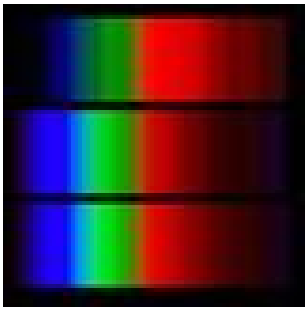

Technological Feasibility Analysis

2022-11-09

Project Sponsor: Joe Llama, Lowell Observatory

Faculty Mentor: Rudhira Talla, Northern Arizona University

Team: Empyrean- Henry Fye, Nhat Linh Nguyen, Jakob Pirkl, Jacob Penney, Kadan Seward

Overview

The purpose of this document is to identify the key technological challenges and major design decisions that our project presents, explore what approaches or existing products may exist that could address those challenges, and make a preliminary decision on which approaches or products the team is going to choose to solve each of these challenges.

Contents

1.0 Introduction

Astronomers rely on a variety of tools to obtain the valuable data that they need to understand our universe and our place in it more fully. Telescopes are commonly perceived as the main tool employed by astronomers, but they are not the only tool - astronomers additionally observe incoming light by separating it into its constituent wavelengths utilizing what is called a spectrograph. Many astronomers use these every day to gather as much data as they can from the celestial objects they observe. However, these machines are incredibly precise, and incredibly delicate and thus require a long and complicated setup which must be done in person - sometimes taking up to 4 hours. Further, setup has to be done at night since that is when the most useful observations are done. And after setup, astronomers must be physically present to send the spectrograph commands to produce actual results. This means that the astronomers have to work many late nights a year doing complicated setup and control instead of doing analysis or more study which is a much more valuable use of their time.

Dr. Llama of Lowell University uses spectrographs to search for smaller exoplanets using the extreme precision the spectrograph provides. He currently must be at Lowell to perform observations, with no option to do them remotely. The operation of this tool, being a complicated and technical machine, requires much practice and has a steep learning curve. This barrier often limits the research of astronomers as they have to invest much of their valuable time just to learn the tools they need to do their job.

As mentioned previously, these spectrographs take hours to set up, which can lead to mistakes and wasted time for the astronomer. Often a situation where automation would do well, there are sparse options available to astronomers, and, because they are all proprietary - none without a hefty licensing fee. This offers astronomers a choice: spend hours every night they are able, possibly making mistakes that can affect their data, and lead to frustration, or spend a significant portion of their budget to license this automated software, and expect this to be a recurring expense. Lowell has taken the proprietary route, using software that is mostly adapted for amateur use, leaving it inadequate for Dr. Llama.

Empyrean sees this as a serious and unnecessary decision. Open source software with the power to automate these spectrographs allows astronomers to have both the extra time they save from automations as well as the budget resources they do not have to spend on licensing. Empyrean envisions an interface that will allow astronomers to operate their spectrograph remotely, be it from a nearby office or from their own home. Further, an implementation where the user could log into a website, set up their spectrograph, choose a place in the sky to focus on, and see the resulting data once observations are complete. The front end interface would be a modern way to interact with the spectrograph, requiring no experience with Fortran, and the automatic setup would solve all of the complications involved with the daily use of this invaluable tool. In sum, we are creating a simple, free-to-use tool that will allow astronomers to gather data they require faster, easier, and more comfortably.

2.0 Segue into Technological Challenges

To accomplish this goal, our team will require an array of technologies that satisfy and possess various specific needs and qualities, respectively. In the succeeding section, “Technological Challenges”, we introduce our preliminary investigation into the specific high-level requirements innate to our project. In the subsections therein, we explore the qualities that each technology should or must possess for us to best overcome the corresponding technological challenge, the various technologies that exist that we have evaluated to be suitable candidates, and which of these we expect to actually use in the implementation of our software as a consequence of its fitness for our implementation’s needs. In the “Technology Integration” section, we elaborate the high-level architecture we’ve envisioned for integrating these chosen approaches together to produce an effective and coherent system and give a walkthrough of the user’s experience after this architecture has been realized. We will conclude the document with a recapitulation of the problem that our software intends to solve, highlights from the document which summarize our plan for solving the problem, and a brief statement about our plans moving forward.

3.0 Technological Challenges

We will need:

- A web front end (ReactJS) that, to a professional astronomer, behaves intuitively and presents relevant, desired information in a sensible and readable way.

- A highly responsive messaging server (built in Flask) that can parse and execute requests from the front end, receive information from the back end, and accurately deliver information to both.

- A database (PostgreSQL) that quickly retrieves moderate to large amounts of data, has an internal structure that corresponds to the structure of the client's data, and is as scalable as the needs of the client dictate.

All of the solutions to these challenges must be built using technologies that are commonly known to professional astronomers, who are the demographic that our tool is built for and will be used and primarily maintained by.

4.0 Technology Analysis

4.1 Front end

4.1.1 Challenge

From 1995, when JavaScript was first developed, to almost a decade later in 2006, when JQuery was first introduced, development of dynamic websites was fraught. For several years, there were no standards in how JS had to be implemented, leading to websites working in one browser, but being completely broken in another. The end result of all of this hassle were websites that were time consuming to build, dreadful to test, and inconsistent to maintain.

To improve the experience of developing web applications, a software engineer called John Resig designed the first framework for developing Frontend JavaScript applications: JQuerry. This allowed developers to have increased stability while developing - one of JQuery's primary features was that it was cross-compatible on all major browsers at the time, while also improving developer interactions building user interfaces. Since 2006 when JQuerry was first released, dozens of other JS frameworks have been released, each with their own style and optimizations to make it easier to write website code.

Therein details the importance of a frontend framework: it allows code to fit into a standardized format through more modular or condensed coding practices, it simplifies the structure of projects by combining parts of development, and it reduces cost of maintenance via shared expectations about the code.

4.1.2 Desired Characteristics

Previously it was mentioned why frameworks, especially for web development are important in general, but this does not imply that these will be the same characteristics to be desired in this project. There are dozens of frameworks, for example, that standardize format, but that does not mean that they standardize code in a useful way for Empyrean. Additionally, a candidate frontend will need to have more specific characteristics to be more apt for our situation. The following are the qualities to be considered in choosing which technology to adopt:

- **Mature and Robust Community**

  The benefits of a community that have been around for a longer length of time have more support in forums, especially if the support is still strong. This also comes with greater support from packages, as more people are interested in using the framework. In our project, neither the team, nor the project sponsor, who will be maintaining the project after the end of Capstone, have much experience with frontend development, and thus support from a community in case of issues will be vital to a smooth pace of progress.

- **Comprehensive Documentation**

  Documentation allows for an easier writing of new code as well as parsing of existing code. The reason this will be important follows from the previous point. When this project is given back to the sponsor, there will be questions about some of the existing structure or functions. Full and well written documentation will ease the work of figuring out how to further improve the project beyond the capstone experience.

- **Sponsor and Team Familiarity**

  The sponsor of this project, Dr. Joe Llama will be maintaining this project in the future, as mentioned before. This means if Dr. Llama has experience with a frontend, any alternative would need strong evidence to disregard existing experience

- **Ease of Maintenance**

  As the team develops new features, new bugs will doubtless be found, and fixing them must be a priority. Thus the team must be able to quickly diagnose and correct errors in code. The easier it is to maintain i.e. search and refactor code, the easier it will be to fix bugs and move on to newer features.

- **Simple to Learn**

  Due to the relatively short time frame of the project, it would be introducing unnecessary difficulty into the project to work with a framework which required extensive knowledge of web development in order to be effective. Thus, the chosen technology should be easy to start, easy to use, and easy to learn about.

4.1.3 Alternatives

With the qualities of an ideal frontend defined, options can be searched for. It was noted that many of the qualities above were directly correlated with the popularity of the tool. Thus, the following choices were largely picked for their market share of web development as of 2022.

First is React.js, developed by Meta, and the most popular of the web development front ends in 2022. React has been around for almost a decade, having been released in early 2013 and is often the first choice for new projects. One of the reasons it is so popular is that it has been used in many huge apps that live in the public consciousness such as Facebook, Netflix,

Uber, and dozens of other unusually successful products. This framework was also recommended to the team by the sponsor.

The second most widely used frontend as of this year is Angular, Google's design of a web frontend. Originally released in 2010 as Angular JS and rewritten into Angular in 2016, Angular has attracted many large companies like Microsoft Office, or PayPal, as well as being the tool of choice for Gmail, the backbone of so much communication. One of the major draws to Angular is its use of Typescript allowing for safer code development.

Last, but only barely, is Vue.js. Vue is technically the newest of these three frameworks, being released in 2014. Vue is also gaining popularity in China, having been adopted by large Chinese companies such as Alibaba, Bilibili, and Xiaomi. In China, Vue is nearly as popular as React, and has been for several years.

4.1.4 Analysis:

To compare each of these tools, the ideal criterion for a framework were distilled into the following quantitative as well as qualitative measurements to be compared side by side.

- **Time on the Market**

  To evaluate the maturity and robustness of the community, the time on the market will be measured, as a longer community will be more mature. The share of the market value will also be discussed. A tool which has more of the market share would have more forums as more problems would arise.

- **Comprehensive Documentation**

  To analyze comprehensive documentation, three characteristics will be analyzed. The quality will be subpages on each document. Meaning that if one piece of documentation references another, a link should be provided. The second factor is examples. Examples in documentation increase understanding, so pages with more examples will be more useful for a team learning these new tools. Finally, Parameter descriptions will be analyzed for understandability. Ensuring that any one document has language that is not too technical for the novice, while still being useful will be important for learning a new frontend and its capabilities.

- **Sponsor and Team Familiarity**

  This quality will be analyzed by asking if any group member, or our sponsor have experience in each of these tools.

- **Ease of Maintenance**

  This will be analyzed via each tool's strength of convention. As described below, a simple app was generated with each of these tools. Maintenance can be performed

more easily if there is good convention to write code in, as programmers down the line will not have to spend as much time parsing existing code.

- **Simple to Learn**

  This was analyzed by creating a simple application in each of these tools. While creating each app, attention was paid to the ease of component usage, a feature that each of these frameworks have, as well as the simplicity of its syntax. Each of these contribute to a likeliness that the framework will be easier to adopt and work with by our client and other professional astronomers.

4.1.5 Chosen Approach

After applying these analysis to each of the frameworks, React, Angular, and Vue, each had places where they shined, and weaknesses. Some of the most important findings will be mentioned, before a table is presented with the rest of the results. First: React is an incredibly popular tool with long time documentation, but its use of different markdown syntax when compared to the others makes it more difficult to learn. Angular is simple to start with and modularizing code, but is too powerful, with many unnecessary tools such as routing that bloat the runtime. Additionally, Angular's documentation suffers due to its rewrite ending in 2016. Vue is a newer tool, and so has a very simple and lightweight implementation, but is not backed by as large of companies as the other two, so it has lacking documentation or forums - especially in English since Vue is more popular in China. Looking at Figure 4.1, each of the products were compared with the analysis criterion.

Relative Rankings of Various Front Ends Per Desired Characteristic			
Framework	**React.js**	**Angular**	**Vue.js**
Market Share (%)	**40**	**8**	**20**
Time on Market (Years)	**9**	**6**	**8**
Ease of Development (Overall)	**1st**	**2nd**	**3rd**
Ease of Component Usage	**2nd**	**3rd**	**1st**
Simple Syntax	**1st**	**2nd**	**3rd**
Strength of Convention	**2nd**	**1st**	**3rd**
Experience in Group	✘	✓	✘
Sponsor Experience	✓	✘	✘
Documentation Quality (Overall)	**1st**	**2nd**	**3rd**
Sub-links (to other pages)	**1st**	**2nd**	**3rd**
Examples	**2nd**	**1st**	**3rd**
Parameter Description	**1st**	**2nd**	**3rd**

Figure 4.1: Each criterion and their value. Unless otherwise noted, the value represents relative ranking when compared with the other two tools.

Because of its ease of development for our use case, as well as the quality of documentation and the experience of our sponsor, React.js is the most apt tool to create our web Frontend. Angular was a close second, but for our purposes, some of Angular's features, like routing, would leave the project cluttered with useless files, making maintenance convoluted. Vue was also very lightweight, with the modularity of its implementation of components being the easiest to use in the limited testing that was conducted, but it lacked strong conventional styles or good documentation when compared to the other tools listed. All of this reinforces the recommendation of Dr. Llama to use React.

4.1.6 Proving Feasibility:

Clearly, reading about a framework and doing simple basics of the service are not the same as developing with React to produce a working project - more work will need to be done to confirm that this will be the correct choice for our use case. There are two demos that would demonstrate React is an effective tool for automating astronomical tools. One, would be to

design a setup to be able to have an interface to collect login information from a user, clean the data and send it to the backend, and redirect the user if the credentials were valid. The second would be a demonstration of the main use case of the project: being able to design an interface to select a celestial body to observe, and send data about this object to the backend for analysis.

4.2 Back end

4.2.1 Challenge

Empyrean has considered the backend part of the website application as a crucial design decision because the backend plays an integral part in connecting the front-end (section 4.1) and the database (section 4.3). More specifically, Empyrean plans to apply the website framework which is capable of handling the big data from astronomy APIs and provides for developers the basic features for a website application, which will be discussed in more detail in next subsections.

4.2.2 Desired Characteristics

For Empyrean, an ideal solution for a website backend framework would be the capability to connect to APIs, ease of use with databases and the speed/performance of data transition from backend to frontend.

- **Support in APIs Connection**

  The main goal of the project helps us to declare why this criteria is the most important and considered in a website application. Overall, our project concentrates on manipulating the data from spectrographs and telescopes. Obviously, connecting to these APIs is necessary to improve the speed/performance of the website.

- **Ease of Use with Databases**

  The database is the most important part of a website controlling the spectrographs and telescopes. Because astronomers need to utilize spectrographs and telescopes every night, the amount of data received from these tools is huge and needs to be controlled by a database for data manipulation. Thus, the website framework selected must be easily accessible into the database through the functions and not too complicated to apply the data utilization.

- **Speed/Performance in Development**

  During the development process, Empyrean realizes that the website needs to handle a huge amount of data and the speed at which pages and tables of data load on a website is important. If speeds are too slow for page load or a carousel is set too fast for comfortable reading, site visitors might feel frustrated.

4.2.3 Alternatives

Based on the desired criterias for a best possible back-end framework, it is necessary to conduct research comparing possible approaches. Firstly, Flask is a micro framework which was created in 2010 by Pallet project and offers basic features of web applications. This framework has no dependencies on external libraries. The framework offers extensions for form validation, object-relational mappers, open authentication systems, uploading mechanism, and several

other tools. All members in team Empyrean have experienced the Flask in Computer Science projects because Flask is free and open source; and the main goal is to implement the CRUD website applications.

Secondly, Django is a web development framework created in 2005 for the programming language Python. This framework offers a standard method for fast and effective website development. It helps you in building and maintaining quality web applications. It enables you to make the development process smooth and time-saving. It is a high-level web framework which allows performing rapid development. The primary goal of this web framework is to create complex database-driven websites. A few members in team Empyrean had worked on other projects using Django as the website backend framework.

4.2.4 Analysis

In order to evaluate the desired characteristics of all the possible approaches, Empyrean has collected the experience from team members and researched a few projects or sources related to these technologies. Below is the analysis of all the desired characteristics:

- **Support in APIs Connection**

  The project focuses on the manipulation of data from telescopes and spectrographs, leading to the necessity of API which helps Empyrean to collect the data. More specifically, Empyrean developers implement connecting to API in each framework and then we compare the complexity in the setup of APIs between frameworks.

- **Ease of use with databases**

  As mentioned at the section 4.2.2, working with data is the most important part and thus the framework needs to support developers in the way of database setup and manipulation. Empyrean developers implement the testing of database manipulation on each framework and based on the experience from each member, the team will decide the ranking for each framework.

- **Speed/Performance in Development**

  In the design phase, Empyrean plans to test different tools including APIs and databases. Thus, the performance of chosen framework will be necessary and able to affect the development of MVP products. The evaluation of this characteristic is based on the experience from team members and researching for a few sources using these frameworks.

4.2.5 Chosen Approach

After the analysis section between Django and Flask, Empyrean demonstrates that the best fit framework for the project needs to support APIs and not be too complicated for using the database. For the API connection, Empyrean considers Flask as the ideal framework since Flask is used for small-scale, simpler applications and a great environment to implement API.

Meanwhile, Django ships with an in-built ORM (Object Relational Mapping) solution compatible with several popular relational (SQL) databases like MySQL, PostgreSQL, SQLite, etc. With Flask, you can work with whichever type of database and mapping solutions you like – through extensions and libraries of your choice. For example, Flask-SQLAlchemy is a popular library for Flask users for managing database interactions using SQLAlchemy, a famous ORM for Python. The table below represents the overall rankings between Django and Flask:

Relative Fitness of Back-end Options Per Desired Characteristic			
Option	**APIs Connection Support**	**Easy Database Implementation**	**Speed/Performance of Development**
Flask	1st	1st	2nd
Django	2nd	2nd	1st

Figure 4.2: Each criterion and their value. Unless otherwise noted, the value represents relative ranking when compared with the other tool.

As can be seen from the Figure 4.2, Flask overcomes the other framework Django in terms of API Support and Database Implementation. On the one hand, Flask has the functionality and unlimited regulations to support the API and access the data directly from the main function. On the other hand, it is undeniable that Django has a more complicated system including routing and database modeling, which makes the API connection not flexible in the setup format. Secondly, the project mainly concentrates on the database implementation, so a framework, which has an easy setup with a database and accepts many kinds of databases such as MySQL and PostgresQL, makes Flask the top choice in this field. Meanwhile, Django requires developers to use the ORM framework to apply the database inquiries, which makes the project stick with one database technology and unable to test others in the design process. Otherwise, the speed or performance of Flask operates slowly, compared to Django, which is a larger-scale framework responsible for building high quality website applications. More specifically, Django REST Framework is the first choice of developers who want to create a Minimum Viable Product example before they start pouring in all their investment into the project. Nearly all the necessary features that an MVP requires come already packed in Django. Consequently, the framework having better rating over all criterias is the Flask.

4.2.6 Proving Feasibility

From the analysis and chosen solution sections above, Empyrean decided to choose Flask as the main back-end framework. It is necessary to implement Flask and test it in a development environment. In more details, the team is planning to implement the Flask framework by creating a demo website using Python as main backend programming language and testing the APIs connection, database application as well as the performance in the development process.

4.3 Database

4.3.1 Challenge

Our application will require a database to store the large amount of astronomical data that our client and their organization will gather. This database will interface with a messaging server to provide the user, via the front end, with data that is relevant to the work that they will conduct using our application, such as lists of recently collected observations, astronomical targets that have been examined, and logs of when these observations and their corresponding datum were made. The challenge for our team is choosing a database which is simple to maintain and interact with, fits the the structure of the client's data well, and performs the aforementioned tasks with a speed that is perceived as fast by humans. If these criteria are met, it will allow our client and other astronomers to maintain their data more easily and will provide performance that makes our application feel snappy and responsive, leading to a higher degree of comfort and ease for the astronomers in performing their work and, consequently, a high level of satisfaction with our product.

4.3.2 Desired Characteristics

The solution to this technological challenge will be a database type which:

- **Is overall well-known and mature**

  We are interested in a solution which has wide support, a very active community which maintains it, is robust, and is proven in industry as beyond competent.

- **Has a well-known query language that performs all of the functionality that our client may need**

  Besides being a mature and well-used system overall, we would like the database management system that we use in our project to have a query language that our client will have experience with and which promotes ease of adoption by new researchers and maintainers.

- **Is easily scalable and, more generally, maintainable**

  The system that we choose must be easy to scale as the amount of data that our client produces dictates. More generally, the database management system should be relatively simple to maintain so that our client and their colleagues, who are not professional system administrators or software developers, can maintain it on their own, without having to rely upon a third party.

- **Offers a model which fits our client's data as appropriately as possible**

  The database that we choose should offer schemas for managing the data, such as the relational model, that fit our client’s data as closely as possible. This will reduce

complexity of use and management by removing the need for our client to configure the database management system to accommodate their data.

- **Is perceived as fast when storing or retrieving moderate to large amounts of data that is structured like the client's data is**

  The database should be able to accept requests from and return the relevant data to the messaging server with a speed that feels responsive to the front end user. This characteristic is subjective because it is based upon perception and we will refine it in further documentation if we receive instructions from the client which specifies a minimum performance standard.

- **Offers strong reliability, security, and data integrity**

  The database that we choose must not lose or modify data in any way while processing an invalid query or if an external failure incident occurs (such as a power outage). An example of a standard that would ensure compliance with these needs would be ACID compliance.

- **Is financially manageable**

  The database should be as cost-accessible as can be found, ideally completely free of cost.

- **Is permissive to use**

  The database should be as permissive to use, per its license, as can be found. For example, the MIT license is considered far more permissive than the GNU General Public License v3.0 because of what it allows the end user to do with the licensed software. The ideal database will be relatively permissive so as to avoid unforeseen legal complications as a consequence of use that we have not accounted for.

4.3.3 Alternatives

There is a taxonomy of databases which categorizes them by their internal structure and abilities, and variegated options exist inside of each category. The primary options that we see as relevant and useful to our project are relational, or SQL. Non-relational, or NoSQL, databases, are very common in web projects such as ours, but we do not view them as viable options for our project. Here, we will explore the options inside of the category of relational databases that we will seriously consider using in our project implementation.

Relational databases, also commonly referred to as SQL databases, are ecumenical and our team was familiar with them not only through exposure to them in our education but also as software users and through work experience. For example, several members of our team have worked with varieties of them, such as PostgreSQL, in internships or formal research positions. The relational model, which “represents both data and relationships between data as tables” and underpins relational databases, has existed as a theoretical tool since 1970, and

implementations have existed since 1973. Since then, relational databases have become the “most common in practical use”, permeating industry.

As referenced above, there are several popular and mature relational database management systems available for software developers to choose from. The most popular relational database management system and database system overall among professional developers is PostgreSQL, which, again, our team has heard of as a consequence of our work and research histories. The package that was to spawn PostgreSQL was developed at the University of California at Berkeley and “was sponsored by the Defense Advanced Research Projects Agency (DARPA), the Army Research Office (ARO), the National Science Foundation (NSF), and ESL, Inc.”. PostgreSQL appears to have been released to the general public in June of 1990 and has been stable since 1991. Since that time, it has been used in “a financial data analysis system, a jet engine performance monitoring package, an asteroid tracking database, a medical information database, and several geographic information systems” and the authors of the documentation claim that it “is now the most advanced open-source database available anywhere”.

The second most popular database among professional developers as of right now, per the 2022 Stack Overflow Developer Survey, and the other options that we are considering is MySQL. Our team was aware of MySQL simply because of how prolific it is, but none of us have ever used it in development. Per the founder and primary developer Michael Widenius, it was released in 1995 and has since been used by major public and private organizations, such as Bank of America, Nasa, the U.S. Navy, Bayer, and Eli Lilly.

4.3.4 Analysis

It is difficult to investigate the database characteristics that we have asserted as desirable because most, if not all, of them must be evaluated under industrial conditions, such as using large datasets, and by performing precise testing to substantiate arguments, for example about performance or reliability. Instead, we attempted to find sentiments about each option from major industrial organizations and used sources published by the organizations that maintain each of the options to gather claims made about the products. Below is a list of each characteristic we evaluated our options by and a description of how that characteristic was evaluated using sentiment and claims analysis.

- **Is well-known and mature overall**

  We investigated the popularity of the option, whether the option was used in industry and by which firms on what size of project, and how long the project has been in use.

- **Has a well-known query language that performs all of the functionality that our client may need**

  Much like the overall maturity metric, we investigated the popularity of the option, whether the option was used in industry and by which firms on what size of project, and how long the project has been in use.

- **Is easily scalable and, more generally, maintainable**

  We investigated whether sentiment indicated that the option was relatively easy to scale up as data grows and whether there was a high degree of complexity in managing the option's model.

- **Offers a model which fits our client's data as appropriately as possible**

  We investigated whether the models offered by the option fit our understanding of how the client expects their data to be structured.

- **Is perceived as fast when storing or retrieving moderate to large amounts of data that is structured like the client's data is**

  We investigated sentiment about whether the option was considered performant under industrial conditions and, if so, via what means.

- **Offers strong reliability, security, and data integrity**

  We investigated sentiments and claims about what features the option afforded which allow the administrators to guarantee the reliability of the option, maintain security, guarantee that gathered data will not be corrupted.

- **Is financially manageable**

  We investigated how costly the options and their various plans were to use.

- **Is permissive to use**

  We investigated what license each option used and evaluated whether they would make usage of the option in our application more complicated.

4.3.5 Chosen Approach

Both options presented above meet most of the desired characteristics that we have proposed for an ideal database management system for our project, specifically characteristics that concern the actual functionality of the database management system. MySQL differs from PostgreSQL in that it has fewer features, occupies a smaller memory footprint, and is considered simpler in general. PostgreSQL has the reputation of being capable of more complex labor by being more featureful. Our client may not need the extensive plugin ecosystem or all of the complex functionality that comes with PostgreSQL, but PostgreSQL has a permissive license and appears to be completely free. MySQL, on the other hand, is simpler and may fit our client's needs a little better in that regard, but it has a more restrictive license and appears to have an admittedly confusing tiered plan system that could involve costs at the scale that our client is working at.

Relative Fitness of Database Options Per Desired Characteristic		
	PostgreSQL	**MySQL**
Overall maturity	✓	✓
Maturity of query language	✓	✓
Maintainability	✓	✓
Model fit	✓	✓
Performance	✓	✓
Relative reliability, security, and integrity	✓	✓
Financially manageable	✓	⚠
Permissively used	✓	✘

Figure 4.3: Each criterion and their value. The assigned values indicate if the tool completely, partially, or dubiously fulfills our desired characteristics.

In the above table, you may see how our team has evaluated the two database options for each desired characteristic based on our understanding of their properties via research. Both options are competent performers: they are mature and proven in industry, fast, scalable, ACID-compliant, offer features that ensure data integrity and reliability, have dialects of the same well-known and popular query language, and have internal models that fit our client's data well. Where MySQL is attractive because of third-party sentiments about its simplicity, PostgreSQL shines because it is completely free and permissive. These two qualities, along with its claims and third-party sentiment about its performance, featurefulness, and reliability, make it the most promising solution as a database management system for this particular application, in our team's opinion.

4.3.6 Proving Feasibility

To validate the facts and sentiments that we've found about PostgreSQL during research, we plan on developing two demos that will exercise the database using simulations of the two most common tasks that the database will perform in production. The first and arguably most important task for the database will be returning relevant pieces of data to the user via the front end. We will create a demo that queries the database for large quantities of data, both large individual rows and large numbers of rows, and measures how fast the operation concludes. The second most common task that the database will perform is data insertion. We want to prioritize testing the simultaneous insertion of large quantities of data, both to ensure

adequate database performance and that the database performs well under the real-world condition of being used by several astronomers at once.

5.0 Technology Integration

No matter the previous considerations of different technologies, it is for a fact that this project is going to require a frontend UI, a backend server, and a database to interact with Lowell's spectrograph and achieve the wishes of our client. The most important part of explaining the different technologies that will be utilized in this project is how these different pieces all fit together into one coherent product. To give a general overview as to how each piece of the puzzle will interact with each other, Figure 5.1 demonstrates a full product.

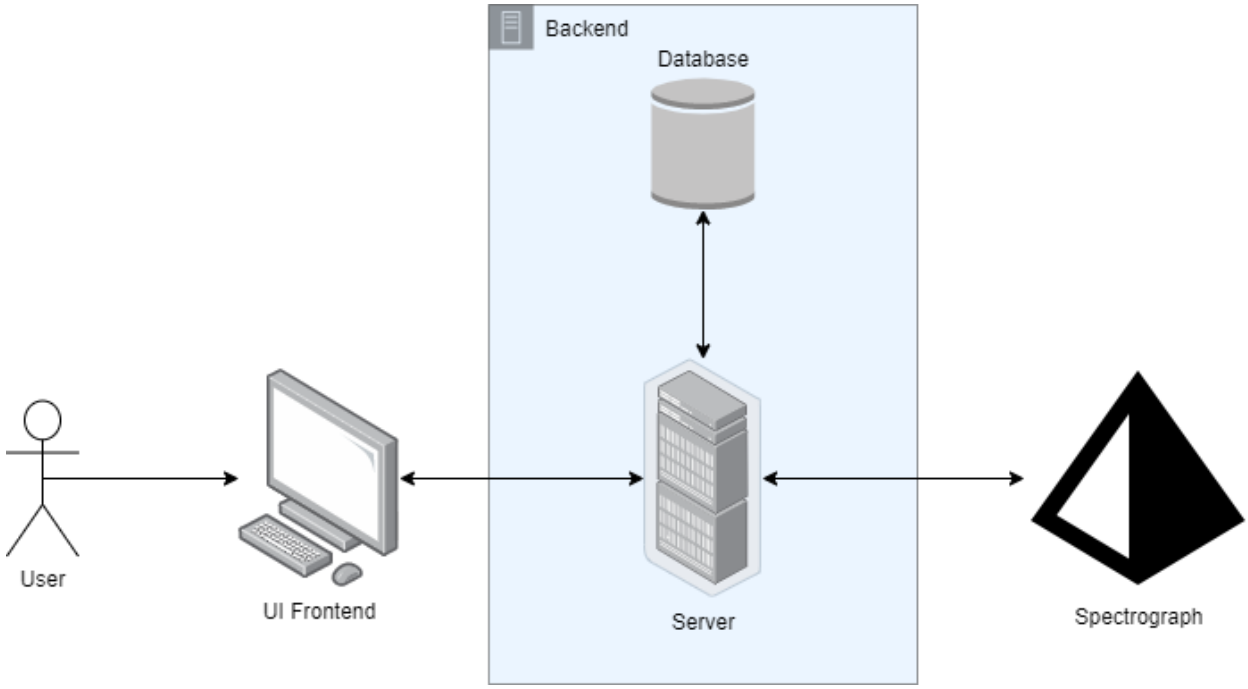

Figure 5.1: Basic high level design for the architecture of the project

5.1 High-Level Architecture

The high-level architecture of this system is fairly straightforward and has a very common setup similar to many other applications. The main point of contact between the user and spectrograph will be a user-friendly frontend UI. This UI will interact directly with a backend server that will act as the engine of the system. The frontend and backend will then work in conjunction to provide a few different functionalities: to store user information, to provide access to a backlog of stored spectrograph observations, and to allow interactions with the spectrograph to create new observations. The database's main responsibility will be taking in, storing, and sending back relevant observational data from past uses of the spectrograph. This data will be passed from the spectrograph to the database through the backend server. This means the server will also need to be able to take in the information provided from the spectrograph, parse it, and place that data into the database in an appropriate format for later querying. The server will also require the ability to talk back to the spectrograph so that when a request is made to create an observation, the server will send that data to the spectrograph and

wait for a response. Once the response is received, the server will ensure that the information is intact and will then store the data into the database.

5.2 System Walkthrough

A simple walkthrough of the system will go as follows:

1. User navigates to the correct web page by inputting a provided URL
2. User logs in or signs up to gain access to the rest of the website
3. User is taken to a homepage that has directional links to access the spectrograph and database
4. User navigates to spectrograph page
    a. User is prompted with and enters relevant observational data to create new spectrograph observation
    b. Observation is sent back to the UI, displays to users, and is stored in database
5. User navigates to database page
    a. User is displayed a table with past observations
    b. User inputs selects/inputs filters to view relevant observations
    c. User exports selected observations

6.0 Conclusion

The tools that astronomers use to conduct their daily research play a key role in helping humanity understand and explore the universe. Creating a free to use, open source software that is easily accessible and helps further research into the cosmos is both an important and exciting task that must be done with the utmost care.

6.1 Highlights

The system going to be built for this spectrograph has a few key components to fulfill its requirements:

- A front end website that is intuitive for a professional astronomer
- A database to store observations and user data
- A messaging server that serves as the backbone of all operations

To fulfill these requirements, a few different technologies have been decided on:

- React
- PostgreSQL
- Flask

These technologies best fit our use case by providing all the functionality required for the project along with the simplicity and widely available documentation to help astronomers understand and contribute in the future. The final product will be a well-organized website that allows astronomers to easily create new observations with a spectrograph while comfortably knowing that all observations are backed up.

6.2 Closing

Ensuring that a project with this many moving parts becomes one coherent system is very important and can also be challenging. We are confident that with our research, the chosen technologies, and the time allotted that there will be no issue delivering on this project to Lowell Observatory and we are proud that we get to create such an exciting open-source product that will hopefully live on for many years to come.