Software Design Document

February 4, 2021

LOST EXPRES

Sponsor: Joe Llama

Mentor: Volodymyr Saruta

Team Lead: Brooke Caldwell

Architect: Jared Cox

Customer Communicator: Olivia Thoney

Release Manager: Ian McIlrath

Meeting Recorder: Austin Bacon

Table of Contents

1. Introduction 3
2. Implementation Overview 4
3. Architectural Overview 6
4. Module and Interface Descriptions 8
    4.1. Graphing 8
    4.2. Server 10
    4.3. Web Application 12
5. Implementation Plan 14
6. Conclusion 15

1. Introduction

Since the discovery of the first exoplanet in the early 1990s, society has been infatuated with the exploration of outer space with the goal of finding more and more planets outside of Earth's solar system. A vast number of these exoplanets have been discovered, but none have been found to be Earth-like planets orbiting sun-like stars. Assistant astronomer Dr. Joe Llama and his team at Lowell Observatory specialize in the study of exoplanets and they aim to find a planet that is similar to Earth.

Dr. Llama's research has brought him to gather data on the Sun in order to better analyze its radial velocity. When a planet orbits a star, its equal force of gravity causes the star to slightly wobble, allowing for the change to be measured. Our Sun gives a great opportunity for astronomers to examine a star with a significant amount of precision. By gathering data on the Sun, we may more precisely detect the presence of an Earth-like planet around a foreign star. Yale University has developed the EXtreme PREcision Spectrograph (EXPRES), which is a tool to measure such data. Lowell Observatory has partnered with Yale University to use their EXPRES tool on the Lowell Observatory Solar Telescope (LOST) to gather data on the Sun.

Dr. Llama is leading the team in gathering the data, and at a rate of 40GB per day. This high rate is due to the fact that Dr. Llama takes 5-minute exposures with EXPRES, for around 8 hours a day. Now that Dr. Llama and his team have begun gathering and storing data, they need a way to view and analyze it. He has tasked LOST EXPRES with the job of creating a web application for him and his team which will access their database and display graphical data of what they have gathered. The major requirements for this web application are:

- Role-based permission admission that will allow for registered users to access more data, and allow for administrators to add and delete data as well as making it visible to the public, unregistered users.
- Fast database queries to allow for quick and easy navigation of the web application as well as proper graphing of data.
- Plotting graphs of both radial velocity and individual spectrum in the radial velocity to view data from different angles and displaying it.

- Maintaining a user-friendly environment for the public to use because the unregistered users will most make up a large portion of traffic on the web application.
- All of this information will be displayed in a web application that will link off of the Lowell Observatory website.
-

This application will help Dr. Llama and his team with the goal of collecting data and making it accessible to the common person. With this application, the general public will be able to view more information and hopefully gain an interest in the overall goal, finding a new earth-like world.

2. Implementation Overview

In order to benefit both researchers and the general public, The majority of stored data will be open to the public. However, some newer sets of data will only be available to researchers. This is in order to limit the amount of access the public user role has.

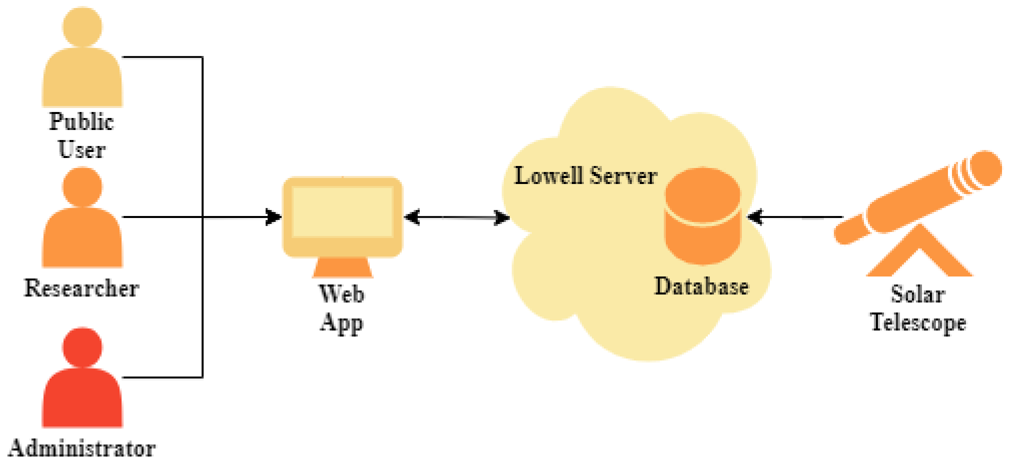


Figure 2.0 - System Diagram

Our broad solution to the client's problem, as seen in Figure 2.0, contains several components. The first is the web application that is the interface for the various users of the site, interacting with the server provided to us by Lowell Observatory. The server will host a local database that stores solar data recorded from the solar telescope and tools to parse the spectrograph's FITS files into an easily stored format. The web interface will consist of:

Key Technologies

In order to implement our solution vision, we will use the tools and libraries listed below.

- Web Interface
  - Bootstrap is a front-end framework for designing user interfaces in a relatively short amount of time. It was chosen because of its compatibility to the
  - JQuery Slim, and Popper.js are used as dependencies for the Bootstrap library.
- Server
  - Python is the language we chose to build the server, because the LOST team would use Jupyter Notebook for manually visualizing the desired graph, using several libraries such as AstroPy.
  - Flask is the Python server that we chose to implement, as it has a large user base, as well as an assortment of server extensions to help implement secure features. These features range from logging in and session tracking,
  - Flask-login keeps track of which users are logged in, and integrates tools to route users accordingly.
  - Flask-mail can send emails to users who have forgotten their passwords, as well as send requests to Administrators when a researcher signs up to the website.
  - Dash is a Flask extension that specializes in serving reactive webpages and Plotly graphs through Python
  - Pandas is a software library written in Python for data manipulation and analysis.
  - AstroPy is a library that will be used to parse FITS files from the telescope into data stored on the database
  - Docker is a virtualized platform that will house the operational Flask server when moved to Lowell's remote server.

- Database
    - MongoDB will be the implemented database on Lowell's server to store past exposures in a document-based approach.
    - PyMongo will be used to communicate with the Database from the Flask server to return data

3. Architectural Overview

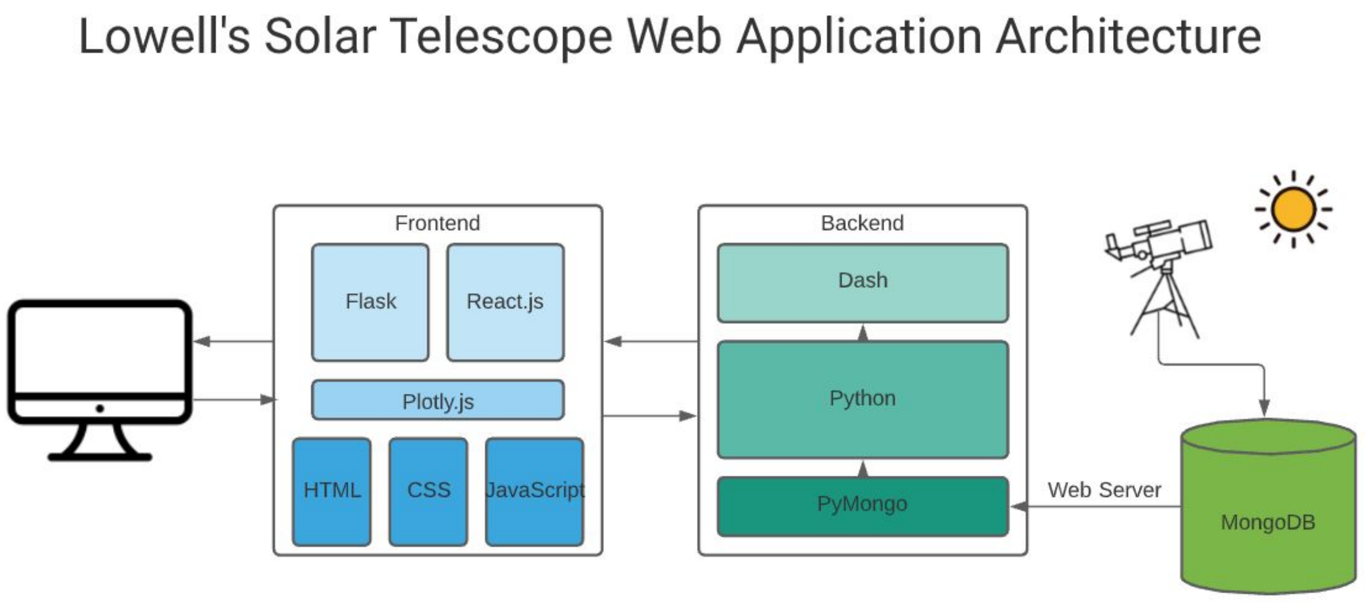


Figure 3.0 - Application Model

The final web application will be built using a Python backend that will interact with a database to render visually appealing, interactive graphs with the Plotly library. The entire database will all be hosted on the Lowell Observatory server. The architecture of the application consists of 4 main parts: the database layer, the Python backend, the front end frameworks, and the client-side.

Within the database layer, the Lowell server will host a MongoDB database containing collections of user-profiles and data from the solar telescope. Data uploaded to the server from the telescope will be contained in separate CSV files which will be parsed and entered into the database. The database will contain a collection for all radial velocity data gathered since the project began in August. Each radial velocity has a unique MJD and a set of one-dimensional and two-dimensional spectra data. These two data sets will be stored in two additional collections. Since the solar telescope collects and uploads new data every day, the database will be updated every day. The main purpose of this database layer is to store and organize all data the web application will need. This includes all solar data needed to render graphs, and also all user data need to verify and grant access to researchers.

The second layer of the application will consist of the Python backend. In order to interact with the MongoDB database, the Python backend will utilize PyMongo. Python needs a MongoDB driver to access the MongoDB database, which is exactly what PyMongo is. Documents in MongoDB will map directly to Python classes. The main purpose of this layer is to retrieve the necessary data needed for the front-end libraries to render interactive graphs. This is potentially the most critical layer in the architecture as proper data-retrieval is detrimental to the purpose of the application.

Once the data has been mapped to Python classes from the database, the front-end is able to use the data to render web pages. The application will utilize the front-end frameworks Flask, which serves pages, and React.js, for the Javascript user interface. However, all developer interaction with these front-end frameworks will be through the Dash library. Dash is an open-source framework created by the Plotly team that leverages Flask, plotly.js, and React.js to build custom data visualization apps. Plotly.js will be the Javascript library that actually creates the interactive graphs. By using Dash to interact with the Plotly.js library, we can build the highly interactive web application solely using Python code. In addition, by using Python, it is simple to incorporate all of Python's tools for manipulating the data. Python is a popular programming language for data manipulation, which means it comes with useful libraries for data analysis and visualization. The library we will mostly be using is Pandas. Even though Pandas has a few of its own plotting capabilities, Dash will allow for the access of all the additional plotting capabilities of Plotly's existing framework. Specifically, using Dash will allow for the use of Plotly's interactive graphing features.

Finally, by using this combination of technologies, HTML pages containing highly interactive graphs can be generated from Python code. With some additional styling using custom CSS and CSS libraries such as Bootstrap, the client will be able to access LOST data in a visually appealing and user intuitive way.

4. Module and Interface Descriptions

4.1. Graphing

One of the main components of our team's project will be graphing the Radial Velocities and High-Resolution Spectroscopy in both one-dimensional and two-dimensional graphs. These graphs are based upon the data recorded out of the Lowell Observatory Solar Telescope. Graphing on a web application can be a complex task, with multiple technologies assisting different components of the graphing process. Some of the technologies in this project that are related to graphing are Python, Plotly (a Python library), and finally, MongoDB will be used as the official database of the whole design.

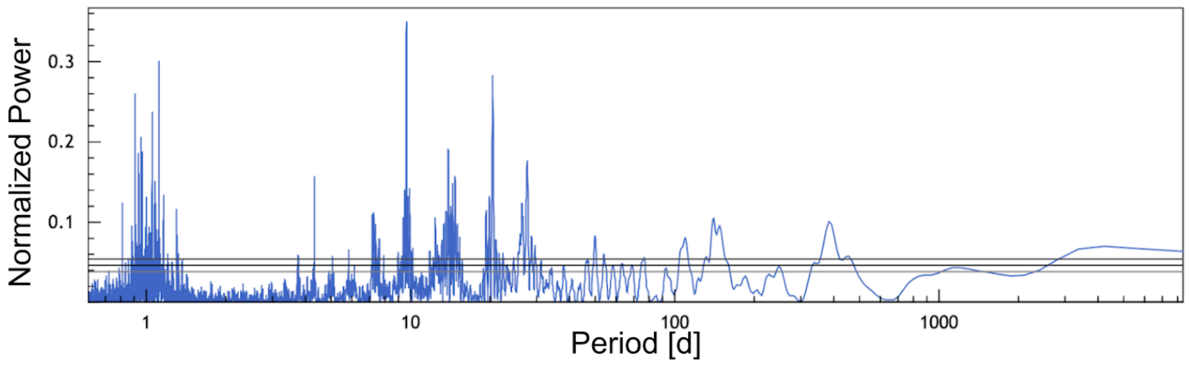


Figure 4.0 - Radial Velocity Graph

Figure 4.0 shows one of the graphs to be created, that being a Radial Velocity graph. The radial velocity of a given object with respect to a point is the rate of change of the distance between the point and the object.

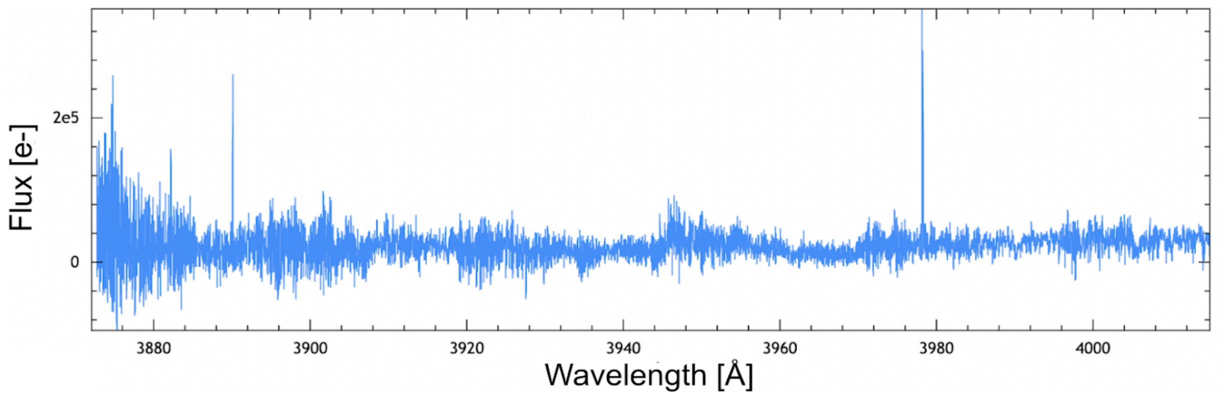


Figure 4.1 - Spectroscopy 1D Spectrum

The other graphs to be created are one-dimensional and two-dimensional spectroscopy graphs. High-resolution spectroscopy is used to analyze chemicals, biological and pharmaceutical compounds, food and beverages, and high-tech materials. In this instance, it is used to measure variants in distance from the sun.

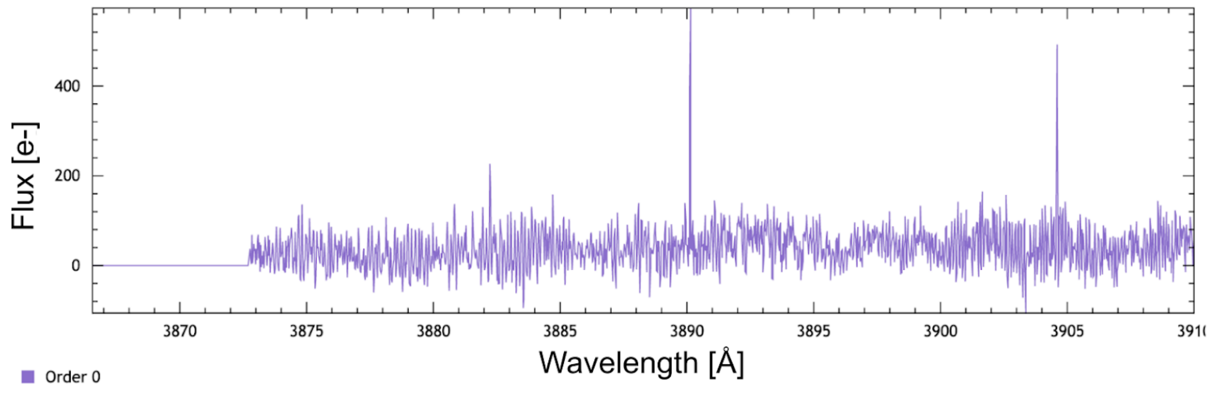


Figure 4.2 - Spectroscopy 2D Spectrum

Python will be used to parse through the data recorded by the LOST and input the data directly into a database. The data is recorded by the telescope and then saved as a Flexible Image Transport System (FITS) file. This file is then converted into a Comma Separated Values (CSV) file. The CSV files are what the Python parser is looking through to collect specific data points that need to be graphed. After collecting the data, the parser will dump what is needed into the database for storage and future use.

Plotly is a Python graphing library with a plethora of uses that it can be applied to for instance; data can be imported, copy and pasted, or even streamed to be updated live on a web application. Plotly will be doing the bulk of the work on this project and it is vital that it be used for easy clean and simple graphing of very complex data.

MongoDB is the database that will be used. MongoDB is a NoSQL database application that uses JSON-like documents to store data. The reason that this DBMS is vital is due to the fact that it is very scalable. When the project is finished it will be in continuous use for an unknown extended period, so any added implementation will need to be doable, as well as the fact that data will be added to it daily for the rest of the web application's life span, meaning the database will need to be able to grow at a rapid pace without anything being overloaded.

Ensuring that the data is accurately and precisely graphed is very important to this project. Because the team's system uses Dash, the webpage should be defined in Python, written similarly to the HTML in the system, but as Python scripts rather than tags. To display a graph on the webpage a Plotly graph is defined, then the name of the graph is set wherever it is desired on the page layout. Any reactive elements can be implemented with callbacks, where input fields can be defined as function decorators, and the Plotly library will automatically calculate the outputs.

4.2. Server

The server module will handle the storing of all data needed for the website. The data stored will include graphing points that are sorted by day, as well as account information for each user. The server will be accessed by users via the website in order to view whatever graphing data that they wish to view.

The primary use case of this module will be to query data points from the MongoDB server. Whenever the user wants to access specific data from a certain day, they will make a database query from the graphing webpage. The query sent by the user will retrieve the spectrum, 1D, and 2D graphs for the chosen day. There will also be a function to handle whether the current user can access data from days that are currently set to private access. This function will check whether the user is logged in or not in order to check if they are allowed to view private data.

This module will also handle the storing of account information. For every person that makes an account, a username, password, email, and institution they belong to will be stored on the server using a database. Whenever a user tries to log in, a query will be sent to the database that compares the sent username and password to all username/password pairings. If there is a match, the user will be successfully logged in, otherwise, they will have to try again. Wherever a user goes to their account page, they will be able to see and/or change all of their account information besides their password.

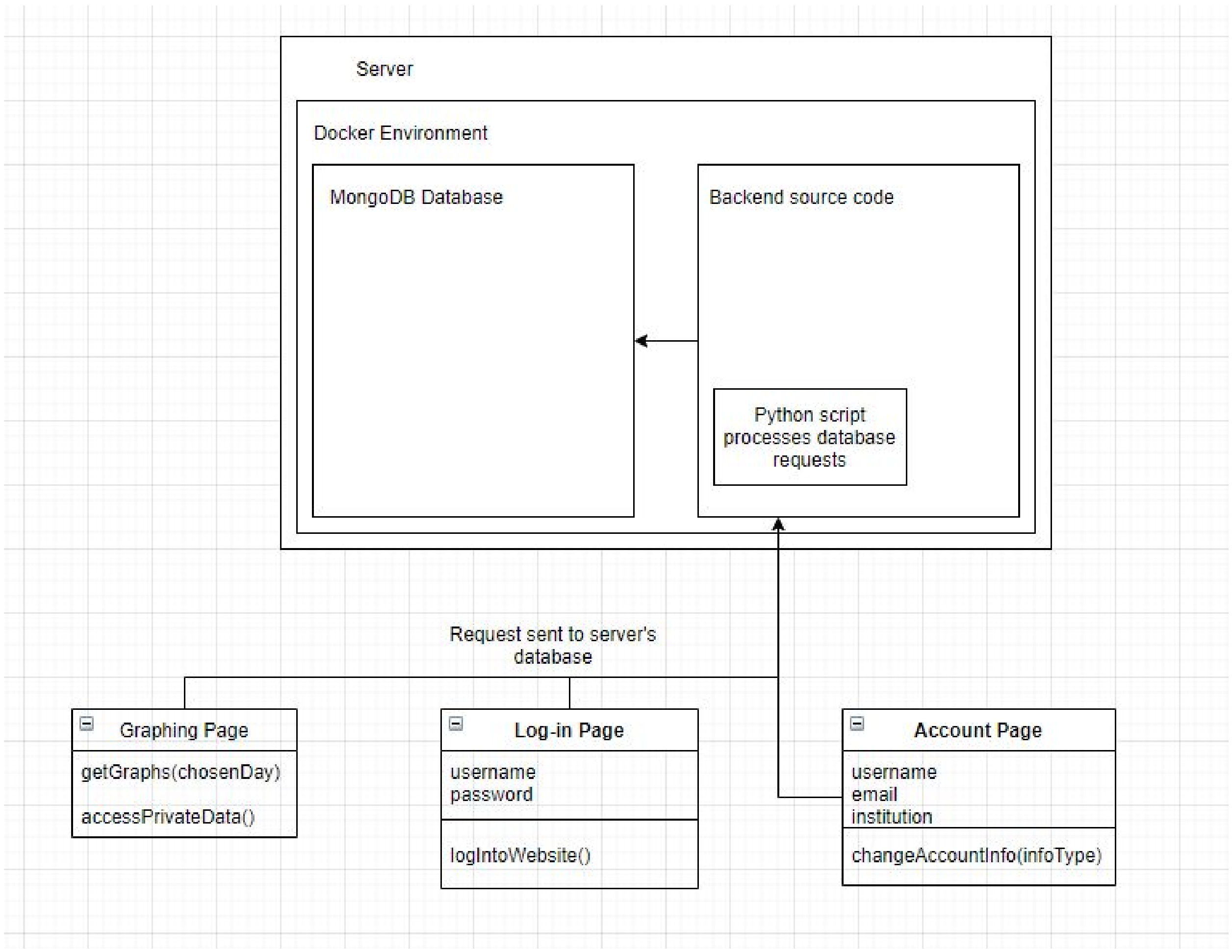


Figure 4.3 - A UML diagram of the server module

The server module will handle all of the functionalities listed below:

- getGraphs(chosenDay) - This function will send a query to the mongoDB database and retrieve all of the plotted points for the selected day. When this data is retrieved, Plotly will be used in order to plot the radial velocity, 1D, and 2D graphs for the corresponding day.
- accessPrivateData() - This function will be called whenever a user tries to access a data set. It will be used to check whether or not the data is set to private. If it is private, this function will then check if the user is logged in and only grant access if they are logged in.
- logIntoWebsite() - This function will handle logging in for users with an account. The username and password that the user inputs will be checked against all accounts stored in the mongoDB database in order to verify if the user is allowed to log in or not.
- changeAccountInfo(infoType) - This function will change a user's own account information, based on which type of account information they wish to change. The three types of information that users are allowed to change about themselves are username, email, and institution they belong to.

4.3. Web Application

The website is how users will be able to view the graphs. Navigation through the site is mostly the same for everyone. However, both researchers and administrators will be able to access more website functionalities than a public user is allowed to.

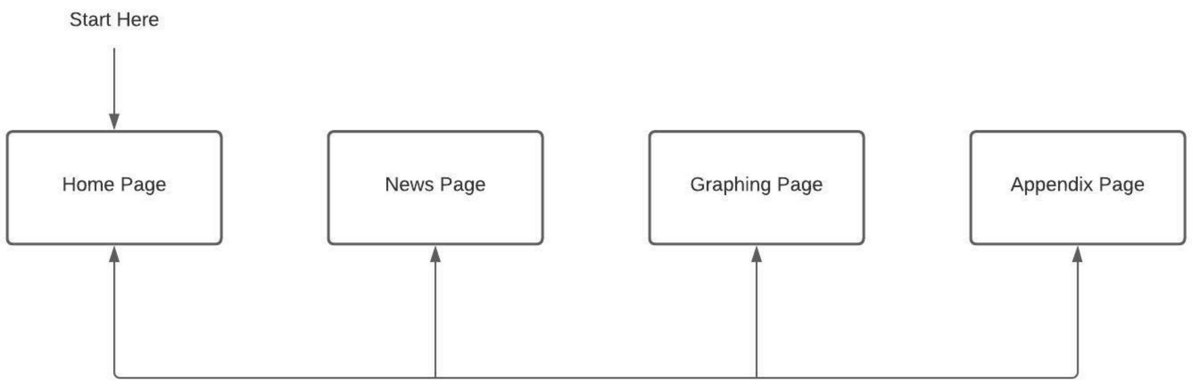


Figure 4.4 - Public User Diagram

When first loading the site, everyone will have the public level of access. For navigation, a public user can navigate back and forth between the home page, news page, graphing page, and appendix page. This is made simple by a navigation bar at the top of each web page. The home page will display a selected Spectrum graph and a news story, both decided by an administrator. A user who views the news page will be able to read articles surrounding Lowell Observatory, the Sun, and other interesting astrological discoveries. The most recent article will be first on the news page, with the option to view older articles. The appendix page will include terminology definitions and equations. This will allow users who are more unfamiliar with astronomer jargon to understand the rest of the website.

All users will also be able to see the graphing page. This page will graph the data gathered from the Lowell Database. The first graph on the page will be of the radial velocity data dating from the start of the project to what the administrator has allowed the public user to view. A second graph can be viewed when a user clicks on one of the points of the radial velocity graph. They will have the option to view either a one-dimensional or two-dimensional spectrum graph.

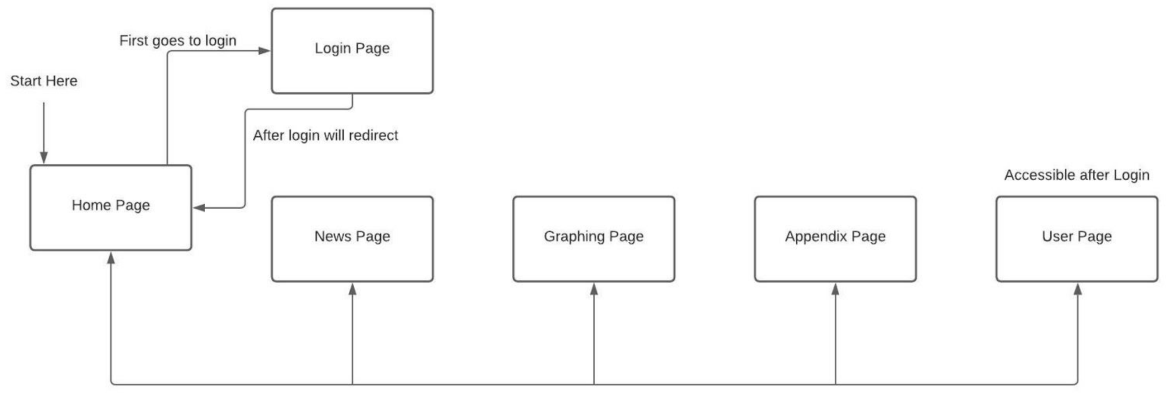


Figure 4.5 - Researcher User Diagram

If the user is from a university or observatory and wants to access more data, they could then apply to be a researcher for the website. To do this a user would go to the login page to sign up. After signing up, the user would have to wait for approval from the administrator to be a researcher. Once approved the researcher will return to the login page and enter their login information. It will redirect them back to the homepage. Now the researcher will have two differences. The first being a user page where they can update their information including institution and email. The second difference is the graphing page. Now they will have access to the full listing of points on the radial velocity graph. Also giving them access to the rest of the spectrum graphs.

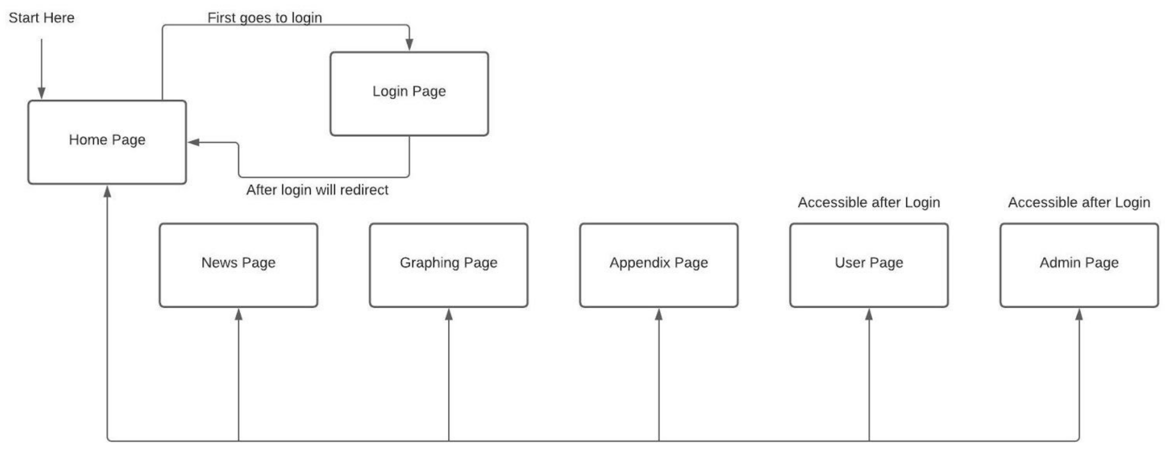
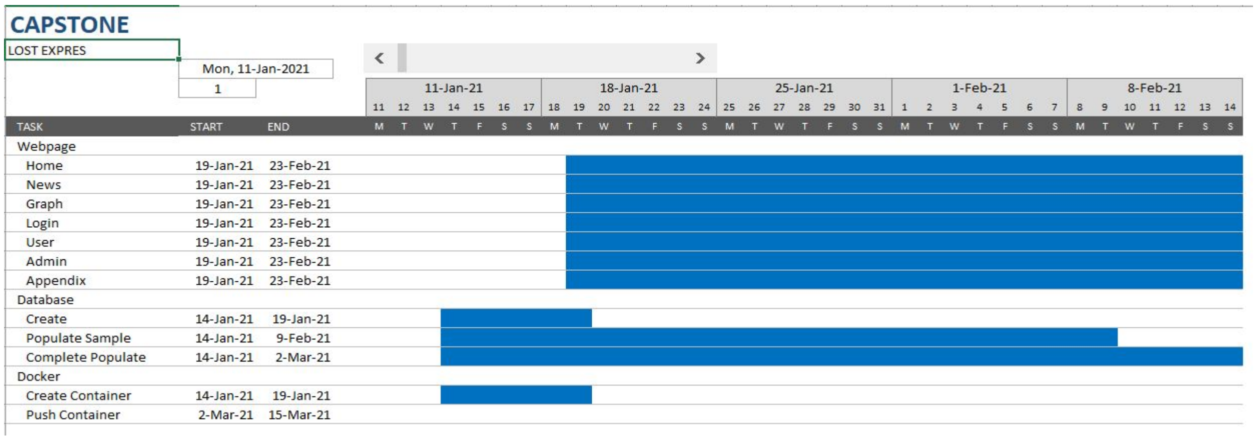


Figure 4.6 - Admin User Diagram

An administrator will have to log into the website the same as a researcher. Once logged in they will have the same user pages as the researcher along with an Admin page. This page will allow the administrator to set the home page as well as what data the public is able to view. This will be viewed as a chart where all the administrator has to do is select with a radio button.

5. Implementation Plan



CAPSTONE
LOST EXPRES
Mon, 11-Jan-2021
1

TASK	START	END
Webpage		
Home	19-Jan-21	23-Feb-21
News	19-Jan-21	23-Feb-21
Graph	19-Jan-21	23-Feb-21
Login	19-Jan-21	23-Feb-21
User	19-Jan-21	23-Feb-21
Admin	19-Jan-21	23-Feb-21
Appendix	19-Jan-21	23-Feb-21
Database		
Create	14-Jan-21	19-Jan-21
Populate Sample	14-Jan-21	9-Feb-21
Complete Populate	14-Jan-21	2-Mar-21
Docker		
Create Container	14-Jan-21	19-Jan-21
Push Container	2-Mar-21	15-Mar-21

Table 5.0 - Implementation Gantt Chart

The chart above shows the estimated time and work of each team member to complete the web application. The main sections we have to complete include the Docker environment, the Database, and the actual web pages. Our client has asked for the project to be within a Docker container. In order to fulfill this, we will do periodic testing of the web application within a Docker environment. This way we can ensure when we hand over the container there should be no issues.

Another section is the Database, including creating tables, doing sample populations, and then ending with a full population of all data our client has gathered. We want to start with samples before worrying about all of the data for testing and development purposes. The samples allow those working on the graphing section to implement the page.

The last and major section is completing each web page. Most of these webpages are tightly coupled, which means we are having to work on all the pages around the same time. Most pages should be handled mainly by one person except for the graphing page. Since it is the main focus of the project, there are three people assigned to it. We hope this will allow that section to be completed as soon as possible, allowing the other pages to be fully implemented.

6. Conclusion

The design of the LOST web application has been carefully thought out to ensure the most efficient user of resources and the fastest user experience. By using new and modern frameworks, Lowell will have a state-of-the-art data visualization tool for analyzing solar data. The key features of the design include using MongoDB as the back-end database. This NoSQL database stores data in individual documents that have a JSON-like structure. This results in simple integration with the Python backend, using the MongoDB driver PyMongo. Using Python, the go-to big data language for modern companies will provide the project with various data analysis and visualization libraries. This data-focused backend will pair seamlessly with the Flask server. The final key piece of the software will be Dash, a Flask extension. This will be the central technology that utilizes Plotly.py to serve reactive web pages.

The LOST EXPRES team has created a skeleton site for the web application by utilizing Python, Dash and Bootstrap. Available pages to navigate to include the home page, data page, appendix page, login page, and an account page. Current functionality includes a working Plotly visualization of three days worth of radial velocity data. In addition to zooming in/out, taking screen captures, and adjusting the axis, the application also recognizes “clicks” on the graph, which consequently display another Plotly graph.

By using these key technologies, we will be able to deliver, via a docker environment, an interactive experience to multiple types of users. Public users will gain insight into Lowell Observatory’s research, satisfying their mission of public education and outreach. Meanwhile, research users, approved by a Lowell admin, will be able to view all of LOST’s data, advancing space exploration and scientific collaboration. The sole admin user, our client Dr. Llama, will have the satisfaction of sharing his research findings in a modern way while maintaining control over the application. Overall, the design of the LOST web application ensures that Dr. Llama will receive an elegant solution to his current problem: the lack of any simple solar data visualization. The technologies we chose will all contribute to a simple, yet pleasing application that is reliable and scalable to the future of Lowell Observatory.