Team Ceres

Technological Feasibility Report

October 6, 2020

Sponsored By: David Trilling, Michael Gowanlock

Team Mentor: Fabio Santos

Team Members: Javier Quintana, Joseph Sirna, Miles Barrios, Zach Messenger

1.Introduction

In this document we will be introducing our project on a more technical basis, and then analyzing the feasibility of the technological challenges that we have to overcome. This will be broken down into multiple subsections for the various components of the project in order to address each challenge individually; additionally, we will conclude the technical analysis with an overview of how each of the solutions are able to come together, but first, let's get to the project overview.

1.1 The Problem

Astronomers all around the world participate in all-sky surveys every night in hopes to gain knowledge of both small and large bodies in space. These surveys produce very large quantities of data that researchers are then able to filter and sort for their own analysis. The information they gain can be applied as follows:

- Drawing conclusions about the formation of the galaxy and our solar system
- Enlightening us to the nature of our universe
- Aid us in preparing for the eventuality of another catastrophic meteor impacting the planet
    - This event being the most motivating for this field of study

Scientists have found evidence of large asteroid impacts over the course of Earth's existence, and eventually another one will impact with our planet. This event could be catastrophic for human life and at the very least cost billions of dollars in damage to the infrastructure humans have built here. That is why it is important to be able to study these bodies in space with ease, but with the amount of data being collected every night and the lack of a good medium for this data analysis, it is almost impossible for astronomers and researchers alike to study these all-sky surveys.

1.2 Current Solution

Currently, Zwicky Transient Facility (ZTF), located in SanDiego County, California, is operating and producing nearly 2 terabytes of data per night! Once the construction of the Rubin Observatory in Chili is complete, this number is expected to go up to 20 terabytes of data per night, and there are almost no accessible interfaces for accessing and analyzing this. Our clients here at NAU are ingesting the portion of the data from ZTF that is broadcasting information related to asteroids; however, they do not have an existing interface for analyzing the data.

Currently, our clients work with another colleague who runs programs on the raw data in order to provide them with the desired smaller subsets and diagrams they need to analyze this data. This is not an ideal solution because a number of things inhibit their research. Any analysis of data that they want to perform must first go through their colleague in order to extract the necessary data, then images have to be created from the resulting data set. This is already a slow process, but their entire method of analysis also relies on another worker. If their colleague is ever sick or unable to help with the data extraction and visualization, then they are no longer able to analyze the data. This is the problem that Team Ceres aims to solve.

1.3 Team Ceres' Solution

Our goal is to create an easily accessible web interface that allows users to view and process large amounts of data both visually and in tables with ease. The database of raw astronomical data will be linked to the web application, allowing researchers to be able to view data and draw conclusions almost seamlessly. We want to eliminate the need for tedious manual labor or constant back and forth discussions in order to obtain the correct data.

We plan to work with our clients over the next year to refine the requirements and develop this application to further enhance their research. The key components of our web application will be:

- A responsive front end user interface with tools for applying data analysis
- A functional back-end api to serve search requests on the database
- Handling large size data sets efficiently and effectively without loss of responsiveness in the webpage

This is a very general description of the main components of our project, but throughout this document we will provide more details on the feasibility of the various portions. Although we are confident that we can develop the previously described technology, there still exist some technological challenges that we need to overcome along the way to our goal.

2.Technological Challenges

This section is going to discuss the main challenges that have been discovered for this project. The challenge descriptions will provide a brief overview of what each challenge is and how it pertains to the overall problem we are solving for the database holding the ZTF asteroid information. These challenges will be analyzed in the following section to decide what the best possible solution is for each problem.

Our project currently has the following challenges:

- Importing data into a web-accessible database
- Choosing a framework to develop a user-friendly web interface
- Data visualization
- Selecting a service to host the web application
- Authentication for role-based permissions.

2.1 Importing data into a web-accessible database

Currently, all raw observation data is stored in a single sqlite 3 database file. This approach does not scale well for web applications, especially those where several concurrent queries may be run depending on website traffic. Because of the amount of data currently contained within these database files, it is impractical to expect anything other than an industry standard database engine to manage this data efficiently.

2.2 Choosing a framework to develop a user-friendly web interface

This project is built around the idea of building a user-friendly interface for the SNAPS database so that users are able to retrieve information regarding asteroids quickly and easily. In order to build a responsive web application, a framework would provide an efficient and effective solution to developing the application. When choosing a framework to develop a web interface, the developer must consider a few characteristics of the frameworks including: performance, scalability, maintainability, and learning curve.

2.3 Data Visualization

As it currently stands, the SNAPS database has no interface to export data graphically and all visuals have to be generated by the person who manages the database by running queries on specified data and then having them run through a program they made in Python. This is problematic for a number of reasons but the main ones being that without a person who manages the database (which is currently only one person at

this time), no data can be visualized or exported into graphs without the script to run it and without the person running the database no data can be exported at all.

The obstacle that makes data visualization a challenge for this project is finding a suitable library in JavaScript that will allow us to create scalable, easy to read graphs, that will best summarize the data being exported. The reason this is an obstacle is because while the SNAPS database is a smaller scale version of the Zwicky Transient Facility (a data stream that participates in collecting all-sky survey data), it still collects thousands of data points that users may want graphed or summarized in a single table or graph. Because of this, Team Ceres needs to ensure that the graphic library that we choose is capable of handling large amounts of data without suffering on performance or quality.

2.4 Selecting a service to host the web application

Due to the requirement of needing a user-friendly web application, we need to have a place to host that web application. This web application will need a web hosting service that is reliable, secure, efficient, and scalable. The web hosting service should be one that does not require a large amount of cost but still provides the services needed to run the web application.

2.5 Authentication for role-based permissions

One of the requirements for our web application is that we have a system for authentication of users, and delegating role based permissions. This system will serve the purpose for some users to have the ability to perform critical alterations/modifications on the database, while other users are only able to access, view, and query the database normally. Additionally, we want users to be able to have some of their data stored that would be useful for them, such as specific data sets, visualizations, or query logs.

3.Technological Analysis

An important step in determining which solutions to use for smaller problems is analyzing them critically against alternatives. In this section, different approaches for various solutions we have identified as usable will be analyzed and compared. Below, we address problems including database implementation details, choosing a framework for the frontend of the application, determining an efficient way to visualize the data, and authenticating users while avoiding security vulnerabilities.

3.1 Importing data into a web-accessible database

3.1.1 Current Database Implementation

The existing infrastructure with regards to observational data is a variety of sqlite 3 .db files, each containing one table. One .db file serves as an intermediary step between raw observational data from ZTF and calculated information about them. These calculated metrics are manually generated using a python script, and output to a similar table schema, in a separate database file. Regardless of the specific database driver used (sqlite 3 in this case), singular database files stored on disk are generally regarded as insufficient to service a web application because of the inevitability of concurrent read/write operations, limited disk throughput, and lack of sufficient protections against attacks on the database.

Since a non-hosted database implementation was concluded to be insufficient for a public-facing web application, the next reasonable conclusion is that this web application must be serviced using a hosted database. A hosted database will manage concurrent read/write operations and enforce basic authentication, to ensure that different database roles are restricted only to the scope they absolutely require (e.g. a read-only query will not have permission to write to the database, offering some protection against SQL injection).

3.1.2 Choosing a Database Engine

Whichever database engine is decided upon, it must satisfy the following requirements:

1. ***Compatibility with hosting environments.*** Since this will be a public-facing web application, a managed hosting environment is inevitable. The database engine should have a good track record of compatibility with this environment.

2. ***Support for large data quantities.*** The potential end-of-life database size for this project could approach 1TB, according to the person responsible for maintaining the current database implementation.

3. ***Development Experience.*** This is the backbone of the project. Learning a DBMS entirely foreign to everyone on the development team uses precious development time that could be spent optimizing for responsiveness.

4. ***Licensing Costs.*** Because of the limited resources of the team, the price of a database management system license is an extremely important factor. For example, Microsoft SQL Server might seem like a good choice on paper, but the prohibitive licensing costs make it infeasible for this project[12].

3.1.3 Alternatives

1. **Microsoft SQL Server**

   If there's one database engine that screams "Industry Standard," it's Microsoft SQL Server. First introduced in 1989 and used in 98 of the top 100 Fortune companies [2], SQL Server has proven to be able to handle large amounts of data with relative ease.

2. **MySQL**

   Developed by Oracle since 2010, MySQL is the most popular open-source database engine. Including all standard SQL features, MySQL is a good solution for most database applications.

3. **PostgreSQL**

   PostgreSQL was first prototyped in 1988, and has since grown to be another popular open-source database engine.

3.1.4 Analysis

1. **Microsoft SQL Server**

   Though reliable and efficient, MS SQL Server is a commercial application, with commercial-sized license costs.

   Pros:

   - High compatibility with most popular managed hosting environments (AWS, Azure)
   - Scalability
   - Performance monitoring tools[13]
   - Team development experience with T-SQL

   Cons:

   - High licensensing costs (MS SQL Server 2016 Standard running on Windows Server 2016 costs potentially $0.864/hr on AWS[8])
   - A single-core MS SQL 2019 Standard license is $899[12].

2. **MySQL**

   MySQL has no licensing costs, and is a popular alternative to MS SQL Server.

   Pros:
   - Compatibility with hosting environments[15]
   - Performance
   - No licensing costs

   Cons:
   - Syntax slightly different from T-SQL
   - Database-blocking backup operations[16]

3. **PostgreSQL**

   PostgreSQL is a proven DBMS, but nobody on the development team has experience using it.

   Pros:
   - Performance
   - No licensing costs
   - More data management and abstraction tools than MySQL[14]

   Cons:
   - SQL syntax slightly different from T-SQL
   - Contains useful features, but unnecessary for this project

3.1.5 Chosen Approach

Each of the alternatives listed above are given a (semi-)arbitrary score, 1-5, based on the analysis. The cost of licensure will also be rated on a 1-5, with a 5 being free and a 1 being extremely expensive (relative to the other alternatives).

	Compatibility	Performance	Development Experience	Cost of Licensure	Total
MS SQL Server	5	5	5	1	16
MySQL	5	4	3	5	17
PostgreSQL	5	3	0	5	13

Microsoft SQL Server is the preferred DBMS for this project, but prohibitive license costs make it infeasible. MySQL and Postgres have no licensing costs, and so do not have this problem. MySQL edges out over Postgres mostly because of the past development experience of the team with MySQL. Postgres has more data management features, but they are unnecessary on this project.

3.1.6 Proving Feasibility

To prove the project-specific feasibility of MySQL, it must be first tested to ensure that the proposed database import job (see: technology integration section) can easily insert rows from a sqlite3 .db file into a MySQL server without a significant performance hit. In the event of a performance hit, the job can be scheduled to run at odd hours of the night, during historically low-traffic times.

Integration with the REST API is also essential. .NET Core provides a multitude of database connection drivers, and MySQL is supported among them. A benchmark-driven performance test would be a useful tool to pinpoint bottlenecks in this implementation. Here, MySQL can also be tested for performance with a large amount of data.

3.2 Choosing a framework to develop a user-friendly web interface

3.2.1 Choosing a framework

The information currently being brought in from the Zwicky Transient Facility is currently stored on a database and simply queried when that data is needed. However, this current route is inefficient and requires a simpler way to allow any scientist to access this database and search for the information they need without going through a middleman. Providing a simple, responsive user interface on top of the database will decrease the time it takes to get asteroid data and will allow for a better user experience.

3.2.2 Desired Characteristics

Choosing a framework must take multiple characteristics into consideration. These characteristics can influence the performance, reliability, and responsiveness of an application. These characteristics include:

- ***Performance.*** This web application deals with the visualization of big data relating to asteroids. Due to the large amount of data, it is key that the web application is able to perform at high speeds, even for massive amounts of data.

- ***Scalability.*** Due to the potential of massive growth in the amount of data stored in the database, the web application needs to be able to handle even more amounts of data/visualizations in order to assist researchers.

- ***Maintainability.*** This project is assumed to grow in size and thus it is mostly safe to assume that this web application will need maintenance in the future in order to keep it functioning and responsive. The framework needs to have a high level of maintainability to plan for the future.

- ***Learning Curve.*** As this web application is to be finished by May of 2021, it is important to choose a framework that has a moderate learning curve. This is to reduce the need to spend a large amount of our time understanding how to use the framework rather than producing code.

3.2.3 Alternatives

1. React

An open source Javascript library that was originally started back in 2013. This library was created by Facebook and has quickly become the leader in front-end libraries/frameworks. This library focuses on building single-page applications[21]

2. Angular

An open source framework designed to simplify the development of single-page applications. Angular was developed by Google back in 2016 and has become one of the main frameworks used for web applications. Angular uses typescript, a language built around Javascript that adds types to increase the development experience[22]

3. Vue.js

An open source Javascript framework that was launched in 2013. Vue.js was developed by Evan You after working for Google using Angular. Vue.js excels at developing highly adaptable user interfaces and single-page applications[23]

3.2.4 Analysis

Below is an explanation of the pros and cons regarding the frameworks for front-end development introduced about:

1. React

- ***Pros***
  - Easy to learn due to design[18]
  - Uses a Virtual DOM (Document Object Model) which allows high performance speeds[17][18]
  - Has large support for server-side rendering which makes it great for applications that contain large amounts of content[17]
  - Implements Functional Programming practices creating easy to test, reusable code[18]
- ***Cons***
  - Moved away from traditional class-based components which creates a barrier for programmers used to Object Oriented Programming[17]
  - Due to the loose structure of React, it leaves developers having to make decisions regarding development[17]

2. Angular

- ***Pros***
  - Compiles HTML and TypeScript into JavaScript during development. This means all the code is compiled in the backend prior to loading the web application[19]
  - Provides good structure and architecture that allows for easy scaling[17]
  - Highly detailed documentation that allows for a developer to answer all of their own questions[17]
- ***Cons***
  - Has more structures including : injectables, components, ppes, modules, etc.). This increase in the number of structures can cause developers to have to spend more time learning the framework[17]

3. Vue.js

- ***Pros***
    - Detailed documentation that allows for a fast learning curve for developers[17][20]
    - Vue.js is not limited to only single-page applications. Vue.js allows for the development of more difficult web interfaces[17]
    - Due to the reusable templates, Vue.js can be scaled quickly[17][20]

- ***Cons***
    - Due to its smaller size in the JS framework market, there is less knowledge available to developers to aid in software development[17]
    - Vue.js can struggle when being implemented into larger projects and has had limited testing when attempting to do so[17][20]

3.2.5 Comparison and Chosen Approach

Below is a chart detailing the analysis based on the desired characteristics mentioned above, these criteria will be ranked on a scale from low - high:

Framework	**Performance**	**Scalability**	**Maintainability**	**Ease of use**
React	high	high	high	medium
Angular	medium	high	high	low
Vue.js	high	low	medium	high

As seen in the chart above, the options we provided are all quite competitive in what they have to offer. The chosen approach for our team is React, this is because React's ratings outperformed Angular in performance while having a higher ease of use than Angular. This will allow the team to learn React quickly and implement it in an efficient and effective way.

3.2.6 Feasibility

Testing the feasibility of React will be quite different from testing the feasibility of other challenges. The testing will be the thoughts and opinions after doing the following tests:

1. Reading through documentation to ensure that it is understandable and does not require a large amount of prior knowledge on single-page applications. This in turn will prove that React has a medium learning curve which does not prove to be an issue.

2. Building out a small, barebones web application to showcase the power of reusable components which in turn shows off scalability. After building out the barebones web application, this provides us with a small peak at performance by showing off the responsiveness of the web components.

Along with testing the framework on its own, the team will need to ensure that the framework communicates well with our backend language as these will make up a large amount of the web application. Proving feasibility will rely heavily on the thoughts of the developers as the front-end framework needs to communicate well with other technologies being used.

3.3 Data visualization

3.3.1 Overview of Data Visualization

In order to make the data from the SNAPS database easier to examine, there will be components of the application that make it possible to view and export selected data into graphs and tables. By adding this feature to the interface, it will make the application more useful for users who are interested in spotting trends in data and will save users time in creating graphs that they would have otherwise made by manually entering data points into another program. Because the bulk of the data visualization is being done on the web interface.

3.3.2 Desired Characteristics

The ideal data visualization component has the following desired characteristics that will impact the decision of the graphic library we choose to use:

- ***Scalable Data Representation.*** It is anticipated that data sets of various sizes will be evaluated using the data visualization tool, so it would prove useful to show graphs that fit the data better and do not skew the appearance visually. In order to combat this problem, the data visualisation tool will create any graphs it makes based on the amount of values being graphed and a five-number summary of the data being evaluated. The reason the graphs will be created this way is because the five-number summary accounts for the minimum and maximum values of the data thus making it easier to determine the scale of the graph axes and how much they should increment.

- ***Data Summarization.*** For large sets of data it would be highly beneficial to have a summary generated of outstanding data points and outliers. The reason this would be useful to have is because if a graph is generated with hundreds or thousands of data points it may be hard to pinpoint any specific data that stands out from the rest. Ideally, either a five-number or seven-number summary would be generated from the data to help comprehend the spread of the values in a plot.

- ***Ability to Export Graphs in Various File Formats.*** When it comes to exporting data, Team Ceres wants to ensure that the library we select is capable (or compatible with other software) of formatting the graphs into file formats that will benefit users for research, documentation, or presentation. Because of this, it is important that the visualized data can be exported into the following formats: PDF, PNG, and CSV. This will allow any specific data that was requested to be reusable for our users and flexible to show off.

3.3.3 Alternatives

1. **Google Charts**

   Google's open source web API that was originally released in 2007. This library was created to allow users to embed graphs and charts into their web pages while maintaining a low learning curve. Google Charts latest release was in July of 2020 [5].

2. **Dygraphs**

   Dygraphs is an open source JavaScript library that focuses on graphing and charting data in an interactive way. It has focused on handling huge data sets, highly customizable graphs, and scalable interactive data [6]. It was released in 2013 and had its most recent update in December of 2017.

3. **Chartist.js**

   Chartist is an open source JavaScript library that focuses on plotting data in SVG format and has built in animations to make graphs more visually appealing. It is developed by a community of developers via GitHub and seems to have been released in 2014. The latest version of Chartist was released in 2019 [7].

3.3.4 Analysis

Below is a breakdown of the libraries that the team is looking into along with the *pros* and *cons* of each library.

1. Google Charts.

- ***Pros:***
  - Cross browser compatibility (even on mobile devices)
  - Automatic data refreshing/real time uploads on graphs [5]
  - Very large selection of charts to select from
  - Ability to hover over graphs to view point values
  - Support exporting graphs into PNG and CSV format [5]
- ***Cons:***
  - Does not mention ability to zoom or pan over graphs after they have been created
  - No mention of large data set capability

2. Dygraphs.

- ***Pros:***
    - Made to hand huge data sets (claims millions of data points without getting bogged down) [6]
    - Interactive graphs with zoom, pan, and mouseover functionality
    - Compatible with all browsers (including mobile devices) [6]
    - Scalable graphs and range selector to view specific ranges of data

- ***Cons:***
    - When zooming in on a graph it seems that you have to reload the graph to get the full picture again
    - No explicit function seems to outline data summarization
    - Does not seem to have built in functionality for exporting graphs and tables

3. Chartist.js.

- ***Pros:***
    - Premade templates that can be used for data visualization dashboard
    - Large variety of graph types that can be made
    - Graphs can be animated [7]
- ***Cons:***
    - Graphs do not seem to be able to be zoomed in, panned around, or moused over
    - Unable to export to any file format with graphs that are created
    - No mentioned of large data capabilities

3.3.5 Comparison and Chosen Approach

Graphing Library	**Scalability**	**Data Summarization**	**Data Exporting**
Google Charts	Not Mentioned	Possible	PNG and CSV
Dygraphs	Possible	Not Mentioned	Not Mentioned
Chartist.js	Not mentioned	Not Mentioned	Not Mentioned

3.3.6 Feasibility

After researching the libraries mentioned above it is very apparent that the Chartist.js library brings very little to the table that will be needed for this project and for this reason it will not be considered for our data visualization component.

While Google Charts has the most benefits marked in the table, Dygraphs is still a contender worth looking into. Both libraries will have to be tested in order to be certain of which is best but the reason Dygraphs is still worth looking into is because its specific mention of large data capabilities and its interactive graph features that make outlining data possible. Google charts however has the most uses but it lacks interactive capabilities and fails to mention whether or not large quantities of data will function well.

3.4 Selecting a service to host the web application

3.5.1 Overview of web hosting services

Choosing a web hosting service is important when developing a web application because it provides the actual web application to live. This web host needs to be able to provide a reliable service that is accessible to the users and ideally would provide some extra resources to the web application itself. Choosing a secure, easy to integrate web host will allow for our application to live safely online, available for use to many.

3.5.2 Desired Characteristics

Choosing a web hosting service requires for a few different characteristics to be examined prior to making a decision. The following are the criteria that will be evaluated for each:

- ***Cost.*** Cost plays a big role in deciding on a web hosting service because reducing the cost of hosting the website would allow for money to potentially go towards other features and/or services that may be used in the future.
- ***Availability.*** In order to ensure that our application is always usable for the researchers and other users, the web hosting service should rate highly when looking at the how available it keeps our web application.
- ***Space.*** Since this web application deals with a large amount of data, it is important to ensure that the web hosting service can handle a decent amount of data being transferred.
- ***Reliability.*** Due to the location of the various researchers and scientists that might be using the web application, our team needs to ensure that the web hosting service we choose has a high uptime (low number of outages) for the servers.

3.5.3 Alternatives

1. Amazon Web Services (AWS)

AWS is a cloud based service that has quickly risen to popularity. AWS allows for a user to stand up various types of applications on servers that are managed by Amazon. AWS was originally launched in 2006. When it comes to costs, AWS only makes you pay for what you use.

2. Microsoft Azure

Created by Microsoft, Azure is a cloud computing platform that allows users to host web applications, REST APIs, and mobile backends. Azure, similar to AWS, makes you only pay for what you use. Azure was released back in 2010.

3. Monsoon (NAU)

Monsoon is a high performance computing cluster that is used by the NAU research community. Each user is provided with a large amount of storage which would be very useful for our web application. This computing cluster was released to researchers in 2014.

3.5.4 Analysis

1. Amazon Web Services (AWS)

- ***Pros***
  - Diverse set of tools[24]
  - Low Cost (pay for what you use)[24]
  - Reliable security[24]
  - High availability[24]
  - Detailed documentation[24]
- ***Cons***
  - Amazon's EC2 service has limits[24]

2. Microsoft Azure

- ***Pros***
  - High Availability[25]
  - Cost Effective (pay for what you use)[25]
  - Reliable Security[25]
  - Highly Scalable[25]
- ***Cons***
  - Requires extensive knowledge on usage[25]
  - Requires management[25]

3. Monsoon (NAU)

- ***Pros***
  - Owned by NAU, used for research[11]
  - Lots of storage provided for researchers/projects[11]
  - Minimal costs[11]
- ***Cons***
  - Time to set up hosting environment[11]
  - Physical hardware must be managed[11]

3.5.5 Comparison and Chosen Approach

The criteria mentioned above will be used to choose where the web application will be hosted. These will be rated on a scale from 1 - 3:

Web Services	**Cost**	**Availability**	**Space**	**Reliability**	**Total**
AWS	2	3	3	3	11
Azure	2	3	3	2	10
Monsoon	3	1	3	2	9

The three alternatives given above scored within a few points when it came to judging the options based on cost, availability, space, and reliability. Based on the results above (total), it seems that AWS is the chosen web hosting provider for this web application.

3.5.1 Feasibility

In order to test the feasibility of AWS, the team should attempt to host a demo web application on AWS to ensure that it is quick, reliable, and meets our needs. This will ensure that the developers understand how to use and implement AWS into our project. Feasibility can also be tested by roughly calculating the cost to run our web application on AWS so that we can decide if the costs are worth it.

Our chosen approach above provides a great solution to our challenge of finding a web hosting provider that fits the potential size of the web application. However, due to client restrictions at this time, the web application will be hosted on a local machine in order to allow for the advancement of development. Should the web application grow in size and usage, AWS would be a great addition to this project.

3.5 Authentication for Role-Based Permissions

3.5.1 Overview of Authentication

As mentioned above, we want user authentication to be a component of our application.There are numerous reasons for us to do this, but overall we feel it is useful to have user accounts and ability to save data within the website for further use. Users would be able to log into an account within the website, and based on this have data available that they had previously saved or used. Additionally, users would have different role levels for access into the website and the database. We think this would really improve the ease of access for the functionality of the web application.

3.5.2 Desired Characteristics

Overall, authentication should be fairly simple to integrate into our application, but there are still a few key characteristics that we would like to see within our authentication service:

- ***Secure authentication.*** This is pretty straightforward, but we want to make sure that our application and login via authentication is secure for our users. Their account is for their own personal convenience, so if they have any sensitive data within it, we want to make sure that it is protected through our authentication method.

- ***Easy to set up/maintain.*** We want our application to be easy to maintain throughout the future as well, so that this product is useful for a long time. For this reason, we want our authentication service to be easy to set up as well as to maintain. This would allow future programmers on this project to be able to update the application in the future, and maintain all of its functionalities.

3.5.3 Alternatives

1. **Firebase Authentication**

   An authentication platform developed by Google for creating safe mobile and web applications. Features include backend services, easy to use software development kits, and ready-made UI libraries to authenticate users to an application.[3]

2. **Passport.js**

   An authentication middleware for Node.js. This authentication method is very flexible and modular in its implementation. Features include persistent sessions, dynamic scope and permissions, as well as easy to handle success/failure results.[4]

3. **Self Implementation**

   This method of implementing authentication would be the most complicated, but would be able almost guaranteed to satisfy our requirements for authentication on our web application. This would be a very time consuming and tedious process to code on our own.

3.5.4 Analysis

1. Firebase Authentication

- ***Pros***
  - Data is secure [3]
  - Easy to set up, use and maintain
  - Backed by Google [3]
- ***Cons***
  - Some reported user errors in initial setup of authentication, but many seemed able to be resolved.

2. Passport.js

- ***Pros***
  - Data is secure
  - Has built in functionality to aid in setup [4]
- ***Cons***
  - Also some reported user problems in with authentication failures

3. Self Implementation

- ***Pros***
  - Could be built to be secure
- ***Cons***
  - There would be a lot of preparation to get this up and running
  - Would only be as maintainable as we are able to make it

3.5.5 Comparison and Chosen Approach

Method	Security	Ease of Use/ Maintainability
Firebase Authentication	x	x
Passport.js	x	x
Self Implementation	x	

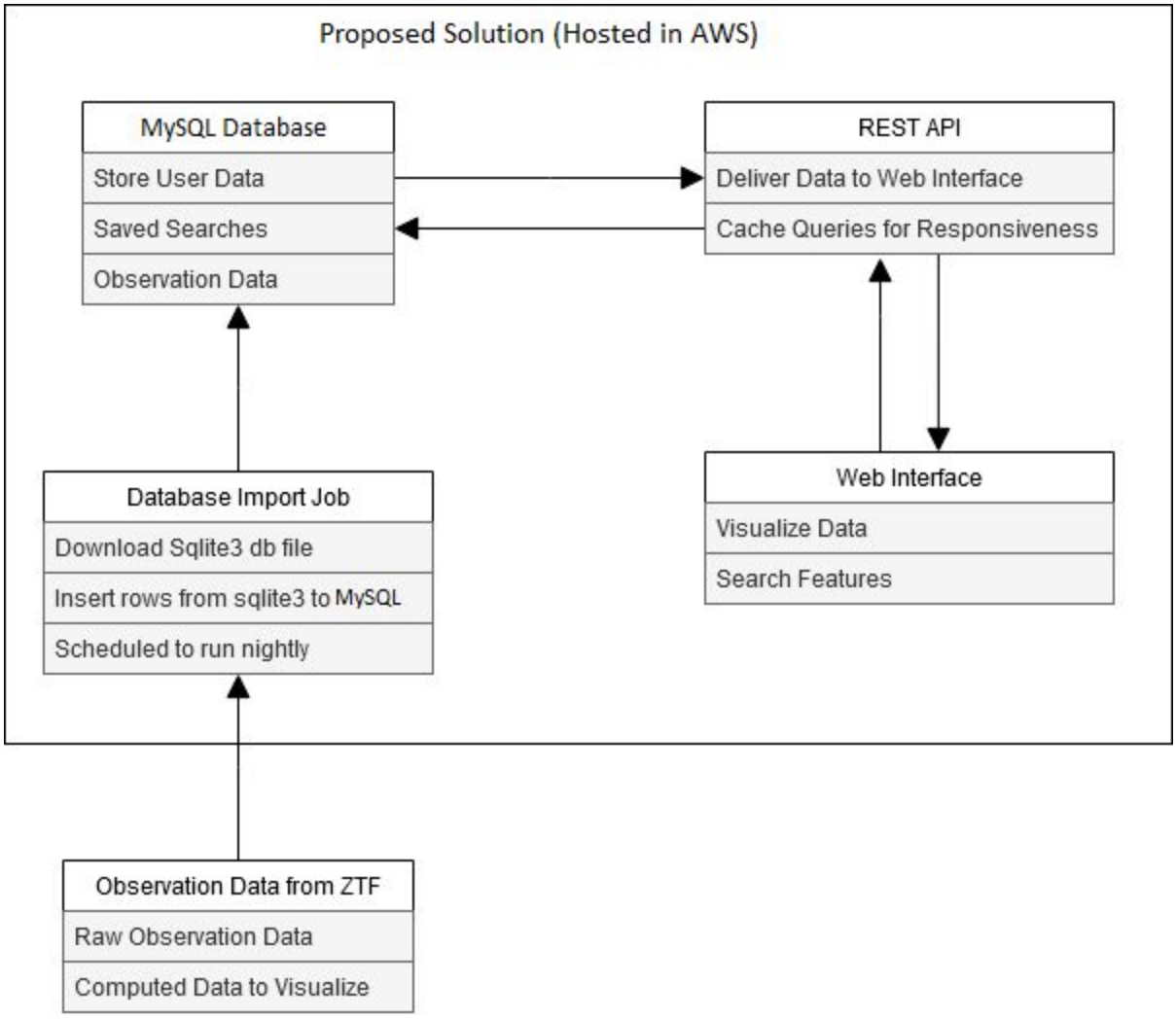
Although both Firebase Authentication and Passport.js both satisfy both of the desired characteristics, we chose Firebase because it is a service that is backed by Google and easily integrates with almost any web application. It provides many additional features already built that the team is able to use as well when building the application.

3.5.6 Feasibility

The approach we are currently thinking about taking to prove feasibility is to implement user login and logout functionality in a web application. There seem to be many tutorials detailing how to set up Firebase Authentication in a web application, so we should have no shortage of resources when implementing this with whichever Javascript framework that we decide to use.

4.Technology Integration

The proposed solution can be split into roughly three components: the web-accessible database, the REST API, and the web interface.



4.1 Database Component

Using a desktop-like hosting environment, the database import job would be scheduled and run on the same machine as the SQL server for lower latency and less otherwise avoidable network traffic. Using a low/no-overhead hosting environment like AWS or Microsoft Azure, the database import job would live separately.

The Database component is responsible for importing ZTF observation data from a sqlite3 file into a SQL server that is able to handle multiple simultaneous requests from the REST API. In addition, the database component stores user data like roles, permissions, saved searches, and login information.

4.2 REST API

The REST API may be hosted in a separate environment, but would ideally have low latency with the SQL server. The REST API's primary responsibility is to translate data visualization requests from the web interface into one or more SQL queries, delivering the relevant data points. The REST API may also be optimized to cache frequently used search queries in memory to improve responsiveness on the web interface as well as reduce the load on the SQL server.

The REST API would also manage the interpretation of user data stored in the SQL database, implementing and enforcing things like roles, logins, and sessions.

4.3 Web Interface

The web interface is primarily responsible for the *responsive* visualization of potentially hundreds of thousands of data points. Frequent calls to the REST API are expected, even when a user does not navigate to another page. For this reason, this part of the proposed solution involves integrating the React.js framework with the REST API. To accomplish this, the API will be written in C# MVC, where there will be separate subsets of controllers for frontend navigation through the site and backend API calls.

5. Conclusion

The big data revolution is coming to astronomy, and there are very few computational methods for analyzing this data as it currently stands. The knowledge to be gained from this data could lead to many powerful discoveries, but most importantly it could help us to prepare for the inevitability of an asteroid impacting Earth. So Team Ceres plans to address this problem. We want to build a scalable, responsive web application in order to aid in this revolution. We plan to use a comprehensive data visualization framework, user authentication, and importing data to a web-accessible database system in order to reach our goals.

We spent a good deal of time researching the various technologies that we could use for this project, and we feel confident in the decisions that we made. Additionally, we will be able to verify our feasibility very soon as we move into the technology demo for this project. This will give us the opportunity to move forward with the decisions that we made in this document and onto the design of our implementation. We are confident that we can reach our goal and complete this web interface for the analysis of bodies in space.

6. References

[1] CS486C – Senior Capstone Design in Computer Science Project Description - A GUI interface for large data stream analysis for all-sky astronomical measurements

[2] Microsoft.com - Microsoft Data Platform (https://www.microsoft.com/en-us/sql-server)

[3] Firebase Authentication - google.com (https://firebase.google.com/docs/auth)

[4] Passport - http://www.passportjs.org/

[5] Google Charts Guides - https://developers.google.com/chart/interactive/docs/datatables_dataviews

[6] Dygraphs - Demo Gallery - https://dygraphs.com/

[7] Chartist - API and Examples - https://gionkunz.github.io/chartist-js/api-documentation.html

[8] AWS Marketplace: SQL Server 2016 Standard with Windows Server 2016 - https://aws.amazon.com/marketplace/pp/B01M3SSA4O

[9] Relevant (Choosing a Javascript framework) - https://relevant.software/blog/angular-vs-react-vs-vue-js-choosing-a-javascript-framework-for-your-project/

[10] How to choose a proper web hosting service for your website - https://yalantis.com/blog/types-of-hosting-solutions/

[11] NAU Monsoon - https://in.nau.edu/hpc/

[12] MS SQL Server 2019 Pricing - https://www.microsoft.com/en-us/sql-server/sql-server-2019-pricing

[13] MS SQL Performance monitoring tools - https://docs.microsoft.com/en-us/sql/relational-databases/performance/performance-monitoring-and-tuning-tools?view=sql-server-ver15

[14] PostgreSQL vs MySQL - https://www.enterprisedb.com/blog/postgresql-vs-mysql-360-degree-comparison-syntax-performance-scalability-and-features

[15] MySQL Supported Environments - https://www.mysql.com/support/supportedplatforms/database.html

[16] MySQL Backups - https://kb.virtubox.net/knowledgebase/backup-your-databases-with-mysqldump/

[17] TechMagic (React vs Angular vs Vue.js) - https://medium.com/techmagic/reactjs-vs-angular5-vs-vue-js-what-to-choose-in-2018-b91e028fa91d

[18] Javatpoint (pros and cons of react) - https://www.javatpoint.com/pros-and-cons-of-react

[19] Pluralsight (Angular 101) - https://www.pluralsight.com/blog/software-development/angular-101

[20] Altexsoft (pros and cons of Vue.js) - https://www.altexsoft.com/blog/engineering/pros-and-cons-of-vue-js/

[21] Education-Ecosystem (React.js History) - https://www.education-ecosystem.com/guides/programming/react-js/history

[22] The Startup Lab (The History of Angular) - https://medium.com/the-startup-lab-blog/the-history-of-angular-3e36f7e828c7

[23] Wikipedia (Vue.js) - https://en.wikipedia.org/wiki/Vue.js#:~:text=Vue%20was%20created%20by%20Evan,in%20a%20number%20of%20projects.&text=The%20first%20source%20code%20commit,the%20following%20February%2C%20in%202014.

[24] The 5 benefits of AWS - https://sados.com/blog/aws-benefits-and-drawbacks/

[25] Corps (Microsoft Azure) - https://blog.icorps.com/pros-and-cons-microsoft-azure