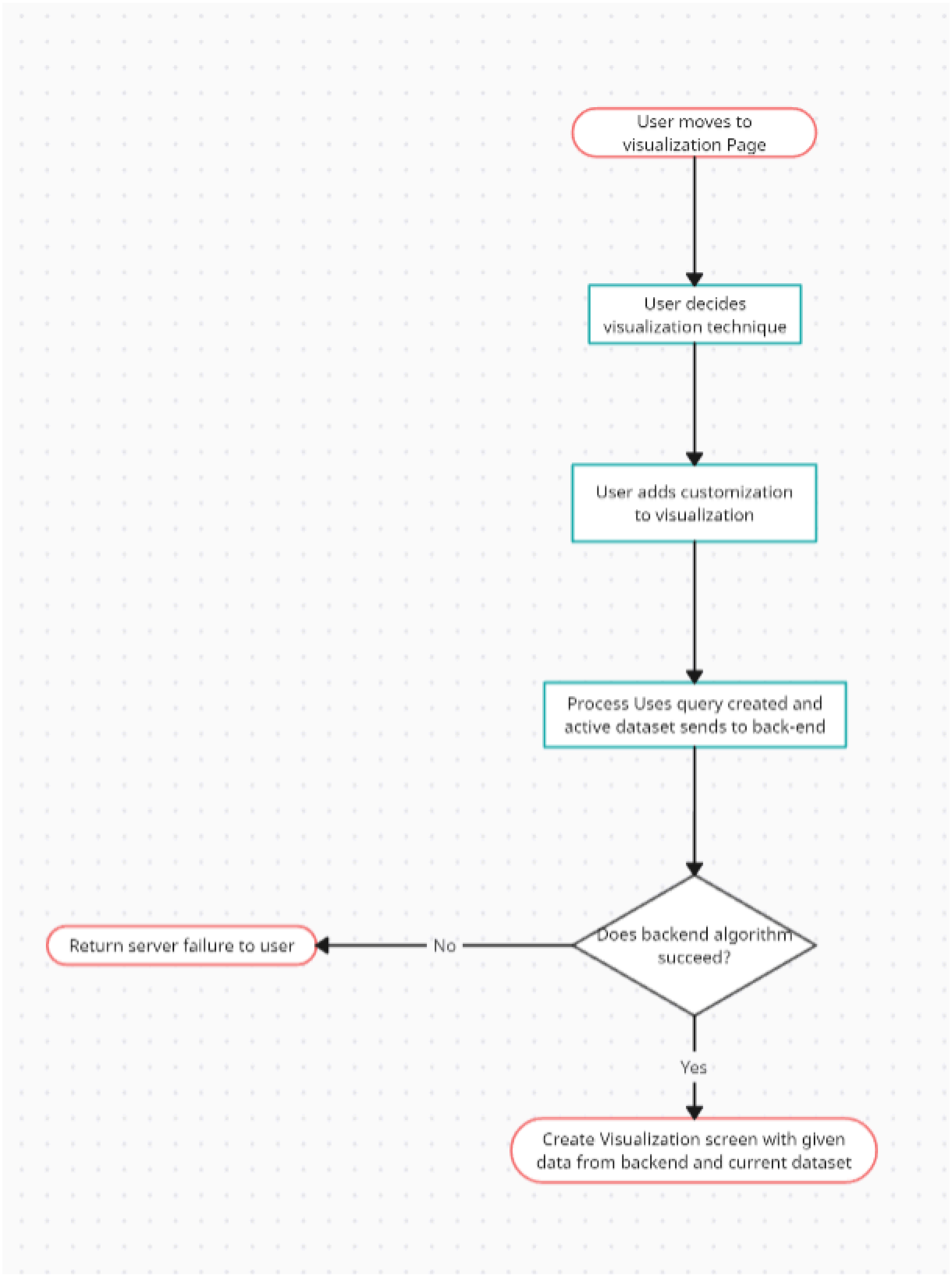

User moves to visualization Page
User decides visualization technique
User adds customization to visualization
Process Uses query created and active dataset sends to back-end
Does backend algorithm succeed?
No
Return server failure to user
Yes
Create Visualization screen with given data from backend and current dataset

Figure 12: Front-end Visualization

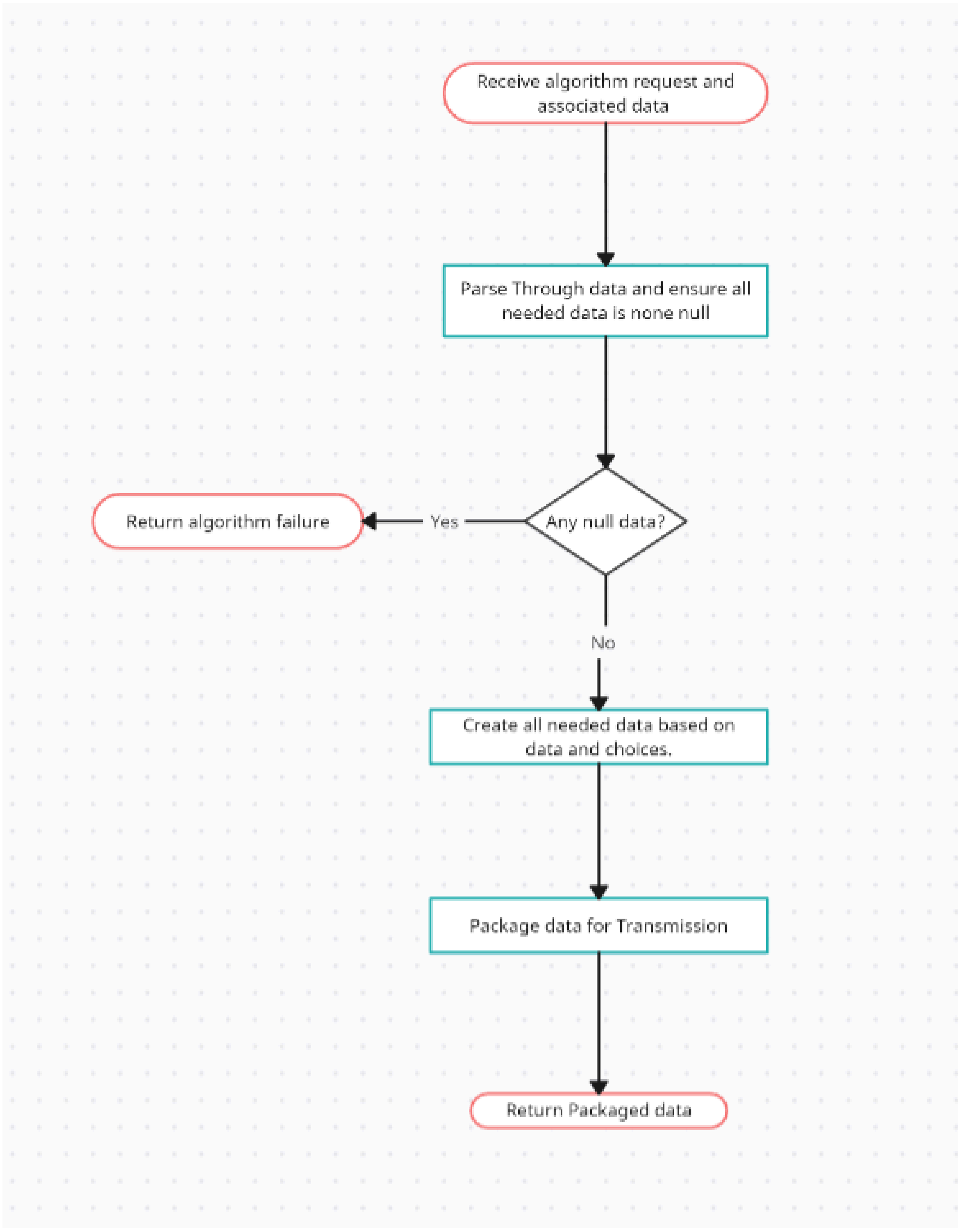

Receive algorithm request and associated data
Parse Through data and ensure all needed data is none null
Any null data?
Yes
Return algorithm failure
No
Create all needed data based on data and choices.
Package data for Transmission
Return Packaged data

Figure 13: Back-end Algoritm Request

4.3.4 Areas of Concern and Development

Our current website application design satisfies many of the requirements and user needs we have established. The application will have the ability to store datasets of a specified type, allow users to manage saved data sets, and display the requested visualizations using one of the two implemented whereabouts algorithms. These functionalities address most of the primary requirements and user needs that we have addressed in previous sections. In the future, our application could likely be expanded to implement additional algorithms and views for visualizations which could appeal to additional users.

As of now, it is unknown how long processing and saving data will take, which may be one area of concern relating to user expectations. Additionally, we have a relatively small team with limited experience in website design. Significant research and learning may be necessary in order for our team to implement the requested features of the application. Because of this, our team may not have enough time to implement all functionalities and algorithms requested by the client.

Our strategy to mitigate this concern will be to work incrementally and prioritize a single algorithm in order to make sure we have a complete and working application by the end of the semester. A question for our TAs and our client would be: how can we prioritize client requirements given that we have a limited amount of time for implementation?

4.4 Technology Considerations

The three main technologies that we will be using are Intellij, Vue.js and mapbox. We will also have a VM from ETG to run our back-end server. Some strengths for using Intellij is it has many built in Spring specific features to work with a Spring Boot server. Another advantage with Intellij is it has many debugging features to help fix errors that come with the creation of a server. Some disadvantages of intellij is it is not friendly for new users, due to the complexity of some of the features. The trade off of many built in features to the learning curve of Intellij is one that we think is worth it as it is one of the best softwares for the design. An alternative to intellij could be the Spring tool suite as it is a specific tool for creating spring boot servers.

A strength of Vue.js is the fact that it is component-based architecture as this makes it easier to create the front-end code as the components can be reused in different functions. Another strength is that it integrates easily with mapbox making what we need to create our visualization tool. A weakness of it is the performance as large visualizations could cause issues and with that our plan is if performance suffers to split the visualization up into multiple segments to be shown as to not slow the performance of the tool. A trade off that we will see is the component-based architecture with performance as the architecture will speed up the development process but with the performance issues finding a balance between the quickest development and the correct optimization will be key. An alternative to Vue.js is React, another javascript based system that can easily integrate with mapbox.

The last main technology used in the design is mapbox. It will have most if not all of the features that will be needed, and it will be tested to ensure all functionality that it will need is part of the software. It also is known for having good performance with large datasets helping to negate some of the performance issues that could happen with Vue. An issue that could arise is that Mapbox does not have all the functionality needed for our design so a backup that can be used possibly with Mapbox if it does not have the functions needed is plotly.js which is also easily implemented into Vue and is our planned backup software if the need arises.

4.5 Design Analysis

So far, we have focused on researching the languages and libraries available for geographic, 3D, and 2D graphing. As we have discovered new tools or information about our chosen libraries, some of our choices have changed subtly, although our overall design plan has remained the same. Currently, we are focused on finding tools that save us additional time during implementation, and we believe that the tools we have chosen will allow us to complete the whereabouts application successfully. We hope to be able to build a more complete prototype that can handle simple data inputs once we have received a VM and website. Once we have set up our server and have a VM running, we will be able to start attempting to connect frontend and backend in a simple prototype website application to prove proof of concept.