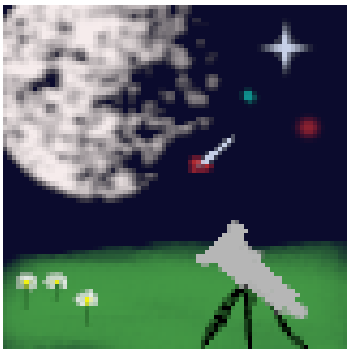

Team First Light

Software Testing Plan

Version 1.0

1 April 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 **3**

2.0 Unit Testing **4**

- **Figure 2.1** **5**

3.0 Integration Testing **8**

- **Figure 3.1** **9**

4.0 Usability Testing **11**

5.0 Conclusion **13**

1.0 Introduction

With the rise of big data collection and data science, the availability of research using large and complex data sets has become much more common. Astronomy as a field is among the most suited for taking advantage of the move to using big data and data science. An example of this is the Vera C. Rubin Observatory's Legacy Survey of Space and Time telescope, which will produce 20 terabytes of all sky astronomical data each night. However, many publicly available Astronomy websites are poorly designed, or do not contain access to these large datasets. The sponsors have tasked the team with creating a data dashboard to help visualize the data from the Zwicky Transient Facility, and that can be extended to also visualize the data from the Legacy Survey of Space and Time.

Software testing allows developers to know that the product they are developing works properly, allows the developers to find and prevent bugs and verify that all functions work together and do what they are supposed to do. This paper will discuss three different types of software testing used on the SNAPS data dashboard. Unit testing allows developers to isolate a function and see whether it performs as expected. Integration testing allows developers to make sure different functionality works as expected together. Usability testing allows a team to know how a user will use the product and test the product's user-friendliness. These are some of the few that developers can choose when testing their own software.

The team will develop unit tests for 5 functions, with multiple tests in each function to ensure that the major components are displaying what they need to display or rerouting to the appropriate page. For integration testing, the team will test whether all functions work together properly and produce the required output. All functions will be tested to ensure that no components or links are broken, plots are properly and correctly made, and users can sign in or

sign up along with other functionality. For usability testing, a group of users, mainly those who are interested in astronomy, will be using this project to do an assortment of tasks to ensure that the web application is easy to navigate and easy to use. Overall, the main focus of the team's testing program is to develop a user-friendly dashboard, since other astronomy dashboards are very difficult to navigate and use. By using unit tests, bugs can easily be found through the different scenarios the functions will be put through. Integration testing will check if each individual module can communicate with each other without any bugs. Usability tests will then be used to check whether the completed web application is user-friendly by having them complete tasks and saying what the difficulty was in achieving these tasks. In the next sections, the paper will describe the specifics of each type of test.

2.0 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 2.1** 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 2.1 - 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 packages. 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.

3.0 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 3.1** is a high-level diagram of the application being built. Following this is a description of the different interactions taking place throughout the application.

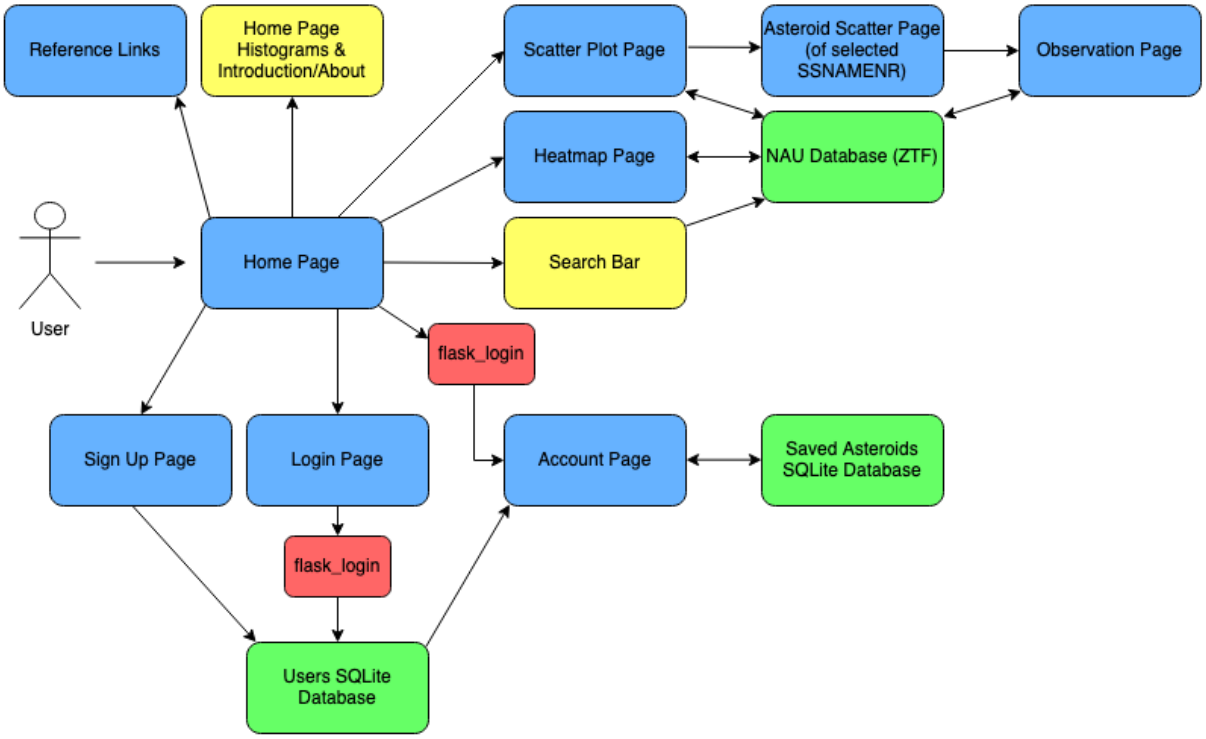

Figure 3.1 - Application Flow Overview

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 be at the current position in development where fine touches and performance improvements are being made.

4.0 Usability Testing

While making sure the various components of the site work well together is important, it is also important that a user is able to navigate and use the site properly. Usability testing is a technique where a group of potential users of the site is asked to use the site in a variety of normal ways, such as creating an account, while members of the team monitor them. This allows the team to find any issues with the usability of the site, such as poorly thought out workflows, and provides an avenue for the team to fix those issues.

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.

Through these steps, the user will get the full experience of the web application and will be able to determine if it is user friendly or not. With these critiques, improvements and further enhancements can be made so that the user has a better experience using the web application. The critiques from the users who are interested in astronomy will be taken into consideration more than normal users since those users have more experience with the subject. The critiques will be written up as the user goes to see where the users struggled or what improvements they recommend. After all testing is complete, the team will review the critiques and see where in the web application the users struggled the most. From these critiques, improvements will be made so that in future deployment, other users will not have to struggle when navigating the web application.

5.0 Conclusion

As a result of the unit testing, integration testing, and usability testing, the team is confident that the implementation of the project demonstrates the necessary functional and non-functional requirements that were stated in the description for this project. The project will be a dashboard capable of supporting visualization and interactivity among cleanly represented astronomical data sets terabytes in size. This will serve as a simple and easy application to use for astronomical observation and research, specifically on asteroids. The problems this project is set to solve include: Visualization of large data sets through plots, charts, plots, etc; Interactivity of data points within visualizations to view characteristics, and needing a simple interface with straightforward web pages that pertain to different modules, i.e. a home page with a high-level summary of all the data vs. an individual asteroid page with specific characteristics. To date, the team has completed all of the requirements for having a minimum viable product, with only parallelizing pulling the data from the database and the construction of the plots as the last focus

being developed by the team. Lately, time has also been spent styling the web application to look appealing to the users to ensure simplicity of the design and navigation. The team is confident and is excited that by May of 2022, the clients, Dr. Gowanlock, Dr. Trilling; along with other researchers, will have a user-friendly web application capable of visualizing vast amounts of astronomical asteroid data.