Team Ceres

Software Design Document

Version 1.0

January 31, 2021

Sponsored By: David Trilling, Michael Gowanlock

Team Mentor: Fabio Santos

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

1. Introduction

The purpose of this document is to outline the software design for our project and review the expected requirements in the final product that Team Ceres creates. The document will be broken down into multiple sections outlining the process of how the various components of our product will come together, and we will further describe the general software design of these components. This document will serve as a blueprint for the general overview of how our project will function and be built.

Background

Every night astronomers across the globe participate in all-sky surveys, where the night sky is recorded in hopes to gain knowledge of the galactic entities that surround the Earth. These surveys can produce very large quantities of data and are useful for cataloging notable bodies, such as asteroids. The Zwicky Transient Facility (ZTF) in San Diego, California generates nearly 2 terabytes of data every night and that data alone is very difficult to examine due to the high rate that data is collected.

It is estimated that when the Vera C. Rubin Observatory is finished being built in 2021, 20 terabytes of data will be collected every night for the next 10 years. By the end of the Rubin Observatory's participation in these all-sky surveys, 73 petabytes of data will have been collected.

Our clients, Professor David Trilling and Professor Michael Gowanlock, are interested in using this data for their personal research but many of the interfaces that use data from the ZTF, are either outdated or not user friendly. The main problem that our clients face however, is that they have no interface available to them with functionality that is user friendly and easy to navigate. Because of this, Professor Trilling and Professor Gowanlock are interested in the development of a new interface that uses data from the ZTF and provides easy to access data that doesn't overwhelm its users on first view. Team Ceres goal is to create this graphical user interface and to create it with all of the requested accommodations to make it a valuable tool for our sponsors.

Current Standing

Right now, our clients will choose specific sections of data to import from the Zwicky Transient Facility (ZTF) observation process, mostly focusing on small bodies and asteroids. In order to analyze this data, our clients must request that a series of scripts and processes are run in order to:

1. extract the relevant data from ZTF
2. store the data in separate database files
3. run analysis on hundreds of thousands of small bodies
4. export the results to another database file
5. chart/graph the requested information using these results

These processes are run by another colleague, who then makes the relevant charts and graphs available on NAU's RCDATA portal.

The current solution in place does not provide the client with an easy way to query data and get results. This is due to the lack of user interface. The current solution is also extremely time consuming as the client needs to reach out to another colleague in charge of performing the queries and then compiling the data analysis into a usable form suitable for their needs. Another issue faced with the current solution is the responsiveness of the database. Whenever a script is run to add new charts and data, the current site directories need to be manually cleared and repopulated. This causes the site to be down momentarily.

This process is workable for our client, but leaves a few things to be desired. Namely:

- Easy access to the charts and graphs
- An intuitive way to browse different aspects of analysis for a particular asteroid
- A compiled overview of the analysis as a whole
- On-demand generation of the above charts and graphs
- A way to share the results of this analysis

- A way to mark particular areas of study as important, unique to each individual user

These are some of the key requirements that will exist within our product, and throughout the rest of this document, we will be overviewing how we intend to implement these features and flesh out more of the design of our software.

2. Implementation Overview

In order to implement the ZTF Asteroid Analysis Tool (ZAAT), there are two main components. We want to start by creating an interactive web application that allows for users to easily view, search, and analyze asteroid data. The second component we want is a back-end database and api to serve data from our database to the front-end interface. These two components will give the user all the functionality required for our product, and allow for easy upgrades and improvements to the interface without affecting the ability of the application to run. Although these are the two main components of our application, there was some breakdown of these technologies in order to allow for them to operate as optimally as possible.

The chosen technologies for this project are broken down below:

1. ZTF Asteroid Analysis Tool (ZAAT)
    a. Front-End React Application
        i. Various styling frameworks to allow for an excellent user experience
        ii. FirebaseJS to allow for user authentication, roles, and personalized preferences and bookmarks
    b. Back-End Database
        i. A live hosted database that is updated regularly to keep data accurate and consistent
        ii. An api to service requests to the database and send them back out to the front-end interface

This is a very general overview of the implementation of our product, and throughout this document we will further develop the concepts of our designs to formulate a blueprint for ZAAT. This should allow for incoming developers to understand our thought process with our designs and understand how the tool was built.

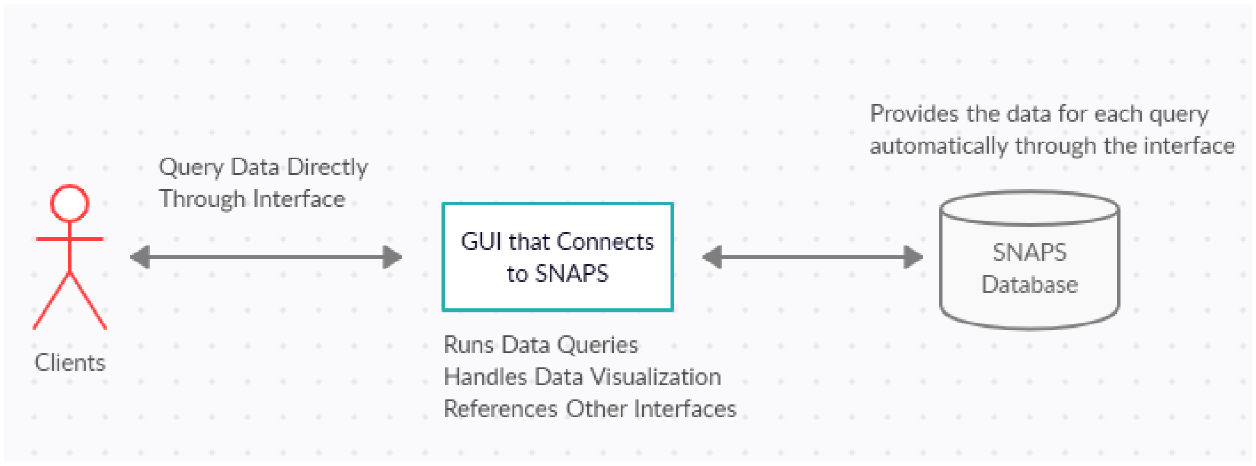

Provides the data for each query automatically through the interface
Query Data Directly Through Interface
GUI that Connects to SNAPS
SNAPS Database
Clients
Runs Data Queries
Handles Data Visualization
References Other Interfaces

Figure 1: Simple Diagram Of the ZAAT's Design

3. Architectural Overview

As seen in the diagram below, our app consists of two main components. These being the frontend user interface and the backend database. Our user interface is able to interact with the database through a rest api. The rest api will send data requests from the frontend to the database, and provide the data back to the frontend. This rest api basically acts as a middleman between the database and our web application.

When viewed in this aspect, the role of each component can very clearly be seen. The database is in charge of keeping our data accurate and consistent. This will be done through regularly scheduled batch jobs that are to import new datasets without disrupting the user's experience. The database will also be answering requests from the rest api in order to present data to our web application.

The rest api is very simply put, a middle man for our product. The rest api will receive requests from the frontend/user and then send these requests to the database or answer from cached queries.

The frontend application will be the user’s main interaction with our product. This component will be visually appealing to the users and responsive with all the actions users might perform. The frontend will display data to the user, manage the user’s authentication status to save personalized bookmarks and preferences, and allow for searching/querying of the database.

Overall, the data flow pattern of our application can best be seen through describing a use case of our application. Suppose a user is viewing asteroid data and wants to search for asteroids with a certain trait, let’s say asteroids that have above 20 observations in our database. This would mean that we have observed this asteroid enough for the user to want to see the different metrics collected over a series of observations in order to perform some form of analysis. Our user would start by going to the search page in our application, applying the appropriate filters, and sending out the request. This is where the data flow pattern of our application comes into play.

Our application starts by calling a frontend function to handle gathering the various search parameters the user provided. These parameters are then used to create a get/post request that will get the desired asteroids from our database. The frontend function passes the request to our API, which in turn is able to execute the actual query upon the database. From here, the database returns the results from the query: either a list of asteroids with the desired traits or an empty list. These results are passed to the API, which in turn sends data back to the frontend of the application. The frontend is able to parse the data received into an easily readable form for the user, and then displays these results.

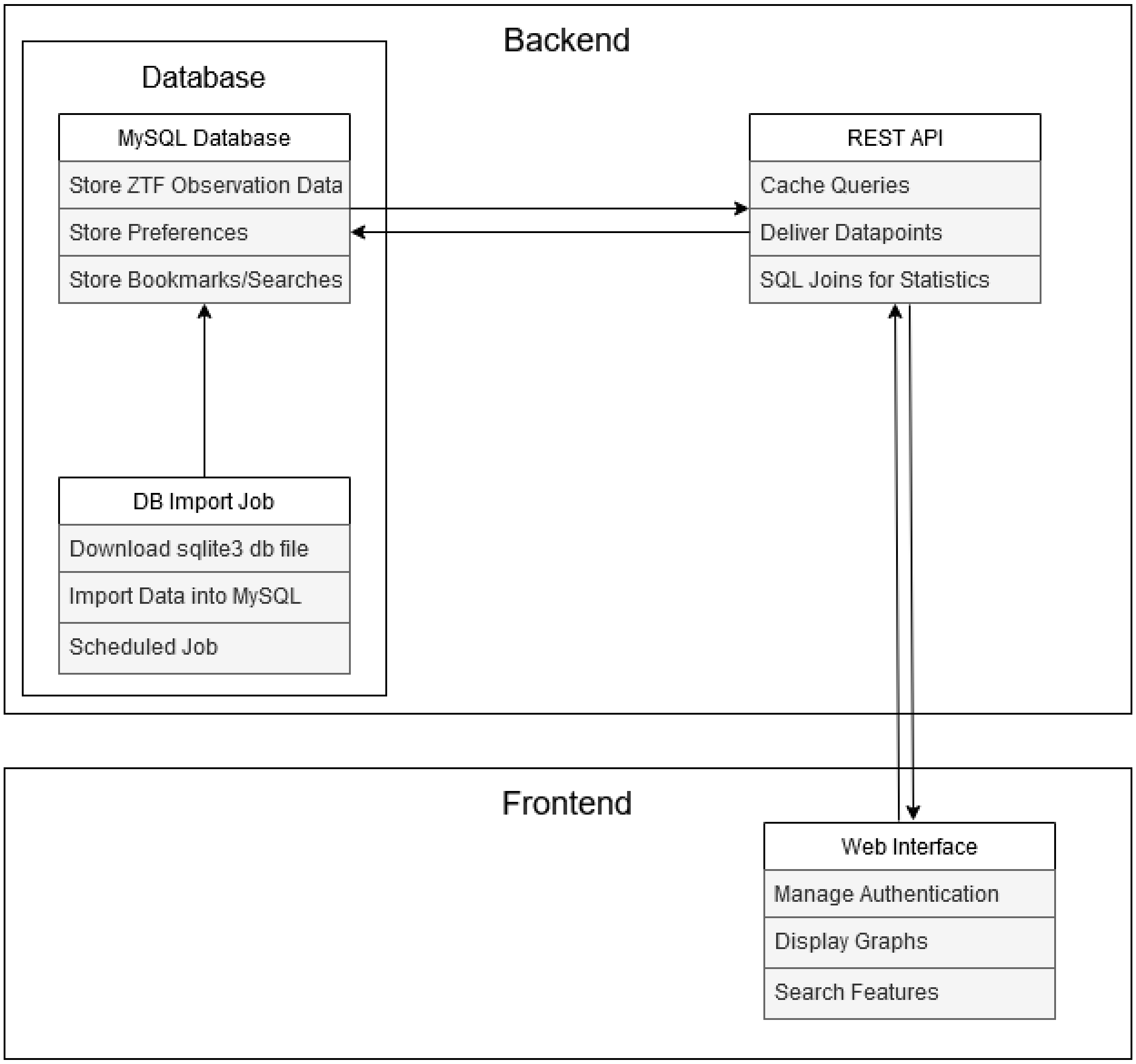


Figure 1: Architectural Layout of the Interface

4. Module and Interface Descriptions

This section will dive deeper into the architectural structure discussed above. The modules discussed in this section represent overall functionality of the application and how these modules work together to accomplish a task.

4.1 Front-end

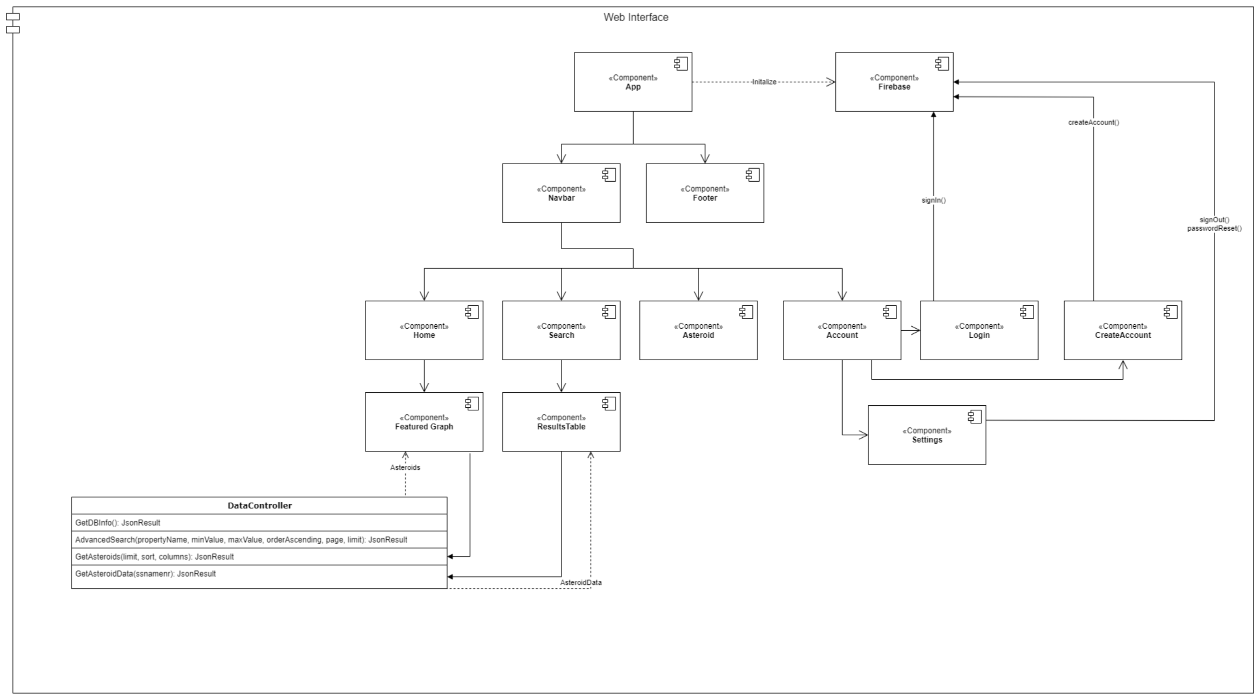


Figure 2: Diagram outlining the Front-End of the program

The front-end follows a component-based structure where the overall web application is represented with the component named "app". The breakdown of the front-end structure is as follows: The app component renders the Navigation bar and the Footer for the web application. From there, the navbar component holds routes that link to the components that make up the "meat" of the application.

These components are Home, Search, Asteroids, and Account. The Home component currently holds four featured graph components which use the REST API to grab data for the graphs (this will be discussed below). The Search component renders the ResultsTable component which also uses the REST API to gather the information from the database. The Asteroid component currently does not render any components but will hold information from the database later on.

The Account component holds three smaller components that are: Login, CreateAccount, and Settings. These three components use the Firebase API to handle all authentication to ensure that the user has a better experience.
Various components in the front-end utilize the REST API in order to query the database and gather data rendered in our web application. This REST API is called DataController and is discussed in the back-end section below. The FeaturedGraphs component uses the GetAsteroids method to gather asteroid metrics. As discussed in earlier sections, the Search component renders the ResultsTable which hits the AdvancedSearch endpoint to allow the user to filter and search for specific asteroids.

4.2 Back-end

The back-end division of the project consists of 3 modules, which work together with the ultimate goal of servicing the web interface in a timely and efficient manner. The general flow of the data is as follows: the scheduled import job runs and imports data collected by ZTF into the MySQL instance. From there, the REST API runs queries on any table and delivers relevant data points to the web interface. If a query is requested more than a (configurable) threshold within a certain time frame, the query result is cached by the API and used in place of another database connection upon any subsequent requests.

4.2.1 REST API

The REST API is an MVC-style C# API with a fully-realized abstract model of the MySQL table structure, with endpoints to deliver relevant data points to the web interface upon request for the purpose of plotting, searching, and comparisons. The main model of the REST API is the Asteroid object, with collections of Observations over time. In addition, the ZTF data component of the Asteroid object stores data not required for normal plotting purposes, yet required for a more detailed histogram of a given asteroid, and so can be excluded from less-detailed requests for asteroid information.

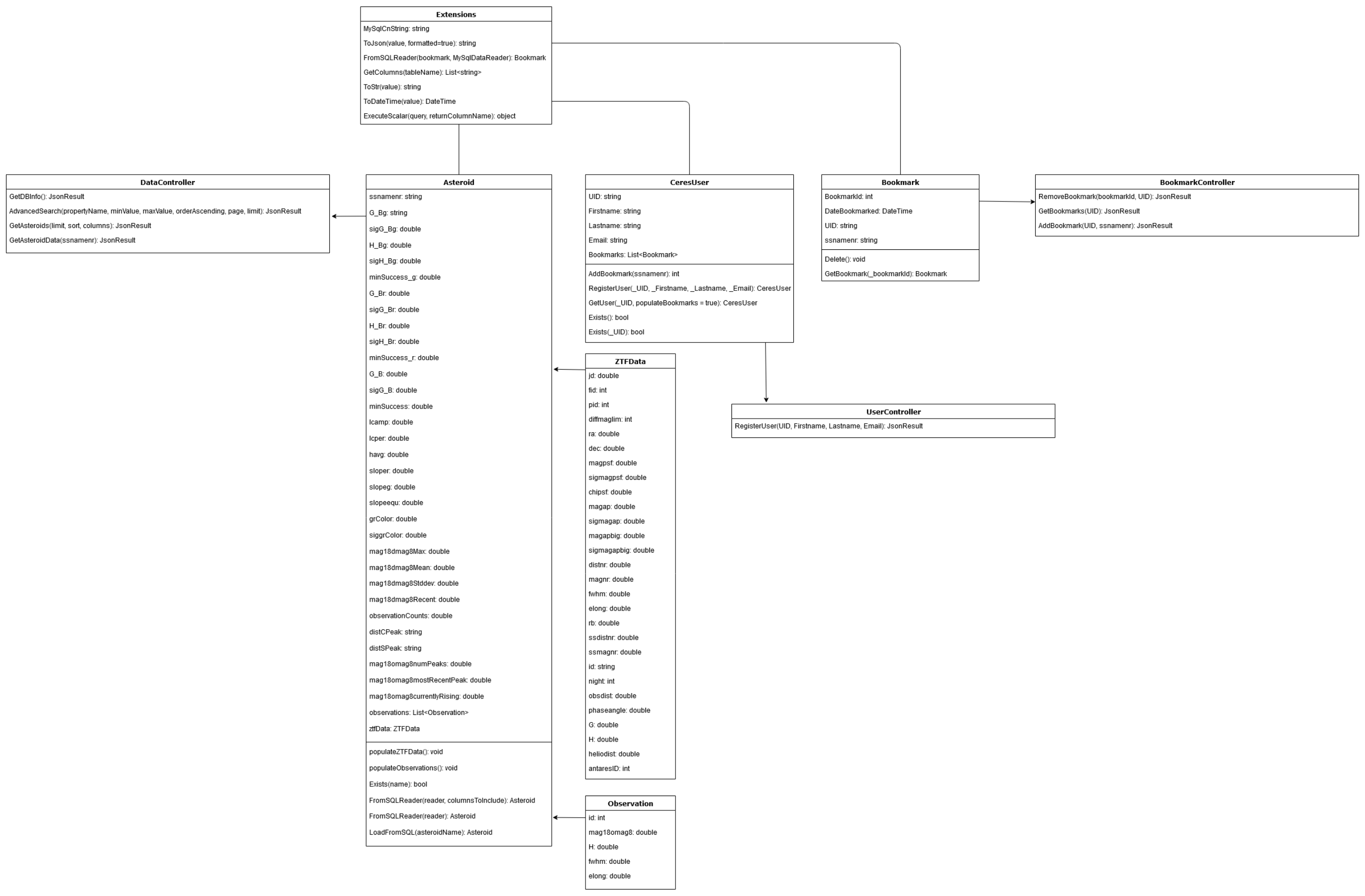


Figure 3: UML Diagram of the REST API

The main endpoints of the REST API can be divided into their respective controllers: Data, User, and Bookmark.

Data Controller

- GetDBInfo: (no parameters)

  returns a json result containing the total number of observations in the database as well as the total number of asteroids

- AdvancedSearch: (propertyName, minValue, maxValue, orderAscending, page, limit)

  The propertyName parameter refers to the column name to filter by in the database schema, the min and max value parameters refer to the numerical min and max with which the API should develop an upper and lower bound to exclude results. The orderAscending parameter is passed directly to the corresponding SQL query, and it specifies whether the list of results should be ordered by the 'propertyName' value in ascending or descending order. 'page' and 'limit' are used for the purpose of paging the search results, if desired. A 'limit' of 0 will include all results returned from the SQL query.

  The expected return of this endpoint is a list of generic type objects, each with the properties 'ssnamenr' to uniquely identify the asteroid, and 'column' to indicate the value of the requested propertyName.

- GetAsteroids: (limit, sort, columns)

  The 'limit' parameter will limit the number of results returned by the API. 'sort' refers to any built-in method of sorting asteroids. 'columns' is a way to explicitly specify which columns should be returned from the API for each asteroid and if left null, the API will return all properties associated with every asteroid.

  The expected return of this endpoint is a list of asteroid objects, in JSON format, with either every property or, optionally, just the properties defined in the 'columns' parameter. This list is not sorted by default, but can be sorted if the 'sort' parameter is passed with a valid value.

- GetAsteroidData: (ssnamenr)

  The only parameter for this endpoint refers to the unique identifier of the asteroid, used to query the ‘asteroids’ table in MySQL.

  The expected return of this endpoint is a single generic type object in JSON format that contains properties of the requested asteroid, as well as a list of observations from the MySQL ‘ztf’ table.

User Controller

- RegisterUser: (UID, Firstname, Lastname, Email)

  These parameters are used to insert a row into the ‘users’ table in MySQL, and the primary key of this table is used to access bookmarks for any given user.

  The expected return of this endpoint is the unique identifier (GUID) assigned by the MySQL auto-increment function on the primary key of the users table.

Bookmark Controller

- RemoveBookmark: (bookmarkId, UID)

  This API endpoint will remove a user created bookmark given the corresponding bookmark id and the user’s UID.

  The expected return of this endpoint is either a http status 500 (if a bookmark/user does not exist or the specified user has no bookmark by the provided Id) or an http status 200 if the bookmark was successfully removed.

- GetBookmarks: (UID)

  Given a user UID, this method will return a JSON-encoded list of all user bookmarks and their properties. If a user does not exist with the provided UID, the endpoint will return an http status 400 along with a status description of this condition.

- AddBookmark: (UID, ssnamenr)

  Given a user UID and asteroid unique identifier, this method will insert a row into the ‘bookmarks’ table in SQL. The expected return of this method is a JSON-encoded object containing the ‘BookmarkId’ of the created bookmark as its only property. If an asteroid with the corresponding ssnamenr does not exist, or a user with the corresponding UID does not exist, the endpoint will return an http status 400 along with the appropriate status description if either of these conditions are met.

4.2.2 Database

The database section of the backend of the application consists of two components: the MySQL RDBMS and the scheduled import job.

The import job is a command line utility written in C# that is highly configurable. The associated configuration file allows specification of a custom working directory, changeable urls to download the sqlite3 .db files, as well as a provision for a different MySQL username and password in the event that the database ever needs to be migrated or the credentials need to be changed. In addition, the import utility accepts a custom database schema JSON-formatted document that allows it to account for slight modifications to database schema without modifying the C# code and recompiling.
The import job is a self-contained and relatively simple utility, with few methods and straightforward operation. The methods it uses are:

- RandomFilename: (no parameters)

returns a random, unique file name in the current working directory

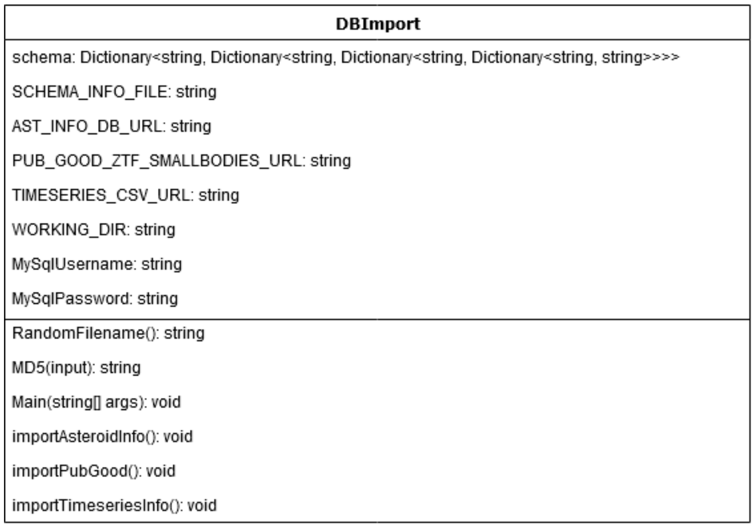
- MD5: (input)
  This method is included as a way to simplify the call to the basic encryption API provided by .NET and return an MD5 hash in string form

- Main: (string[] args)
  The main driver of the program. Downloads appropriate files from configured URLs, verifies configuration file integrity, and calls the below methods in listed order.

- importAsteroidInfo: (no parameters)
  Imports data from a sqlite3 file into the ‘asteroids’ table in MySQL.

- importPubGood: (no parameters)
  Imports data from a sqlite3 file into the ‘ztf’ table in MySQL.

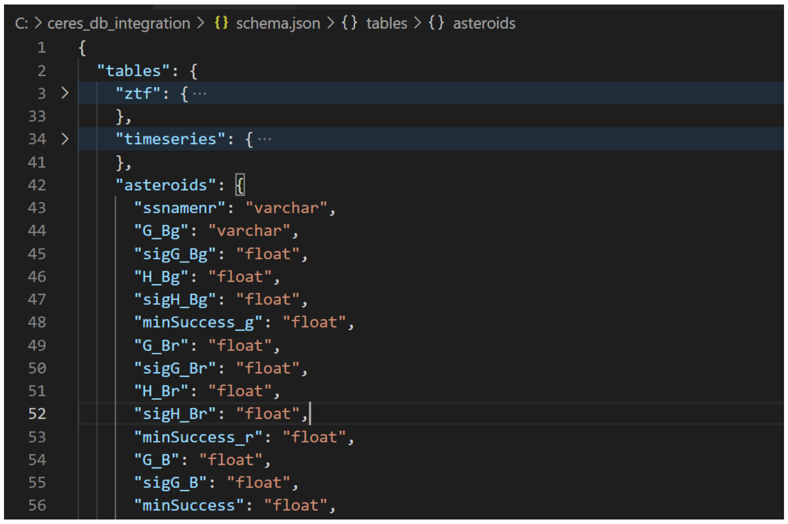
- importTimeseriesInfo: (no parameters)
  Imports data from a CSV file into the ‘timeseries’ table in MySQL.



DBImport

```
schema: Dictionary<string, Dictionary<string, Dictionary<string, Dictionary<string, string>>>>
SCHEMA_INFO_FILE: string
AST_INFO_DB_URL: string
PUB_GOOD_ZTF_SMALLBODIES_URL: string
TIMESERIES_CSV_URL: string
WORKING_DIR: string
MySqlUsername: string
MySqlPassword: string
-------------------------------------
RandomFilename(): string
MD5(input): string
Main(string[] args): void
importAsteroidInfo(): void
importPubGood(): void
importTimeseriesInfo(): void
```

Figure 4: Database Import Job UML Diagram



C: > ceres_db_integration > {} schema.json > {} tables > {} asteroids

```
{
  "tables": {
    "ztf": {⋯
    },
    "timeseries": {⋯
    },
    "asteroids": {
      "ssnamenr": "varchar",
      "G_Bg": "varchar",
      "sigG_Bg": "float",
      "H_Bg": "float",
      "sigH_Bg": "float",
      "minSuccess_g": "float",
      "G_Br": "float",
      "sigG_Br": "float",
      "H_Br": "float",
      "sigH_Br": "float",
      "minSuccess_r": "float",
      "G_B": "float",
      "sigG_B": "float",
      "minSuccess": "float",
```

Figure 5: A snippet of the schema.json file used by the import utility

The MySQL instance is highly configurable as well, with a relatively straightforward table structure corresponding to the initial format laid out in the sqlite3 files. Since the REST API will cache frequently requested queries, the load on the MySQL instance will be greatly reduced.

5. Implementation Plan

Provided below is the implementation plan for our project. In the previous semester, our team made significant development on our product. This allowed us to come into this semester with a very solid foundation for our web application. This allows us to have a very straightforward implementation plan for this semester. Overall as a team, we intend to follow this plan and get our project done well for our clients. Since tasks are very linear, we are able to make modifications using feedback from our clients throughout the semester.

Our main plan of action for this semester is as follows: we want to start with building out the remaining key features of our project being the searching features and creating an interactive analysis tool. Once these are built out, the rest of the time spent will be personalizing the tool, polishing up the interface, ensuring we eliminated all bugs, and getting ready to present our final project. This might sound like we are oversimplifying the implementation of our product; however, using the plans outlined throughout this document the development of this product should go smoothly as we function optimally as a team.

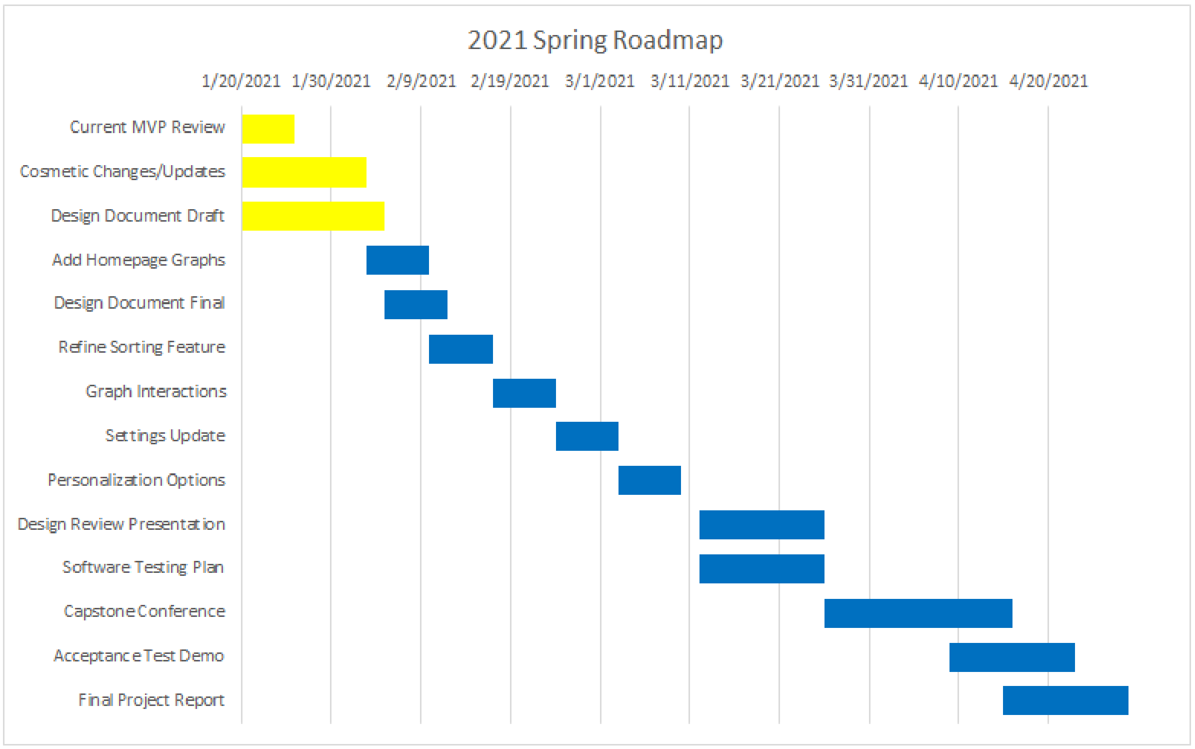


Figure 6: Team Ceres Spring 2021 Schedule

Add Homepage Graphs - Assigned to Joseph

This task will add additional graphs to the ZAAT interface so users can be presented with the new data collected from the Zwicky Transient Facility data stream. The graphs presented will be specifically requested by the clients and may have the chance to have options on which data the user wants displayed.

Refine Sorting Feature - Assigned to Team Ceres

The task to refine the sorting feature on the ZAAT will present users with an easy to read and easy to use search interface that filters data via column name, minimum and maximum values, and may potentially be able to sort the difference between two data values for more specific searches.

Graph Interactions - Assigned to Javier

The graph interactions will allow users to scale the graphs to be more readable in the case that the data presented is hard to read, most likely due to the volume of data being loaded. It will also allow users to click on any data points on the graph's scatter plots to take the user to an asteroid's detailed information page.

Settings Update - Assigned to Zach

The settings update will provide users with more useful tools to make searching on the interface easier. It will provide a menu that users can use to adjust what graphs appear on the home page, options to change account information, and other tools that the clients request as development progresses.

Personalization Options - Assigned to Miles

The last major update will allow users to personalize their account to save specific asteroid pages and data by keeping a journal for them to use. This journal will keep track of any previous asteroids visited by the users and will give them the opportunity to follow the data any certain asteroids provide.

The remaining tasks not mentioned above will all be handled by the members of Team Ceres so we can complete them effectively and efficiently for the clients. All of the tasks above are subject to change, but based on recent discussion with the clients, this is the current vision that is laid out.

6. Conclusion

Ultimately, we are feeling very confident in our ability to provide our clients with a product they will like. As it currently stands, it will soon be impossible to perform analysis on the amount of data being pulled in about asteroids and other small bodies in space. The knowledge gained from this analysis is invaluable though; we could learn about the formation of our solar system and the conditions that led to our civilizations' formation. Additionally, we could better prepare for the inevitability of an asteroid impacting Earth.

That’s where Team Ceres comes in. We plan to give our clients, Professor Trilling and Professor Gowanlock, the tool to be able to perform this analysis. With this, our clients would be able to perform research with ease and efficiency. Currently our clients work with another colleague in order to perform more complex analysis on the database of asteroids that exists, and this is incredibly inefficient. That means that once this database begins to grow exponentially, their research becomes nearly impossible to complete. And that is why our tool is necessary for them. The ZAAT would be some of the first technology of its kind, and could be beneficial to astronomical observers all around the world. With this tool, researchers would be able to perform research at a much faster pace and come to conclusions in record times.