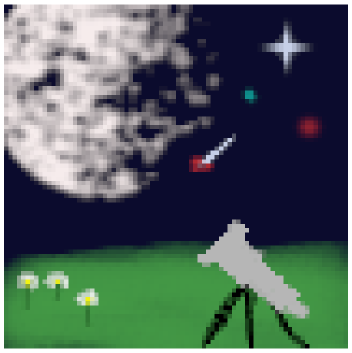
Team First Light



Final Report
Version 1.0

5 May 2022

Sponsors

Dr. David Trilling

Dr. Mike Gowanlock

Faculty Mentor

Felicity Escarzaga

The Team

Matt List- mjl79@nau.edu

Carson Pociask - cmp557@nau.edu

Jakob Nelson - jrn235@nau.edu

William Fuertes- wf69@nau.edu

Table of Contents

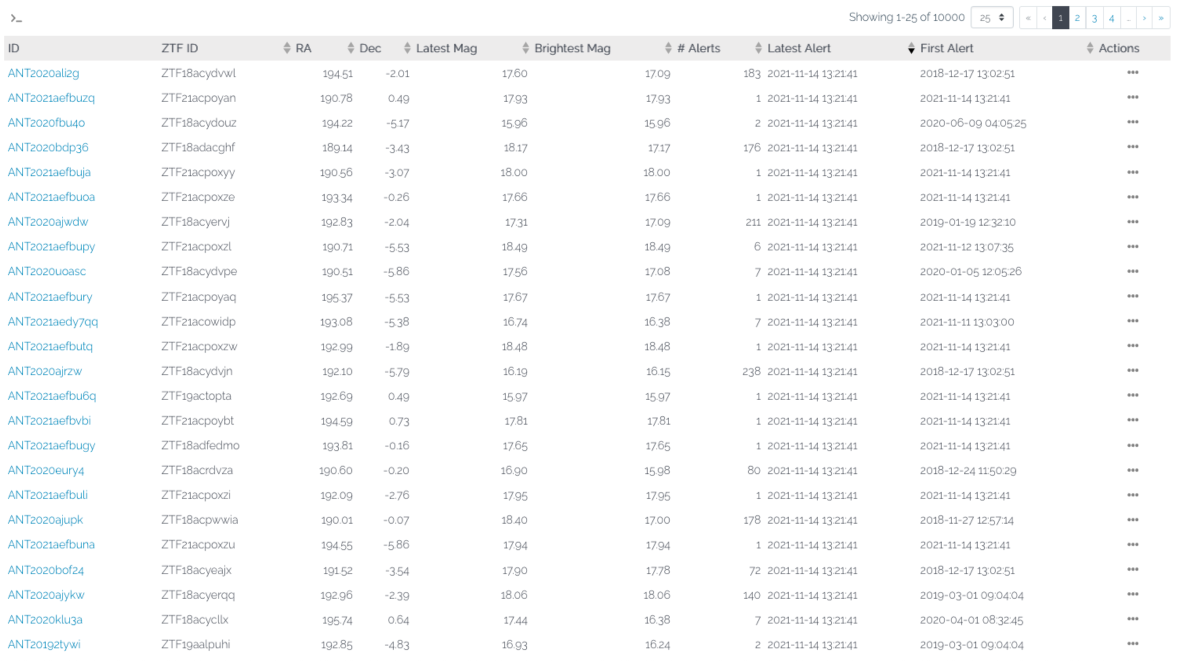
1.0 Introduction

Big data is a relatively new field in technology. The process of extracting information from sets of data too massive to be handled by ordinary technologies can solve difficult business problems. Solving these problems are at the forefront of many of today's technological efforts, and while efficient tools already exist to send, capture, and store large sets of data, the need to perform analysis on them is growing. Forbes predicts that more than 175 zettabytes will need analysis by 2025 (Coughlin, 2018). Researchers, businesses, and everyday people are responsible for today's massive growth in data creation. In order to fully utilize massive amounts of data, it is important that the technologies used to support its operations keep up with the demand of today's growth in data. While it may seem like most of the world's big data technologies are cutting edge, some fields still lack the support needed to draw useful information out of collected data. One of the fields that experiences drawbacks in workflow due to this lack of support is astronomy.

The field of astronomy is rapidly expanding and the data pertaining to night sky observations continues to grow in size, on the scale of terabytes and petabytes. Analyzing astronomical data is becoming increasingly complex and has become much more involved with data science as a whole. An example of this growth in today's world is the Vera C. Rubin Observatory. Currently being constructed on Cerro Pachón in Chile, this site, when operational, will produce roughly 20 terabytes of data every night under the 10-year Legacy Survey of Space and Time (LSST). The telescope will be taking all-sky observations, meaning that astronomical data will be compiled from all visible parts of the sky on a nightly basis. Different researchers across the field of astronomy will handle some percentage of this produced data, and Northern Arizona University (NAU) will be ingesting datasets pertaining to asteroids. The clients that will support this team are professors at NAU: Dr. David Trilling, Professor of Astronomy and Planetary Science, and Dr. Mike Gowanlock, Assistant Professor at the School of Informatics, Computing, and Cyber Systems (SICCS). Both clients perform analysis on the observational data pertaining to asteroids, or rocky bodies in the solar system that are tracers of the formation and evolution of the solar system. Since construction in Chile is still underway, a testbed is being used from the Zwicky Transient Facility (ZTF) located in San Diego, which is currently operating in a fashion similar to the LSST. Once arrived into NAU's database, both Dr. Trilling and Dr. Gowanlock utilize the captured data to draw out further information to perform research and simple observational work. The tracking of asteroids' characteristics over time is a common task for the clients and is useful for learning more about the specifics of the solar system. These observations and the research conducted on the collected data has the ability to help researchers further understand the history and current state of the solar system. Tracking asteroids as they travel across the night sky also helps predict the likelihood of asteroids coming in close contact with the earth.

As previously mentioned, there are already well-built tools in place that are able to capture and transfer very large sets of data. However, once astronomical data is gathered and stored, there exists a bottleneck. This restriction in workflow is caused by the need for data to be analyzed with the use of visualizations to provide more in-depth and easily accessible information. However, the current solutions are lacking in quantity and quality. The workflow of the clients deals with various characteristics derived from observed asteroids, such as magnitude or size. Comparing these characteristics over time using rudimentary graphical tools can prove difficult and problematic when very large data sets are to be considered, on the order of terabytes produced on a daily basis. Currently, utilizing this derived data during research is time consuming to use and difficult to understand due to the lack of graphical support. This greatly hinders the workflow of the clients and other astronomers; the data is collected yet an efficient way to utilize it for useful research is lacking. While the tools that the clients currently use do include visualizations of data, they are lackluster and will not be capable of handling the massive amounts of data that will eventually be used from the Vera C. Rubin Observatory.

A couple of example sites that the clients have previously dealt with encompass the main issues just discussed. The sites are ANTARES (antares.noirlab.edu/loci), and MARS (mars.lco.global/), and they are both centered around collecting and displaying massive astronomical data sets. The main list on the home page of ANTARES can be observed in **Figure 1.1**. A massive list of different asteroids can be observed here, where each has its own link to more in depth information. While the site is organized, the way that the large amount of information is displayed to the user is very overwhelming and uninviting. It is difficult to spot outliers or trends among the asteroids listed.



Showing 1-25 of 10000

ID	ZTF ID	RA	Dec	Latest Mag	Brightest Mag	# Alerts	Latest Alert	First Alert	Actions
ANT2020ali2g	ZTF18acydvwl	194.51	-2.01	17.60	17.09	183	2021-11-14 13:21:41	2018-12-17 13:02:51	•••
ANT2021aefbuzq	ZTF21acpoyan	190.78	0.49	17.93	17.93	1	2021-11-14 13:21:41	2021-11-14 13:21:41	•••
ANT2020fbu4o	ZTF18acydouz	194.22	-5.17	15.96	15.96	2	2021-11-14 13:21:41	2020-06-09 04:05:25	•••
ANT2020bdp36	ZTF18adacghf	189.14	-3.43	18.17	17.17	176	2021-11-14 13:21:41	2018-12-17 13:02:51	•••
ANT2021aefbuja	ZTF21acpoxyy	190.56	-3.07	18.00	18.00	1	2021-11-14 13:21:41	2021-11-14 13:21:41	•••
ANT2021aefbuoa	ZTF21acpoxze	193.34	-0.26	17.66	17.66	1	2021-11-14 13:21:41	2021-11-14 13:21:41	•••
ANT2020ajwdw	ZTF18acyervj	192.83	-2.04	17.31	17.09	211	2021-11-14 13:21:41	2019-01-19 12:32:10	•••
ANT2021aefbupy	ZTF21acpoxzl	190.71	-5.53	18.49	18.49	6	2021-11-14 13:21:41	2021-11-12 13:07:35	•••
ANT2020uoasc	ZTF18acydvpe	190.51	-5.86	17.56	17.08	7	2021-11-14 13:21:41	2020-01-05 12:05:26	•••
ANT2021aefbury	ZTF21acpoyaq	195.37	-5.53	17.67	17.67	1	2021-11-14 13:21:41	2021-11-14 13:21:41	•••
ANT2021aedy7qq	ZTF21acowidp	193.08	-5.38	16.74	16.38	7	2021-11-14 13:21:41	2021-11-11 13:03:00	•••
ANT2021aefbutq	ZTF21acpoxzw	192.99	-1.89	18.48	18.48	1	2021-11-14 13:21:41	2021-11-14 13:21:41	•••
ANT2020ajrzw	ZTF18acydvjn	192.10	-5.79	16.19	16.15	238	2021-11-14 13:21:41	2018-12-17 13:02:51	•••
ANT2021aefbu6q	ZTF19actopta	192.69	0.49	15.97	15.97	1	2021-11-14 13:21:41	2021-11-14 13:21:41	•••
ANT2021aefbvbi	ZTF21acpoybt	194.59	0.73	17.81	17.81	1	2021-11-14 13:21:41	2021-11-14 13:21:41	•••
ANT2021aefbugy	ZTF18adfedmo	193.81	-0.16	17.65	17.65	1	2021-11-14 13:21:41	2021-11-14 13:21:41	•••
ANT2020eury4	ZTF18acrdvza	190.60	-0.20	16.90	15.98	80	2021-11-14 13:21:41	2018-12-24 11:50:29	•••
ANT2021aefbuli	ZTF21acpoxzi	192.09	-2.76	17.95	17.95	1	2021-11-14 13:21:41	2021-11-14 13:21:41	•••
ANT2020ajupk	ZTF18acpwwia	190.01	-0.07	18.40	17.00	178	2021-11-14 13:21:41	2018-11-27 12:57:14	•••
ANT2021aefbuna	ZTF21acpoxzu	194.55	-5.86	17.94	17.94	1	2021-11-14 13:21:41	2021-11-14 13:21:41	•••
ANT2020bof24	ZTF18acyeajx	191.52	-3.54	17.90	17.78	72	2021-11-14 13:21:41	2018-12-17 13:02:51	•••
ANT2020ajykw	ZTF18acyerqq	192.96	-2.39	18.06	18.06	140	2021-11-14 13:21:41	2019-03-01 09:04:04	•••
ANT2020klu3a	ZTF18acycllx	195.74	0.64	17.44	16.38	7	2021-11-14 13:21:41	2020-04-01 08:32:45	•••
ANT2019ztywi	ZTF19aalpuhi	192.85	-4.83	16.93	16.24	2	2021-11-14 13:21:41	2019-03-01 09:04:04	•••

Figure 1.1 - ANTARES Site Home Page

To bring all these problematic points together is helpful in defining a set of problems that the clients sought to have a solution to, and ANTARES acts as a good reference point. In a concise summary, the problems that the clients and other astronomical researchers are experiencing are as follows:

- Lack of visual support, i.e plots, charts, and graphs of large data sets
- Lack of interactivity among large data sets
- Uninviting website pages, i.e massive lists of data points that are difficult to navigate
- The clients' current solution is unusable due to server hosting conflicts.

The solution for this project is a data dashboard capable of supporting visualization and interactivity among cleanly represented astronomical data sets. This solution will serve as a simple and easy to use base for astronomical observation and research, specifically on asteroids. In further detail, the solution encompasses the following features to address the problems at hand:

- Visualization of large data sets through graphs, charts, plots, etc.
- Interactivity of data points within visualizations
- Simple web interface with easy to navigate web pages that pertain to different modules, i.e home page with large data summary vs. page with specifics of chosen asteroid

The ability to pull astronomical data from the database at NAU is crucial as said data is the center point of this project. While the computational operations such as deriving characteristics from asteroids is already complete, the original data still needs to be retrieved from the database in order to continue onto visualization and interactivity. Once the data is retrieved, storing the large sets in memory will need to be considered. One of the most important aspects of the envisioned solution is creating graphical representations on a web page out of the large data sets from tabular structures. This operation will address one of the most crucial problems of the clients, a lack of data visualization. Having data in graphical formats rather than long lists helps ease the research and observation process for the clients and alike researchers.

2.0 Process Overview

From the start of the capstone project back in August of 2021, the team established roles that each member would take on in order to keep structure throughout the development process. In specific, Matt List took on the responsibilities of client communication and team organization as team lead, Carson Pociask as the recorder took responsibility for keeping documentation in order and up to date, Jakob Nelson kept the team's GitHub organized as the release manager, and William Fuertes helped design a high-level view of the product as the architect. The creation and assignment of these diverse roles helped the team define the different tasks that would arise throughout the capstone project. An aspect of any team project in software is the upkeep of communication between all involved parties. From the start, the team established multiple meetings that occurred on a weekly basis. The three meetings were as follows: mentor meetings every Tuesday afternoon, client meetings every Friday morning, and team meetings multiple times per week. This consistent communication helped establish a thorough understanding between all parties of what was complete and what still needed to be accomplished. As mentioned previously, this is crucial to ensure success and agreement and to avoid any gaps in understanding for any part of the project.

Once the team got to the tail end of the Fall 2021 semester, actual development of the product began. It was at this point that the team established another set of communication requirements to be followed going into the latter portion of the project. The same mentor and client meetings were kept in place, however the team agreed to meet more frequently as development of the project ramped up. These meetings carried into the Spring semester of capstone and supported the team in keeping an organized structure.

Once development of the web application began, the team became more acquainted with using GitHub, a version management system that supports software organization. Jakob the release manager ensured that the team's repository was clean and up to date on a consistent basis. Whenever there was a change to the code, GitHub was used to reflect the changes and store an up-to-date version of the project to continue work off of. GitHub was a key piece in the development process for keeping the project and team organized. Discord, a chat room software, served as the main communication point among the team and with the mentor. Discord allows for quick and convenient messaging, voice chatting, and file sharing. Lastly, Zoom, a virtual meeting software, was used as a communication point with the clients at every Friday meeting. The combination of these team standards and technologies were used throughout the entire project and proved to be successful tools in the development process.

3.0 Requirements

Requirements acquisition began at the start of the Fall 2021 semester. Weekly meetings with the team's clients allowed for a routine look into what was expected as a final product. Multiple example dashboards, such as ANTARES (antares.noirlab.edu/loci) and MARS (mars.lco.global/), were provided by the clients to spur ideas of different modules to be implemented. To start, the domain-level requirements must be established. This will allow for more specific and tailored requirements to be explained in subsequent sections to ensure full coverage and understanding of what the clients and the team envisioned as a final product. The domain-level requirements defined for this project are as follows:

- **Visualizations of large percentages of the observations in the database** - A high-level representation of the database should be readily available to users.
- **Interactivity among large data visualizations** - Data points should be able to be selected out of a large representation of data.
- **Ease of use** - Product should provide ease of use when navigating different data on web pages. Product needs to be simplistic in nature so as to not overwhelm users.
- **Filtering the data** - Datasets should be able to be filtered and visualized on the basis of different characteristics such as magnitude.
- **Tracking asteroids over time** - Asteroids should have the ability to be saved by a user to serve as a consistent reference point.
- **Widely accessible** - Product needs to be web-based to provide access to anyone who wishes to view and/or interact with the data. This also expands the reach to those who did not have access to the data beforehand.
- **Up-to-date data** - Product needs to display the most current observations possible as new astronomical data is to be generated on a daily basis.

Now that the domain-level requirements have been laid out, each can be further evaluated and explained. The following subsections entail the functional and non-functional requirements. Each of which may branch into further requirements.

3.1 Functional Requirements

Some of the most important aspects for a successful delivery of this project revolve around the functionality the solution will support. The following section will look into the functional requirements for this project in a hierarchical manner beginning with high level requirements, thus providing the ability for other, more specific requirements to be branched off and explained in detail. As an overview, the functional requirements that will be looked into are as follows:

- Simplistic dashboard design to minimize clutter and improve navigation and visualization
- Visualize millions of objects
- Ability to interact with individual asteroids
- Ability to save/track specific asteroids
- Account creation and access in order to store asteroids for users

3.1.1 Simplistic Dashboard

This project operates around a web application shared by potentially hundreds of astronomical researchers. A dashboard serves as the center point in this web application to provide functionality among the astronomical data. This requirement was important to complete successfully as it is an aspect that is commonly done poorly in current solutions used by the clients and other astronomical researchers. Referring back to **Figure 1.1**, the overwhelming amounts of data displayed is not appealing to interact with. The dashboard created needed to be simplistic in structure and style. The dashboard also had the need to be organized, with links to other parts of the website available so that the main home page does not get cluttered. The sub-requirements of a simplistic dashboard include the ability to save and track asteroids through account creation and modification, access to downloadable data, and access to pages through links on the home page. These are further described below:

1. **Account Creation**
   An important sub requirement is that users should be able to have an account within the web application. Users are able to create an account using a chosen username, email, and password combination. An example of a feature for an account is the ability to link an email address and receive notifications of changes to the website's data. Note that this is an example and was not implemented in the final solution for this project.

2. **Saving and Tracking Asteroids**
   The ability to reference back to asteroids supports ease of use in the clients' and other researchers' workflow. A user of the solution is able to save specific asteroids and easily access them through their account. This acts as a bookmark that the user can use on asteroids of their choosing. As understood from the clients, research will vary in astronomy, even among the asteroids that are utilized in this project. This means that different researchers will want to utilize the solution for their specific purpose(s). This requirement makes it possible for a user to bypass functionality and web pages that do not contain what they are looking for.

3. **Downloadable Data**
   The ability to download data in various formats is another sub requirement. The user of the solution is able to download various file formats pertaining to chosen asteroids. In example, a user is able to download a PNG file of a scatterplot pertaining to an asteroid.

4. **Account Login**
   A user is able to login with their previously created credentials. A sign in page is available on the home screen of the web application.

5. **Links to other Pages**
   On the homepage of the web application, there are links to other pages. These pages will host modules such as account information or asteroid information.

3.1.2 Data Visualization

The main aspect of this project's solution is visualizing large amounts of asteroids. Data visualizations will change how astronomical researchers view asteroids and support their research. As mentioned before, astronomical researchers have very few ways to quickly look at asteroid data and come up with inferences based on that said data without manually checking each individual asteroid and the data that is associated with it. Without the visualizations, research surrounding asteroids is slow and tiresome. Various visualizations have been created out of the astronomical data. Some sub requirements of the data visualization requirement include interactivity among various visualizations, a heatmap that groups asteroids into regions, as well as a scatter plot that will then be used to chart each asteroid in a specific region. Below, the sub requirements of data visualization are described.

1. **Interactivity Among Datasets**
   Visualizations are very important to this project and provide for a great point of data reference. However, interactivity within the visualizations expands the reach and overall functionality of the product, so it has also been implemented as a sub requirement of data visualization. The ability to interact with data points also allows for more effective in-depth research and observations to take place. Interacting with data points within a visualization provides for better ease of use.

   Scenario in Workflow : User is at the home page of the web application where a snapshot of the database is displayed in a visual. While observing, the user notices a group of asteroids that have unusually high magnitude and is interested in learning more. Instead of digging through thousands of data points in an attempt to locate the particular asteroid, the user simply creates a scatter plot in the desired range of values in order to see more about those asteroids. This saves a lot of time in the workflow as less time is spent manually searching extensive, uninviting lists of asteroids.

   Not only does interactivity among visuals improve ease of use in a workflow, it also creates a welcoming environment for the product. An easy to use solution supported by interactivity among visualizations can help draw in more researchers to NAU who wish to use the created tools.

2. **Heatmap Visualization**
   A sub requirement of overall data visualization is the creation and use of heatmaps. A heatmap represents data using color codes and can differentiate asteroid counts quite easily. For example, a heatmap can serve as a high level view of the database.

3. **Scatter Plot Visualization**
   Another sub requirement of data visualization, similar to the previous, is the creation and use of scatter plots. Scatter plots are another form of visualization that are used in this project, serving as a representation of attributes over time.

3.2 Non-functional (Performance) Requirements

In the context of this project, non-functional requirements play an important role in the overall success and usability of the final product. The amount of data handled during development is large enough that ordinary data tools are not feasible. When dealing with massive amounts of data points, performance is often a bottleneck in the workflow of the user. Thus, it is very important to consider and thoroughly explain the non-functional requirements for this project. These requirements are listed below and will be evaluated throughout this section.

- Ability to visualize data sets within a reasonable amount of time
- Ability to handle multiple users making visualization requests simultaneously

3.2.1 Efficient Dataset Visualization

The database currently contains gigabytes of data, and the solution will need to create multiple graphs using this data. A page that loads slowly will frustrate users, which will decrease the usability and desirability of the solution page. Because presenting plots to the end user will be dependent on how quickly the application can receive data from the database, the team will require plots to be built in a reasonable amount of time. Additionally, the application needs to draw the larger graphs, such as the summary heatmaps.

3.2.2 Ability to Handle Multiple Users

Because this application will be publicly available, it is conceivable that multiple users will access this site at the same time. While a large number of users at once is not expected, as this site will be primarily used by a niche audience, the team needs to be aware of the cost of running multiple users, and to do so without significant loss of performance.

4.0 Architecture and Implementation

In order for work to continue on this project, it is important that there is a thorough description of what has been built. This way, whoever takes on the next steps in development for this web application will not be confused at any point as to what is happening at any point in the system. Before diving into the finer details of functionalities built over the last six months, a higher level view of the application must be presented. **Figure 4.1**, as seen below, is a high-level view of the entire system including the different modules implemented and routes that a user can take throughout the application. Take note of the linking of the different modules to better understand the flows a user can navigate.

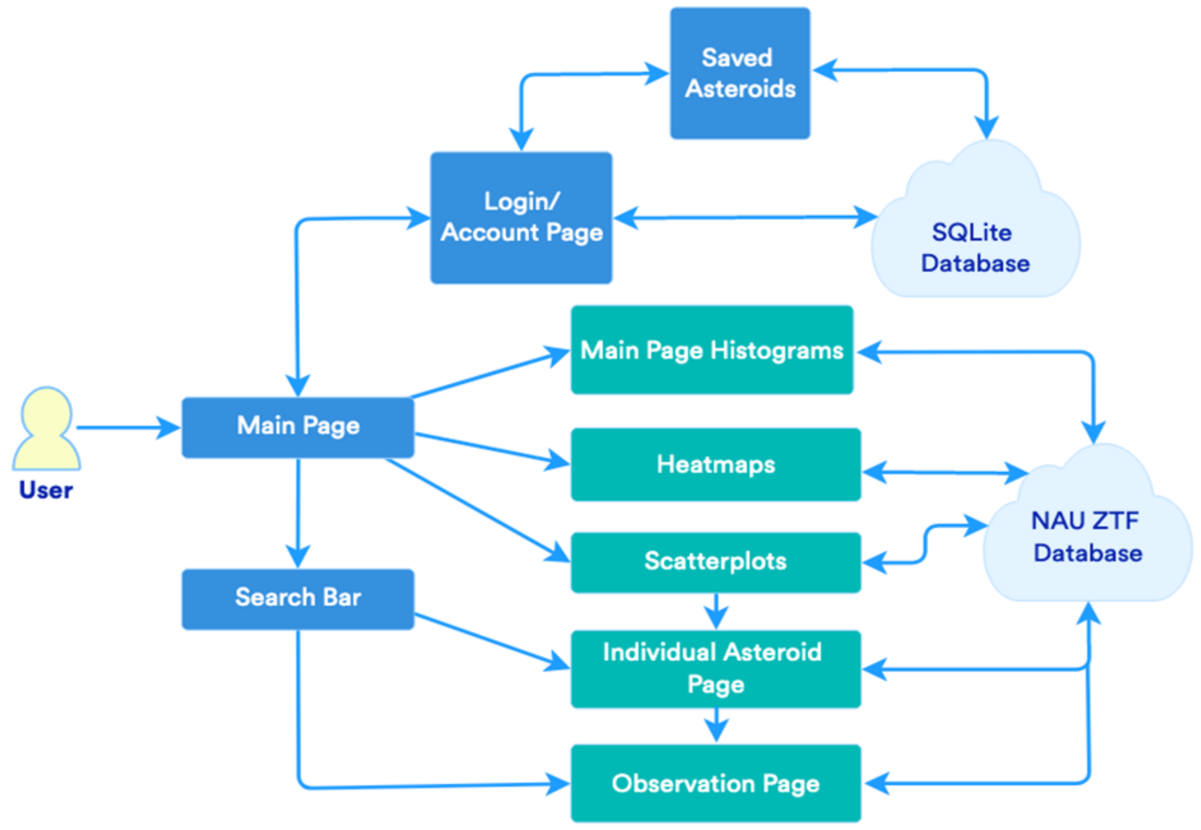


Figure 4.1 - Software Architecture Overview

When the user accesses the web application, they will land on the homepage where static images of histograms are presented. These histograms provide high level views of the ZTF database in that scripts were created to handle massive amounts of data. These histograms can be seen below in **Figure 4.2**.

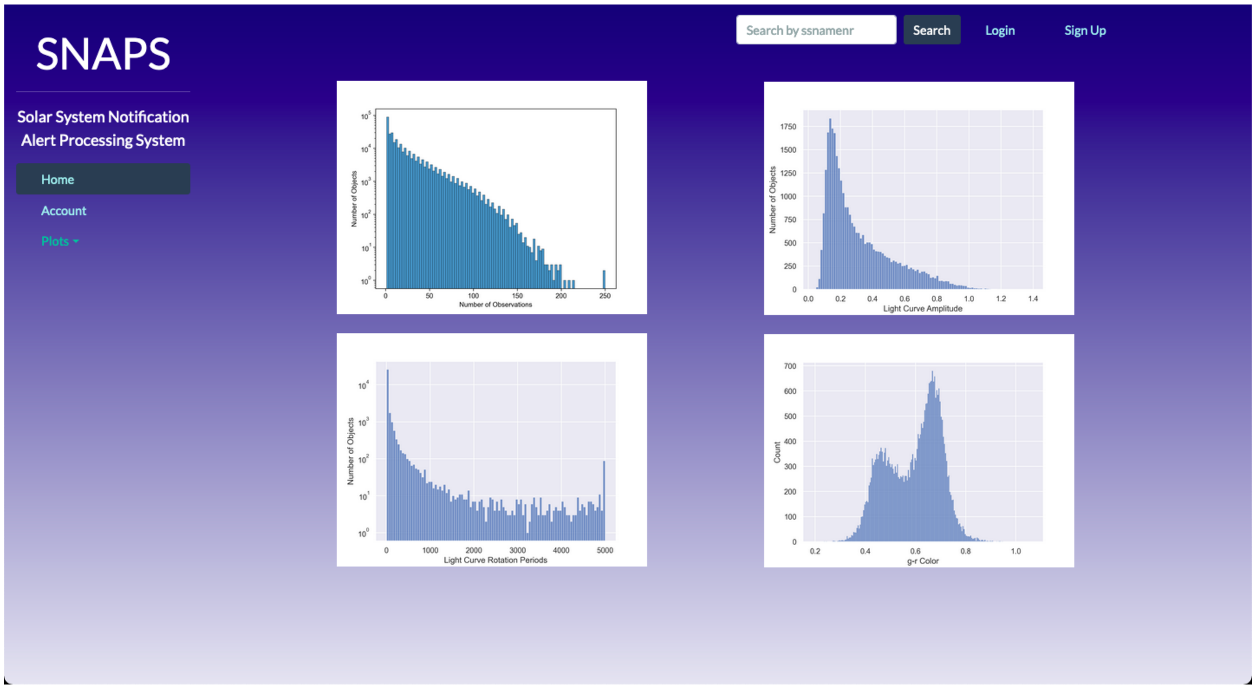
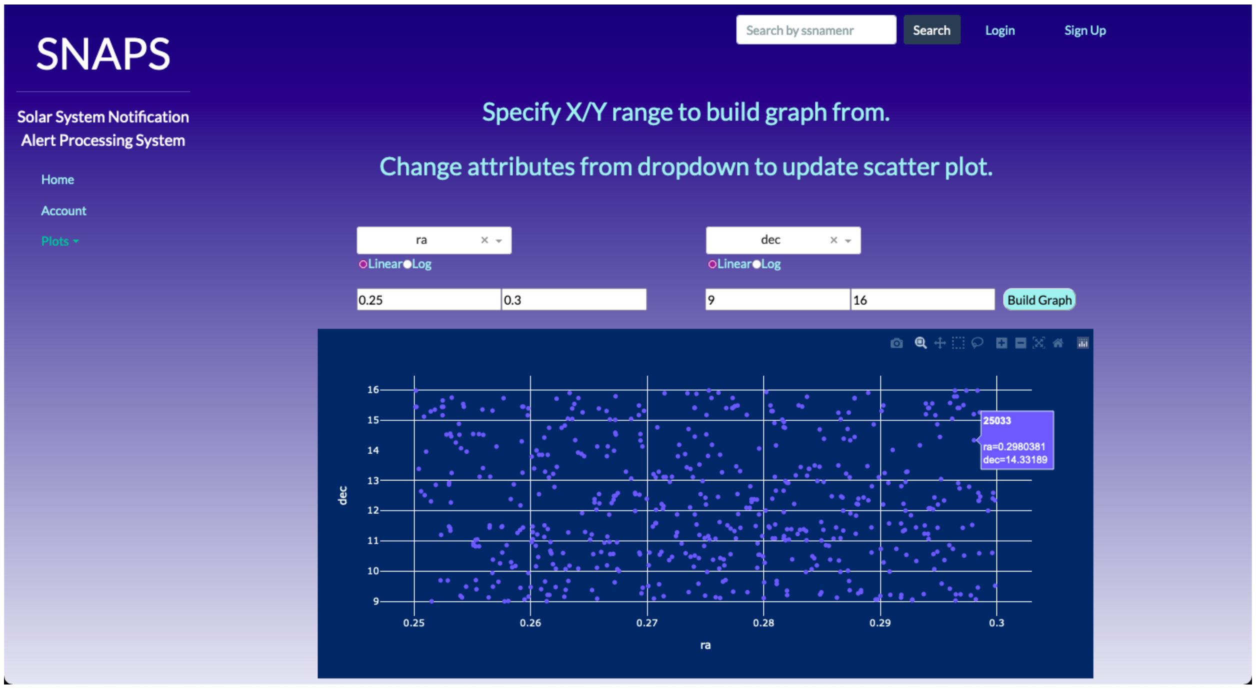


Figure 4.2 - Application Home Page

The NAU ZTF database holds all of the information of all asteroids, such as their observations (one asteroid viewed at a given time). From the main page a user will be able to navigate to plot pages where heatmaps or scatter plots can be built. On both the scatter plot and heat map pages, a user will have to input what X and Y bounds they want to build a graph from. Dropdowns are also in place that contain every attribute pertaining to the asteroids. While the dropdowns default to right ascension (ra) as the X axis and declination (dec) as the Y axis, a user can select a different combination to use in their plot. Once satisfied with the inputs, a user can hit the 'Build Graph' button that triggers the query and graph building. After a short wait, a user will be presented with their newly created plot. An example of this scenario can be seen below in **Figure 4.3**.



Specify X/Y range to build graph from.

Change attributes from dropdown to update scatter plot.

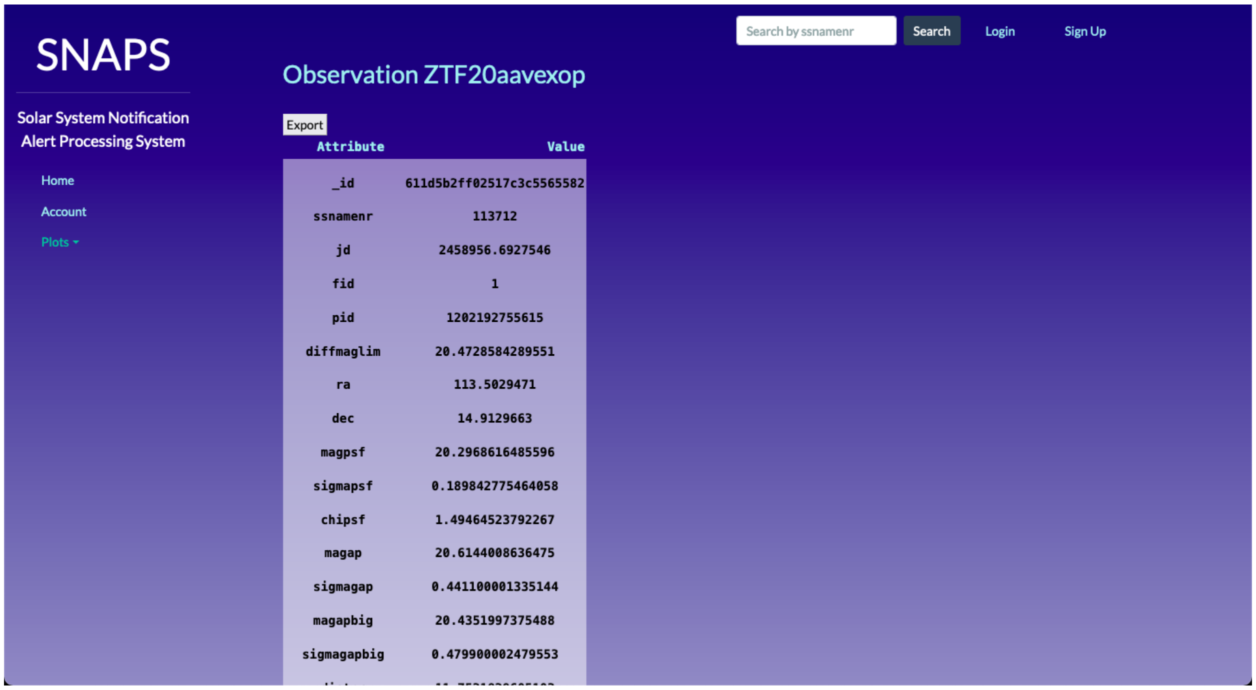
ra — Linear [✓ Log] — 0.25 — 0.3

dec — Linear [✓ Log] — 9 — 16

25033
ra=0.2980381
dec=14.33189

Figure 4.3 - Main Asteroid Scatter Plot

Interaction among both the scatter plots and heat maps include the ability to zoom in/out and pan across the plot. However, far more interaction exists in the scatter plots. Each point within the scatter plots is clickable in that a link will be generated to a new page. Now to loop back to the initial scatter plot discussed above. Once built, a user will be presented with their generated plot where each point within the plot represents an individual asteroid. Once a user clicks an asteroid point, said link will appear below the plot that will bring a user to a new page dedicated to that selected asteroid. Once on this newly created page, a user will be presented with another scatter plot but this time each point represents one observation of that asteroid. The same dropdowns are present on this page as well if a user wanted to measure observations over different attributes. Even further interaction exists on this page in that each observation point within the plot is clickable. The same functionality of a link populating below the graph is in place for this page as well. Once clicked, this link will take a user to a new page dedicated to that selected observation. It is here that a datatable is built that contains every single attribute-value pairing for that observation. This datatable can be exported to a CSV file for later reference. This observation page can be seen below in **Figure 4.4**.



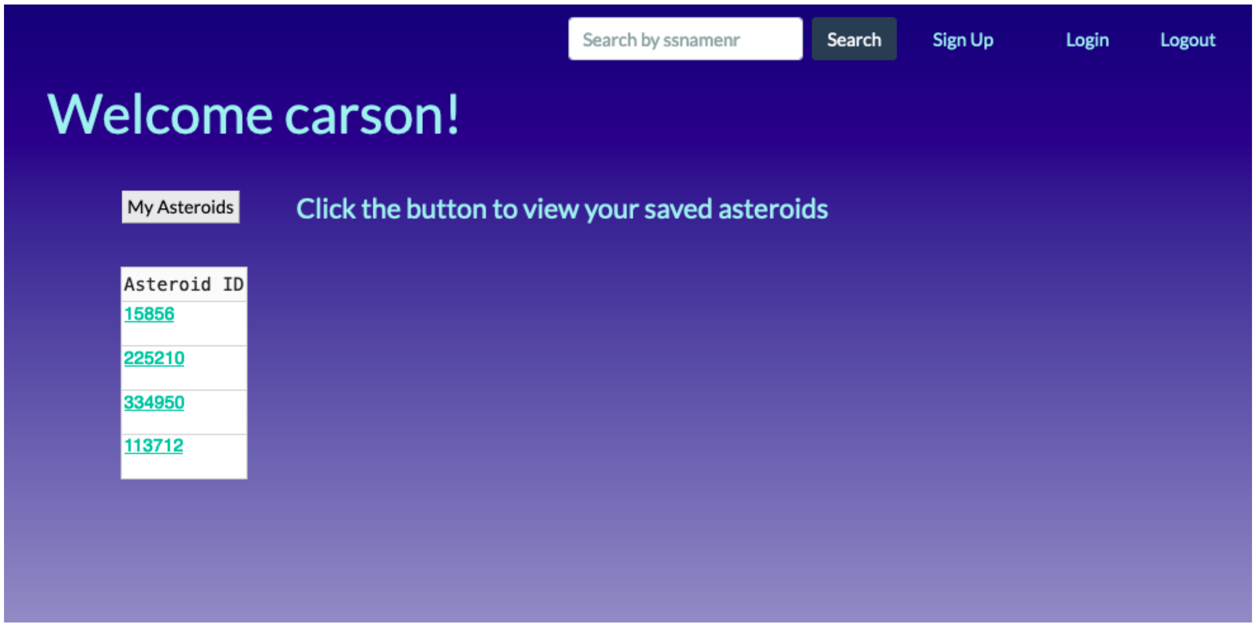
Observation ZTF20aavexop

Attribute	Value
_id	611d5b2ff02517c3c5565582
ssnamenr	113712
jd	2458956.6927546
fid	1
pid	1202192755615
diffmaglim	20.4728584289551
ra	113.5029471
dec	14.9129663
magpsf	20.2968616485596
sigmapsf	0.189842775464058
chipsf	1.49464523792267
magap	20.6144008636475
sigmagap	0.441100001335144
magapbig	20.4351997375488
sigmagapbig	0.479900002479553

Figure 4.4 - Observation Page

All heat maps and scatter plots also have the ability to be exported but to PNG files instead. This is thanks to plotly's strong functionalities. A search bar is also in place to provide quick and convenient access to asteroids if a user already knows the ID of an asteroid that they want to see. To sum up, heat maps and (especially) scatter plots give a user the ability to interact and explore sets of asteroids from the ZTF database.

Another major component of the application is the account functionality. A user of the application is able to sign up for an account and login/logout of their account. The main idea behind this account functionality is to provide a user the ability to save asteroids that they come across. This way, a user does not have to write down or remember IDs of asteroids that they will want to revisit. A simple save button is in place on each individual asteroid page. If a user is logged in, they will be able to save whatever asteroid they come across and have it stored on their account. To reference back to these saved asteroids, a user can click the direct links that reside in a datatable on the account page. The account page can be seen in an example below in **Figure 4.5**.



Asteroid ID
15856
225210
334950
113712

Figure 4.5 - Account Page

Throughout development, the team and the clients often discussed progress and tasks to complete regarding the application. At some points throughout the capstone, there were minor shifts in what was to be implemented for the project. An example of this was the disregarding of further interactive heatmaps. There was a spur of an idea to allow a user to select a portion of a heat map that would result in a scatter plot of that region but this was never implemented. On the flip side, the team provided further interactivity among the scatter plots than what was initially planned. Now that all the major modules, different functionalities, and navigable routes for the created application have been explained, the team can now cover what testing measures were taken during the tail end of the development process.

5.0 Testing

Towards the end of the development phase, the team began the process of testing the product that was built. Three main categories of testing were used in this phase to meet course requirements. They include unit testing, integration testing, and usability testing. Through this testing phase, the team was able to ensure a robust product that meets the needs and requirements of the clients and capstone course.

5.1 Unit Testing

Unit testing is an important part of the software development process that focuses on separating out functionalities rather than testing the application or product as a whole. Before all the parts of a system are tested, individual functionalities are tested to ensure that they properly perform what they are intended to do. This is a key part of the process in software where multiple modules interact with each other. Below, the team covers the plan for all unit testing of the astronomical GUI application.

Throughout the development of the team's application, thorough unit testing was done on a rolling basis as the different modules' functionalities would not properly work on their own without testing various inputs for expected outputs. This testing was done during development because without ensuring that the individual functionalities work on their own before integrating to the larger product, such as having a user login to their account, further testing procedures as described in the rest of this document would be very difficult and time consuming. Since many of the systems in the team's application rely on external packages, such as flask_login, it was important to fully understand the specific functions separate from the entire application. If implemented and tested within the entire application, development would become difficult and messy given how much code the team has written. Specifically, the team has conducted manual unit testing rather than using an automated process that relies on external software packages and libraries. This is because the team has a thorough understanding of the application needs, which are small relative to a larger software system, and the tests to be conducted require known input and output cases. Thus, there will be no mention of any specific testing software packages or libraries in this document. Below in **Figure 5.1** is a table depicting the number of unit tests performed on each piece of functionality that was separated out during the team's development phase of the dashboard. Following this table is a description of the specific functionalities that the team performed unit testing on.

Functionality	**Number of Tests**	**Type of Input**
Create Account	*5*	*Username, Password, eMail*
Login to Account	*3*	*Username, Password*
Logout of Account	*1*	*Username*
Save an Asteroid	*2*	*Username, SSNAMENR (Attribute Value)*
Search Bar	*3*	*Search Bar Input Box*

Figure 5.1.1 - Unit Testing: Function, Number of Tests, and Type of Input

During the initial development starting at the beginning of the Spring 2022 semester, the team implemented various functionalities that, when all integrated together, would create an account system. This account system allows a user to create an account, sign into an account, and logout of an account, however this is not required for a user when visiting the dashboard. While it may seem that all of the functions just mentioned live under the same roof, the specifics of each differ in that various inputs and expected outputs are necessary for a properly working system. This section will start by looking into the specifics of creating an account. In order to create an account, a user must specify three parameters: a unique username, a password, and a unique email address. This set of partitioned inputs have been implemented into functionalities to ensure the uniqueness of all these parameters, thus making testing each an easy task. During development, unit tests of these inputs have been conducted and verfired for robustness in separate files that only involve the use of the flask_login and SQLite libraries. For creating an account, unit tests verified that the username does not already exist in the database, password and confirm password fields matched, email address is a valid address, e.g. input has a '@' symbol and a domain name, email address does not already exist in the database, and that a username does not already exist under the inputted email. All of these checks have the ability to trigger different outputs including error messages and success. So, it was easy to conduct unit tests for each combination of inputs and confirm that the application writes the newly created user into the SQLite database. Similar types of tests have also been applied to login functionality as they both share the same inputs. Input parameters for login are partitioned out to username and password. So, unit tests have been conducted to ensure both fields have been filled out, that the username exists in the database, and that the password and username input pairing is a valid entry pair in the database. Logout functionality is relatively simple in that it only takes the username of the currently logged-in user and executes the logout_user function from the flask_login package. The boundaries in this case only include the username of a logged-in user. The unit test for this functionality is just ensuring that the logout_user function gets called followed by a re-route to the home page.

Saving an asteroid is another feature of the application that is applicable for unit testing. This is a specific functionality that can be tested given that a user is logged in, thus it is a gray area between unit and integration testing. Required input for saving an asteroid includes the username of a logged-in user and the ssnamenr of the selected asteroid, which will be used as a unique identifier when stored in the database. For conducting unit tests on this functionality, the team ensured that the selected asteroid does not already exist under the user's account in which an error will be thrown and no save will occur. Thus the boundary values for this test are that a user is logged in and that the asteroid is not already saved under their account.

The search bar is a key feature of the application in that it provides the user the ability to search for an asteroid by its ssnamenr, the unique identifier for each asteroid in the database. The search bar takes in typed input, so the equivalence partition can be broken down into whatever the user types in, on the ends of incorrect input like a string or list of strings to a correct input of an integer. From these partitions, boundaries for valid and invalid input can be established as integers that represent ssnamenrs in the range of asteroids that exist in the database. There is really only one pair for right and wrong for this search bar: either the input is an integer that corresponds to an asteroid in the database or is of an improper type that yields an incorrect search. Any input that does not yield results from the database query by ssnamenr will default to an error message that tells the user that the asteroid they are searching for does not exist in the database and to attempt another search. The team conducted unit tests on the search bar during development by providing these types of inputs. Whether it be a random string, a negative integer, or an integer in the range of ssnamenrs that exist, the team verified that the search bar properly does what it is intended to do.

5.2 Integration Testing

As mentioned before, there are many types of testing in software development. While unit testing separates out functionalities at the low level to ensure a function does what it is supposed to do, testing the entire application where many modules interact with each other is crucial for turning over a working product to the team's clients. Integration tests ensure that all the different modules of a software system are properly working while integrated together. Without performing integration testing, a system can easily be broken in various ways as there is a lot of interaction between different functionalities.

The application that the team is building is comprised of many modules that integrate together to create the working dashboard. Many of the modules rely on each other in that different forms of output act as input to other functionalities. Similar to unit testing, the team conducted integration testing throughout the stages of development. Since there is not too much going on within the team's application in terms of the number of modules, integration testing was relatively easy, although the most important in the testing phases for the team's dashboard. The team's plan for integration testing is to manually walk through the entire application. Taking all the possible routes into account will ensure that the modules are integrated properly and neatly. For example, a user can take the route of getting to an observation of an asteroid by clicking through two scatter plots. Below in **Figure 5.2.1** is a high-level diagram of the application being built. Following this is a description of the different interactions taking place throughout the application.

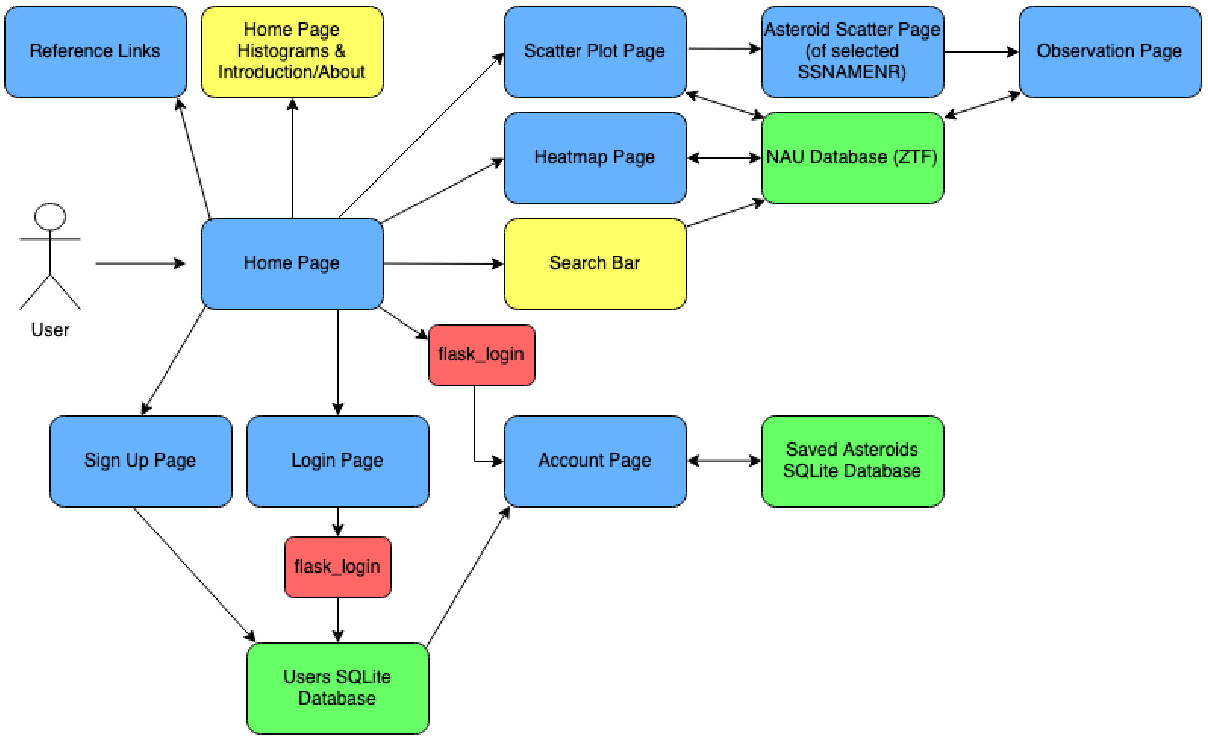


Figure 5.2.1 - Application Flow Overview

When a user first visits the team's site, they will land on the homepage where static histograms and an introduction of the application are displayed. A top and side navigation bar are a part of all pages, so it is expected that these will also render properly on the home page. When the application is run, both locally and on the production server at Northern Arizona University (NAU), the home page is properly loaded with all elements (home page information and navigation bars). The second integration test will cover the account functionality where a user can create an account, login, and logout. This account functionality as a whole is a key piece to the application that allows users to save asteroid data for future reference. The functionalities just described do not necessarily interact directly with each other, but do follow a common path that must be tested for proper working. During the development of the account system, thorough testing was conducted on a rolling basis to ensure that the flow of creating an account, logging in, and logging out was intuitive and properly working. The account system makes use of two separate SQLite files; one to store created users and another to store saved asteroids referenced by an account username. Integration testing for the storing of this data centers around ensuring that many conditional checks are properly tested. These cases have been thoroughly tested to ensure proper function and include:

1) Email address for an account should be uniquely stored
2) Username for an account should be uniquely stored
3) Only one username per email address
4) At account creation, password and password confirmation should match

Another major piece of the team's application is the ability to navigate to a specific asteroid. This can be accomplished in two ways: via the search bar or by selecting a point out of the main scatter plot. Either way, the user will be brought to a scatter plot of observations for the desired asteroid. Similar to the majority of the application, this functionality and integration was verified for robustness during development. For integration testing, the team ensured that both of the paths are still properly working. The search bar will return error messages such as "asteroid not found" when invalid input is given via the search bar. This is a simplistic implementation and is tested by typing invalid inputs such as negative numbers or words, instead of an integer that is in the valid range of the database's ssnamenrs.

Branching off of the asteroid scatter page just covered, a user will then be able to interact with the scatter points that represent all the observations for the selected asteroid. When clicked, a user will be able to click a generated link that navigates to an observation page. This observation page displays a table of all attributes for an observation, such as magnitude and julian date. To ensure proper integration of this linking, the asteroid path mentioned above will be used again. Integration testing of this module that provides interactivity among scatter plots has been conducted through numerous run-throughs of the application.

While the use of specific testing frameworks or tools is lackluster in this document and specific section, the team can confidently say that the application modules are integrated well. Throughout development, thorough testing has been conducted as each new module and specific functionality were introduced. Without properly testing the modules as a whole, e.g creating an account then logging in and out, the team would not have completed development to the specified requirements.

5.3 Usability Testing

The end-users of the site are people that would be interested in finding interesting data about asteroids in the solar system. While the site features a more research-targeted design, any user who might be interested in astronomy should be targeted. The team does not believe that a user needs to be well versed in using technology, so only a basic understanding of how to use a computer and a browser to navigate a site should be expected of the users. This will mean that the team will have a wide base of potential users to draw from.

The team wants a user to:

1) Create an account
2) Find an interesting asteroid, either through search or by using a plot
3) Save the asteroid

4) Download either a plot or csv of the data for future use

From the home page, the user will be tasked to create an account in order to use the site's save asteroid function. Once the user creates an account, the ability to save an asteroid page is now available. The user will then be asked to look through the sidebar and select the scatter plot link or search for an asteroid using the search bar. The scatter plot link will lead the user to a scatter plot page with drop down menus so that the user can select the attributes they want to see. If what the user sees is interesting, the user can save the plot to their account for future reference. The user can also download the plot or the datatable using the download function that the team has implemented. If the user just wants to observe a certain point, the user can click on a data point and a link will pop up leading them to an individual asteroid page with the data pertaining to the asteroid they selected. This data can also be downloaded for future use. This is one route the user can take. If the user decides to use the sidebar after creating an account to search for asteroids, the user will be asked to enter a ssnamenr number with the length of 1 to 6 numbers or a preselected object from the database. If the user decides to enter a number, then the user will be routed to an asteroid list page that contains every asteroid that has the same ssnamenr that the user imputed. The user will then select an asteroid from the list which will then route them to the individual asteroid page that contains a data table pertaining to the asteroid the user selected. However, if the user selects to use a predetermined ssnamenr object, the user will be directly routed to the individual asteroid page.

6.0 Project Timeline

Below in **Figure 6.1** and **Figure 6.2** are gantt charts that depict the team's progress throughout the entirety of the capstone experience. This progress is measured over both academic semesters, spanning from August 2021 to May 2022.

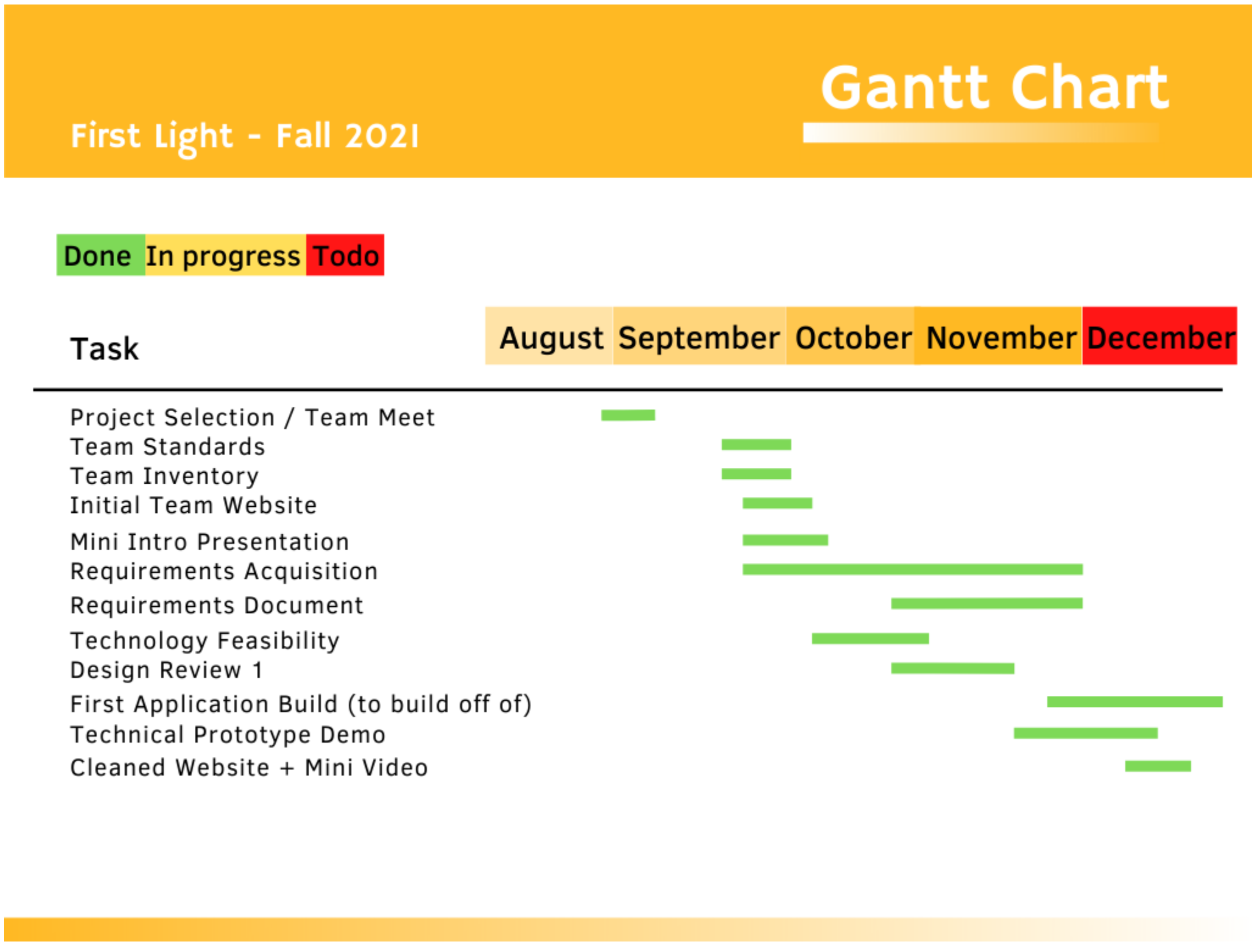


Figure 6.1 Gantt Chart Fall 2021

For the Fall 2021 semester, there were twelve major milestones. Each milestone took the expected amount of time to complete. The only milestone that took longer to complete than the others was the Requirements Acquisition as this went on a rolling basis throughout the semester. Through multiple meetings with the clients over the span of about two and a half months, the team acquired the requirements necessary to complete this project. The team also began to think about the technological feasibility of the project and which technologies could be used to successfully build the desired product. Design Review 1 was presented to update the other teams and mentors on how the project had been going thus far. The First Application Build was then completed to be used at the Technical Prototype Demo, which was then built off of for the Alpha

Demo the next semester (Spring 2022). Upon the conclusion of the semester, the team website was cleaned up and updated. A mini video was also created as a sort of sales pitch to anyone who might be interested in learning more about the project.

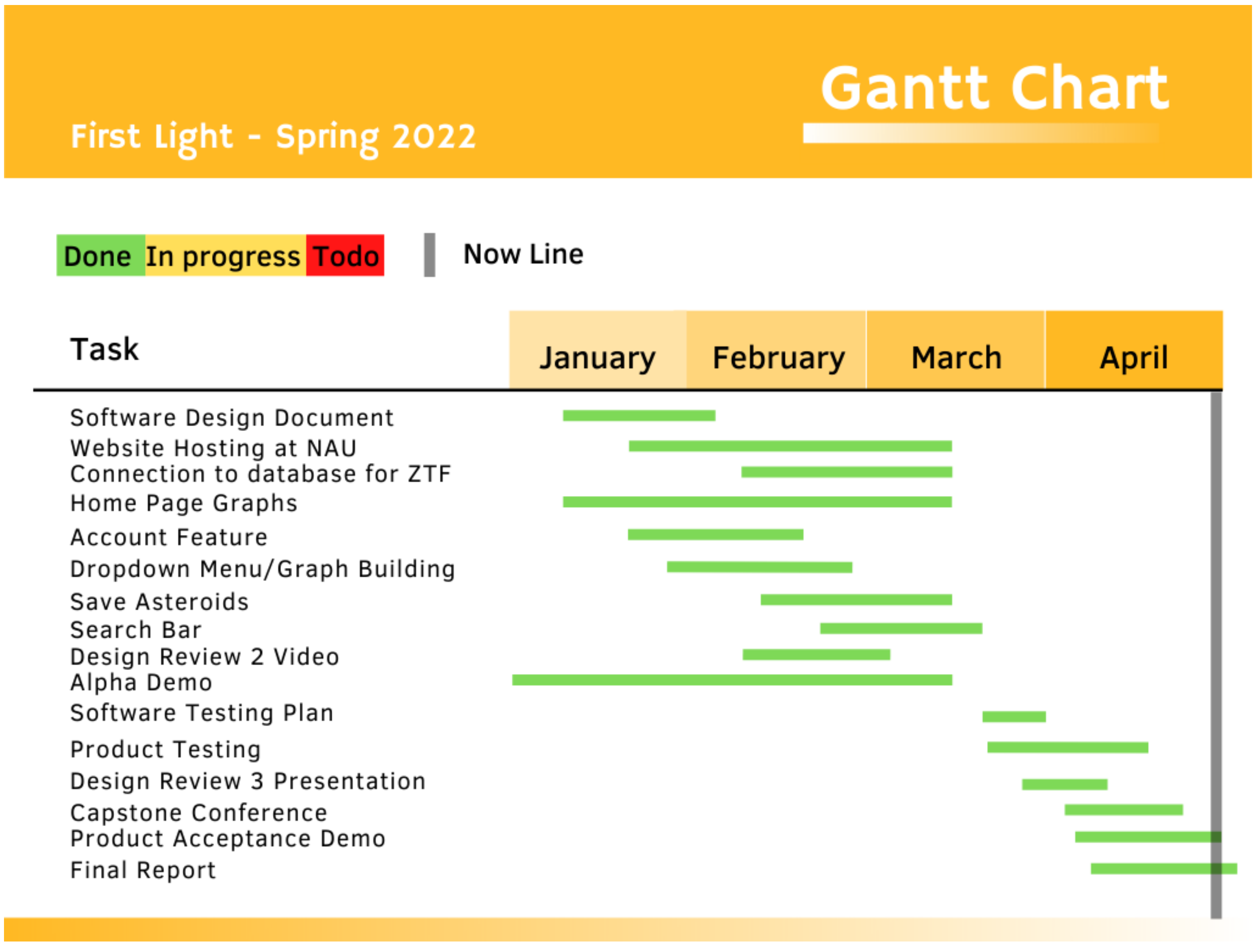


Figure 6.2 Gantt Chart Spring 2022

Figure 6.2, as shown above, is the gantt chart for the Spring 2022 semester where the team had fifteen major milestones; starting with the Software Design Document at the beginning of the semester. This document was where the team outlined the development of the product and went into detail about the overview of the implementation, overview of the architecture, module and interface descriptions, and the implementation plan. After the Software Design Document, there were seven major implementation modules to work on. The account feature, dropdown menu / graph building, save asteroids, and search bar modules each took the expected amount of time for the team to implement.

Three of the milestones took longer than expected. First, hosting the web application at NAU took far longer than expected. This is because the team had to containerize the project with Podman and NAU ITS had no information on containerizing and supporting a Python-based application, only an HTML and PHP application. There was little information on the internet either about why the container was not fully working. After weeks of emails, the team set up a meeting with Clint Baker from NAU ITS. The meeting lasted about 3 hours and it took four different people running different container configurations to figure out fixes to the issues that the team was having. Secondly, connecting to the ZTF database took longer than the team anticipated due to issues getting the right credentials and connecting using Python and not through the MongoDB application. Lastly, for the home page graphs, it was just a matter of knowing which visualizations the clients wanted, so that the team could generate them and embed them within the team's application's home page. Design Review 2 was used to get feedback from other teams on how the project information was presented and if the product was coming along smoothly up until that point, according to the team's sales pitch. The Alpha Demo was presented to the team's faculty mentor to show that all or most of the functionality requirements were complete. Product testing and the Software Testing Plan were then written and carried out to ensure any errors found within the product were fixed. Testing was also used to find out if the application was user friendly and simple to use, like was outlined in the Software Design Document. Design Review 3 was used to present the team's progress up until that point and to show how the team used Product Testing and the Software Testing Plan to fix any errors or usability issues.

Most recently completed was the Capstone Conference where the final product was showcased and presented to the general public. A poster was created to highlight the main focuses of the product, like motivation, architecture, a small walkthrough of the functionality, and challenges that were faced. The last two major milestones are the Product Acceptance Demo and the Final Report, which were completed at the end of the capstone experience.

7.0 Future Work

The team is confident that the product developed is well suited for another development team to branch off of. The current dashboard works well and meets the original requirements however, like most software projects, there is room for improvement. To start, one of the key future work goals the team envisions is giving a user the ability to share data between other colleagues within the dashboard. While the current dashboard supports downloading data to PNG and CSV formats, there is not a way to share this information within the dashboard. This would cut out an external step of emailing the downloaded data to whomever the user chooses. Another key development step for future work has to do with the speed of the dashboard, specifically the queries to the ZTF database. At the moment, the team is experiencing less than desirable results for larger queries. Although steps were taken during development to curb this, the team sees this as a key focus for future work. Speed is a key focus for this project, thus the team envisions future work to speed up the database queries in order to have the graphs be presented to the user in less time. The team also took the time to document the entirety of the project so that each line in each function is thoroughly described. This is to help whoever picks up the code next so they are not confused as to what is happening at any point in the code. Another piece of future work that the team sees as fit is the addition of a datatable or something similar that contains the numerical ranges of all the asteroid attributes. At the moment, a user has to know the valid numerical bounds of their selected asteroid attributes in order to build a working plot. With more information about the attributes, a user might have a better understanding of what they are when interacting with the plot pages.

8.0 Conclusion

The visualization of astronomical data is a crucial component to analyzing asteroids, stars, and other objects. Without visualization tools for analysis, researchers need to comb through thousands of data points using manual methods, making the ability to see patterns and develop trends complicated. This project is intended to help researchers visualize and analyze the large amounts of asteroid data hosted on NAU's servers. The current methodology researchers use involves time intensive analysis that focuses on individual data points rather than larger trends and outliers. The solution of a web application developed during the capstone experience that is able to pull large amounts of data and visualize it in graphs, heat maps, and other visualizations is to help researchers understand the data they are looking at. The web application gives insight to outliers, trends, and distributions that will provide a better understanding of what the data means. This final report highlights the motivations, design process, implementation details, testing procedures, and roadmap for team First Light's capstone experience. The team enjoyed developing the asteroid data dashboard and gained many valuable insights and skills from development to client communication. The team is also excited to see where this project heads and what kind of future developments will take place.

9.0 Glossary

Although most requirements for running this project are described in the following appendix, there is a key piece of information that must be mentioned here. Throughout this document, the ZTF database was mentioned quite often. In order for this project to serve any use at all, a connection to this database must be established. The only hurdle is that this database is hosted and accessed behind proper authentication. Whenever this project is run to serve the average user, a connection to this database must be made via the client-provided connection string.

Appendix A: Development Environment and Toolchain

Hardware

The platforms that the team developed on were Linux, Mac, and Windows operating systems. For the Mac platforms that the team developed on the following: a Macbook Pro (13-inch, 2018) with a 2.7 GHz Quad-Core Intel Core i7 processor and 16GB of RAM, a MacBook Air (13-inch, 2017) with a 1.8 GHz Dual-Core Intel Core i5 processor with 8GB of RAM, and finally a Macbook Air (13-inch, 2020) with an M1 processor and 8GB of RAM. There were two Windows platforms developed on: one of them being a Windows 10 build 19044 with a Ryzen 5 2600 processor and 32GB of RAM, and the other being a Windows 10 build 19044 with a 2.56 GHz Intel Xeon processor and 32GB of RAM. The linux platform used to develop this project is a Red Hat Enterprise Server 7.9 release, which is the NAU linux server. The only hardware requirements for this project is a machine that can ssh into NAU's linux servers to configure and maintain the Podman container from which this project is hosted on.

Toolchain

The development environment(s)/editor(s) that were useful for this project include Sublime Text, Atom, and VS Code. All are text editors that have easy to use GUIs and powerful preloaded tools in place like a spell checker and a syntax checker for different programming languages, like Python.

For the backend databases, this project used MongoDB and SQLite. MongoDB is the ZTF database that the clients use to store all of the data that is received. There are a few tables from which this data is put into and organized, but this project only uses the *ZTF* and *Asteroids* tables to gather and visualize the data. The *ZTF* table is used for the different visualizations and individual data, while the *Asteroids* table is used to find out the total count of observations for a certain SSNAMENR; which is then displayed along with the asteroid scatter plot that is linked from the main scatter plot page. Two separate SQLite database files were used for storing created user accounts and storing asteroids associated with each user. The reason the decision was chosen to use two separate SQLite database files is because it was easier during development to have both SQLite databases up to look at then switching between them when unit testing and integration testing. The team and the clients also agreed to not append to the Mongo database where all the asteroid data is stored so as to not create clutter or confusion.

Python was the programming language of choice for this project due to the fact that the clients are knowledgeable with its syntax and because of the powerful libraries available. This project utilized many different Python libraries to aid in development, the first and most important being Dash, a framework for building data visualization interfaces. There are three main technologies Dash uses for its core: Flask, for the web server functionality. React.js, which is used by Dash for the rendering of the user interface of the web page; and Plotly.js for generating the plots used in the application. Version 2.0.0 was used. It was needed to develop the web-based dashboard from which this project is built off of. It was the main library that all of the product development was branched off from.

Dash Bootstrap Components is another framework that makes it easier to build responsive web application layouts, specifically, Version 1.0.2 was used. This package was used to build the sidebar, where the tabs for the Home page, Scatter Plot page, Heat Map page, and Account page are located. It was also used to build the top navigation bar where the Search Bar, Sign Up, Login, and Logout buttons are located. The inputs and respective labels for the Sign Up and Login pages were also built using this library. The last part of the project that Dash Bootstrap Components were used for was the alerts for successful account creation, saving of an asteroid, and the various errors associated with the Sign Up and Login pages; like an account already registered with the same email and username, if the email entered is invalid, if there was an empty field, and if the username or password entered is incorrect. It is also used to let the user know when an asteroid is saved to their account, or if the asteroid has already been saved before.

Dash Core Components, specifically version 2.0.0, was the next library used. This package provides the core React.js components. This was used to display the plots under dcc.Graph and the dropdown menus for the X and Y axis attributes. It was also used for the Linear and Log buttons that, when clicked, switch the plots from linear to logarithmic. The X and Y bounds for the scatter plots also utilized Dash Core Components. When there was an instance of a function returning a specific location in the web application; like for example when a user is logged in, the user is rerouted to the account page; or when a user is logged out, it reroutes to the home page. This is done through the dcc.Location function, which reroutes the user to which pathname is specified inside the function.

Dash HTML Components version 2.0.0 was the next library to be used under the Dash package. This library makes it easy to create or insert HTML elements into the existing Dash application. For example, the team can use html.H1() to create a heading 1 element and have it be displayed in the application, which was used for the Solar System Notification Processing System title. Another use was to insert datatables into the application when it was not already created, an example of this is when clicking on an individual asteroid observation it links to a page with a datatable of all of that observation's attributes and values.

Flask, Flask_SQLAlchemy, and flask-login were all used to interface with the Flask API for logging in and interacting with the SQLite database. These libraries were used as opposed to the Dash Authentication as the Dash Authentication libraries were locked behind the licensed enterprise edition. Flask Version 2.0.2, Flask_SQLAlchemy Version 2.5.1, and flask-login Version 0.5.0 were used.

Just as Dash is the backbone of the data visualization interface for this project, Plotly is the generator for the data visualizations. Plotly express is used to create the scatter plots, and the heatmaps. Plotly is also the default dcc.Graph() core component function to help render and display graphs or plots with ease. Any Plotly version 5.6.0 or above is compatible with this project. Lenspy's DynamicPlot(), from Lenspy version 1.1.0, is used to set a maximum size on the scatter plots. This is to prevent a user's system from being bogged down with a massive graph. It should be set to 100,000 or more points to still provide a user with access to a large query.

Pandas acts as the glue between the database querying libraries and the graphing libraries. When a query to the ZTF database is made, it is passed into a pymongo object, which is passed into a pandas dataframe. This dataframe provides an interface to that data that Plotly can read and build a graph from. For this project, use pandas 1.4.0 or above is compatible with this project.

Numpy is used to cleanly display the user data when a user is viewing their saved asteroids. Numpy is also used inside the pandas library to speed up pandas operations. Numpy Version 1.19.2 was used.

Pymongo is used for python to interface and query with the MongoDB database. The pymongo queries are then passed to pandas for further operations. Pymongo Version 4.0.2 was used.

Werkzeug.security is used to manage password hashes, such as checking if the hash is valid and generating a hash for the chosen password. Werkzeug Version 2.0.3 was used.

Setup

In order to run the team's web application, versus simply visiting the live website, a few steps must be taken to ensure proper functionality. To start, the Python programming language must be installed on the user's computer due to the fact that it is a Python application. Access to a terminal/command line is required to run the application. This could be via an integrated development environment (IDE) or via a normal terminal. There is only one Python file needed to run this project, named 'app.py'. Another setup requirement to run this application is that the histogram PNG files are present in the same directory as the main python file. The last file requirement is that the user has the 'assets' folder in the same directory as the main python file. This folder doesn't contain any major requirements to run the project, just a few logos used in the footer and browser tab.

Moving onto the more important requirements to run the application. The team utilized many packages during development that are all required for the application to properly work. The team will not go into detail about each package and version in this section as they were mentioned previously in the toolchain section. Lastly, a connection to the ZTF database must be made in order to get all the asteroid data loaded into the application. The clients provided a connection string that the team wishes not to share publicly, but is required for the application to run properly. When a developer is ready to run the application, all they need to do is run the command 'python3 app.py' in a terminal. This will result in the application being run on localhost.

In order to work on the application live on the internet, a developer needs to have Podman installed on their machine. Podman is a containerization software that takes all necessary components of a project and wraps them into a container that can easily be shared and run on different machines. Due to the fact that this project requires hosting on NAU servers, the Podman container lives within the NAU linux server, specifically on two instances, one for development (webapps-dev.ac.nau.edu) and one for production (webapps.ac.nau.edu). These instances can only be accessed with valid credentials and authentication. The instances both run on port 9010 through the username 'ceias_snaps'. There are two URLs that can be used for this project, one for development and one for production. The URLs to be used from these containers are as follows: https://rc.nau.edu/snaps/ and https://rc.nau.edu/snaps-dev/. In order for these URLs to work, a container needs to be created and running using the following commands: 'podman build' and 'podman run'. The Containerfile will also need to be configured to set up the container as follows:

```
FROM python:3
ADD whatever_the_python_file_is_named.py /
ADD constring.py /
ADD assets_folder /
ADD static_histogram_1.png /
ADD static_histogram_2.png /
ADD static_histogram_3.png /
ADD static_histogram_4.png /
COPY requirements.txt ./requirements.txt
RUN pip install -r requirements.txt
CMD [ "python3", "whatever_the_python_file_is_named.py" ]
```

The requirements.txt file will need to include the following libraries and packages:

```
dash                        >= 2.0.0
dash_bootstrap_components    == 1.0.2
dash_core_components        == 2.0.0
dash_html_components        == 2.0.0
dash_table                  == 5.0.0
plotly                      >= 5.6.0
pandas                     >= 1.4.0
Flask                       == 2.0.2
Flask_SQLAlchemy            >= 2.5.1
pandas                      == 1.4.0
numpy                      >= 1.19.2
lenspy                     == 1.1.0
```

pymongo	== 4.0.2
Werkzeug	== 2.0.3
SQLAlchemy	== 1.4.32
flask-login	== 0.5.0

Production Cycle

For a developer new to this project to pick up where this team left off, it is important they know how to properly make changes and have them reflect on the live production, or development server. The team encourages new developers to make and test changes locally on their machines before attempting to run the application live on a Podman container. Once satisfied with the changes, a developer can move the changed code into a Podman container on the NAU linux server. When the new code is inside a container, a developer can build and run the container to see the changes online.