Team Sirius

Technological Feasibility

10/9/2020
Project: NPOI Dashboard Web Application

Team Members:
Mario DeCristofaro
Cameron Hardesty
Hannah Park
Matt Rittenback

Sponsors:
Jim Clark
Adam Schilperoort
Peter Kurtz

Mentor:
David Failing

Version 2.2

Table of Contents

1. Introduction

The Navy Precision Optical Interferometer web application project revolves around the central goal of collecting astronomical observation data which is utilized to better understand our universe and how it works. This project is sponsored by Jim Clark from the Navy Precision Optical Interferometer Naval Research Laboratory, Remote Sensing Division as well as Adam Schilperoort and Peter Kurtz, who are both software engineers at Lowell Observatory.

The Navy Precision Optical Interferometer (NPOI) is the largest optical Interferometer in the world. The NPOI is an astronomical long-baseline optical interferometer that has been in operation in Anderson Mesa since 1994.The NPOI uses an array of six mirrors spaced tens of meters apart to gather data rather than a single telescope, and is so large because it combines star light collected to form a high resolution aperture. The currently collected data is invaluable to inform instrument health and performance. However it remains unusable to administrators, engineers, observers, and researchers alike because this data is so dense, located on different machines as well as on directory trees across the network in a generic text format.

The envisioned solution for this project is a web-based dashboard which quickly and elegantly displays relevant information based on the interested user. This dashboard would provide users a system in which they would be able to interact with graphs of star data relevant to building a better understanding of the space we seek to explore and travel through in the future. This project will at minimum provide a webpage that will be deployed to the server at NPOI with graphs that users will be able to interact with, displaying the most current observational data. This webpage will also display the instrumentation data as plots organized by date. Users will be able to interact with this webpage to edit the configuration of the NPOI, and view individual values of data points by hovering their mouse over a scatter plot’s data. Each station will display its metadata (name of station, status, comment) on hover as well. This web application will provide administrators, engineers, observers, and researchers a more digestible way to view the data collected. This will give the administrators, engineers, observers, and researchers the ability to utilize the data collected at NPOI to its full potential.

In this document the technological challenges such as, security, longevity, and integration are detailed. Following these challenges, this document analyzes which technologies would be best fit to overcome the described hurdles. Finally the technological integration portion of this document will cover how our chosen technologies will integrate together to create a solution that addresses the challenges of the project.

2. Technological Challenges

The goal of this project is to create a web-based dashboard to assist our client with monitoring the different NPOI stations, computers and their status. While also, simultaneously being able to view the collected observational data. This project implementation will lead to several technological challenges but we will overcome these challenges with the technologies we select. Our client works with the military and collects data for individuals that wish to keep their data private, so we will need to select a web framework and database that can ensure a high level of security. The dashboard is intended to be used for many years to come and expanded upon in the future, so we need to consider longevity and ease of use for future developers. Below we will break down the challenges for each specific technology we need to implement for the dashboard.

2.1 Web Frameworks

To implement the web-based dashboard for our client we need to consider a well developed, secure and robust web framework to build it upon. The framework we will choose has to consider the following challenges.

2.1.1 Security

The dashboard will be handling sensitive diagnostic and research data that the client wants to keep private. So, knowing this, the given framework needs to be secure, and ensure that data is not leaked to the wrong individual.

2.1.2 Longevity

The dashboard is intended to be used for many years to come so the web framework technology selected needs to be reliable and have sustainability for the future. The web framework we select must have a supporting organization behind it to show a level of long term support by the framework developers. This long term support is critical to ensure future developers of this project can expand on the dashboard and fix any potential bugs that may arise.

2.1.3 Integration
The full stack development nature of the project necessitates that all technology listed should be supported and integratable. Ideally the given framework must easily work with existing databases as well.

2.1.4 Learning curve
Since the dashboard is intended to be expanded upon in the future, we need to consider which web framework will be easiest or hardest for the team and future developers to learn.

2.2 Database Technologies

The dashboard will be handling and storing large amounts of data related to the NPOI instruments and the observations that it makes. All of this data will be stored in a database and our selected database technology will need to consider the following challenges.

2.2.1 Security
The dashboard will be handling sensitive diagnostic and research data, so the chosen database technology holding all of the data in the back-end needs to be equally as secure as the front-end dashboard. The level of security for this project needs to ensure that data is not leaked to the wrong individual.

2.2.2 Longevity
For the years to come the dashboard is going to be used continuously by people at Lowell Observatory and researchers in Washington, DC.The database must show a level of long term support by a trusted organization to ensure future developers can understand the dashboard to be able to fix any future issues that may arise or to expand on the project.

2.2.3 Integration
The integration for the database will have different tables for the different types of data we will be parsing through. The backend of the database will be getting handled through the team over at Lowell Observatory. Which leads us to just parse the data they have stored, and to upload each specific data group in its corresponding table in NPOI's database. The chosen database will also need to be integrated into the web development framework of our choice.

2.2.4 Management

The database technology will need to have an easy to use and learn management system so that after development is concluded the client will be able to make their own changes to the database when necessary. It will ideally include its own web application for database management to make all of the operations simple and fast as opposed to the alternative of writing your own SQL statements.

2.3 Front-End Graphing Library

The main purpose of the dashboard is to display graphs that help the client visualize the instrument data collected by the NPOI and parse the observational data it collects. When selecting our front-end graphing library we need to consider these challenges.

2.3.1 Longevity

The graphing library we select will need to stay up to date with the web framework we choose as it could cause the whole dashboard to fall apart. Defeating its purpose, if it can't continue to be integrated with each other in the future. The graphing library will be in control of the status meters illustrating whether a station or computer is having issues. So making sure the graphing library we select stays up to data frequently is important moving forward.

2.3.2 Visualization

It is important we select a graphing library that allows us to build and display custom built graphs. Ideally these graphs would also allow us to implement a level of interaction with the graph like drop down menus, status bars, and different styled plots and graphs according to the specific data they are researching.

2.3.3 Integration

The chosen front-end graphing library will integrate with our chosen web framework and database technology. It is vital that we choose a library that does not lose compatibility with future updates of the chosen framework, database, and graphing library. As the graphing portion is responsible for alarming the team at NPOI if there are any issues with the different stations or computers.

2.3.4 Learning curve

There will be a learning curve for the graphing libraries as the majority of the group has not had prior experience with the newer graphing libraries out today. Although, the newer graphing libraries have easy integration features as some are built into

JavaScript libraries, some even come with graphic libraries like Python which could come in handy if we choose to use Django as our web framework.

3. Technology Analysis

This section will outline the decisions made to address each technical challenge. It will overview each challenge individually and introduce different possible solutions we considered for each challenge. Following the explanation of each consideration for our challenges, it will include which solution was chosen and the reasoning for why it was chosen.

3.1 Web Frameworks

The web framework we choose will need to have the features and attributes that we covered in the technological challenges discussed. The largest of which include: security, longevity, the ability to interface with other technology, and the learning curve.

3.1.1 Django

Some members of the team have experience working with the framework. It is a largely popular framework due to the fact that it is developed in Python and its built-in full stack capabilities are easy to use and well documented making it a popular choice for many web applications. Django comes as a complete package so that once installed and configured you can start developing right away.

3.1.2 Ruby on Rails

Rails is another full stack framework that uses Ruby as its framework with common built-in features like, database management and its self proposed simple integration of new technologies. All combined Rails is meant to make many aspects of web application development simpler.

3.1.3 AngularJS

Angular is a much different framework from that of Django and Ruby on Rails, because of its change in focus of where the application should mostly be running. Angular is a front-end based development framework that relies heavily on APIs and the client's computer in order to render pages. Angular focuses heavily on speed of the client's computer to be able to view the web page.

3.1.4 Chosen Approach

	Security	Longevity	Interfacing	Learning Curve
Django	5	4	5	5
Rails	5	4	5	4
Angular	5	4	3	3

Figure 1: Displays the parameters we found important for selecting our web framework. Rated on a Scale of 1-5, with 5 being the best rating in a category.

Based on the results of the table above both Django and Ruby on Rails were neck and neck for our choice of framework. However the largest contributor to the choice of framework came down to the learning curve, with both frameworks having similar features we needed to decide on which framework would be the easiest to work with and due to experience that became Django. In Figure 1, there are also characteristics from each framework ranked, each framework has extensive security measures built into them making them all very secure. All of them have a less than perfect longevity score due to them needing constant updates and the rare chance that the people developing them choose to end support. Angular is the only framework that did not score perfectly on interfacing with other technologies and that comes from its front-end nature forcing connections with things like databases to have to be done in a roundabout way. The learning curve differed between them with Django scoring the highest due to the teams familiarity and the extensive documentation. Rails scored a small amount lower due to the team having no familiarity with the language despite the framework having great documentation. Finally, Angular scored the lowest due to the odd structure the framework carries and its younger age meant that it had less documentation than the other two. All three choices had their pros, like Angular with its speed and Rails with its easy injection of new features, but the downside of learning a new language with Rails or complicating our connection to a database with Angular made Django the best choice. While Django may not excel in the areas that Rails and Angular do, the deficiencies can be made up for in the familiarity the team has with the system and the extensive documentation to learn any new systems within the language.

3.1.5 Overall Feasibility

Given our choice of Django the best steps moving forward would be to make a small mockup web application that implements the other technologies we will be selecting, like the database technologies, front-end libraries, and graphing libraries. and shows that they work together harmoniously. To further test the system it would be necessary to implement a small database with some test data to show a mock-up of what the final dashboard may look like when sent to the client.

3.2 Database Technologies

One of the large technological challenges we need to overcome for our project is a database to store the collected data in. The large components that need to be looked at when we choose the database system is the compatibility of the database with Django, how easy it is to manage the database and the security of the database.

3.2.1 MySQL

MySQL is by far one of the most popular database management systems out there, boasting one of its large selling points being extremely fast read operations. It contains all of the features you could want for a database management system making it a great choice for a wide variety of projects.

3.2.2 PostgreSQL

PostgreSQL is one of the main competitors to MySQL containing a wide variety of features including an admin portal and very fast write operations. PostgresSQL is an object-relational database rather than just a relational database which gives it features like table inheritance and boasts itself as being “the most advanced open-source relational database in the world”.

3.2.3 MongoDB

MongoDB is a NoSQL database that primarily stores its data in a JSON format making it highly flexible and simplifying a few operations like adding new data and querying the database. The storage of data in JSON files allows the data to be stored virtually anywhere on the system. Being a fairly new piece of technology, having only been released in 2009, Mongo is missing some of the more classic features you would find in a typical SQL DBMS.

3.2.4 Chosen Approach

	Django Compatibility	**Management**	**Security**
MySQL	**5**	**5**	**4**
PostgreSQL	**5**	**5**	**4**
MongoDB	**2**	**3**	**4**

Figure 2: Displays the parameters we found important for selecting our database. Rated on a Scale of 1-5, with 5 being the best rating in a category.

Figure 2 illustrates the criteria by which we selected a database management system on a rating scale. When analyzing the compatibility of Django, both MySQL and PostgreSQL are natively supported by the framework. MongoDB would prove difficult to integrate into a Django system given both how the database management system is structured and its NoSQL format. In the management field again both PostgreSQL and MySQL have fantastic management programs to easily manipulate data and edit tables, while since MongoDB saves all of its data in JSON files most data would need to be edited manually or by a much slower process than that of the other two choices. Finally in the security field all 3 database management systems implement extensive security measures on the data but as with almost all forms of data storage they are not perfectly secure. Ultimately our choice of database management system came to be between PostgreSQL and MySQL since our team already has experience working with SQL and our client is already planning to work on their backend in SQL. MongoDB seems like a very novel system and could become a real competitor in the future but the NoSQL format and the lack of true admin software knocked it out of the decision. When it came between Postgres and MySQL both database management systems were extremely similar. Although overall the final choice came down to comfortability and the client having some existing systems already working in MySQL, this meant that MySQL would also end up being the easiest for the client to manage after our work on the project is done.

3.2.5 Overall Feasibility

With the choice of MySQL as our database management system, we will need to link it into a basic Django web application to prove that the two systems are fully compatible by integrating it into a small Django web application. Further testing will be needed to look into the management side of the database to ensure that it can truly fulfill the data storage we need and add in the testing data we will need for other technologies.

3.3 Front-End Graphing Library

One of the major features we need to include in the dashboard is a group of interactive graphs for the client's diagnostic data. This leads to a choice of a few JavaScript libraries with a focus on graphing and other features that may be applicable to our project. We chose to work with JavaScript libraries since the team is familiar with JavaScript and it naturally integrates with Django allowing there to be a small learning curve on working with the libraries.

3.3.1 D3.js

D3.js is a JavaScript library for manipulating web pages based on data. It functions by binding data to a Document Object Model and then applying data-based transformations to the document. Alternatively it can also be used to take the same data to create scalable vector graphic based charts with different transitions and interactions.

3.3.2 Chart.js

Chart.js is an open source charting and graphing project distributed as a JavaScript library. It contains 8 different animated and customizable chart types with support for all modern browsers. Chart prides itself on being simple yet flexible for developers who need to chart their data.

3.3.3 Plotly.js

Plotly.js is an open source graphing library that is actually built on top of D3.js and stack.gl. It includes over 40 chart types including 3D charts, statistical graphs and SVG maps. A large benefit to Plotly.js is that it has built-in support for heatmaps which was a part of one of the features our client wanted implemented in the dashboard.

3.3.4 Chosen Approach

There were no specific criteria that we based the decision of our graphing library on. We decided the library based upon how robust it was and if it included features that were beneficial to the project. The robustness was determined by the number of graphs and data supported by the library which Plotly.js completely trumped the other two libraries in that field. Then when it came to feature support Plotly.js supported more than just graphs but had a few other features we were looking for, a key one being interactable graphs so the client could narrow or widen the field of data being plotted. Plotly.js's support of over 40 different graph types as well as essentially including the D3 library made it a far superior choice to the other options. This taken into consideration with the fact that we no longer would need to find a custom heat mapping library which could allow us to meet a stretch goal for the client when displaying both star and diagnostic data made it the perfect choice for our needs.

3.3.5 Overall Feasibility

Testing for the chosen graphing library will ultimately come down to the implementation of a few graphs with some testing data. We would need to test out some of the other functionalities of the library like its heat mapping capabilities among other features if we can get to that point of this project.

3.4 Basic Front-End Libraries

Front-end libraries are key in saving development time and making a unified theme for any web application. We selected two different libraries one for JavaScript and another for CSS that should drastically simplify the programming needed for aspects of the dashboard. There were not major alternatives considered because both of the libraries are some of the most used for their respective features and seem to be best in class for their functions. This meant that the implementation of the libraries, that is how they are connected into the web application, is the only thing that would need to be in question, referenced to an external copy of the library or stored locally.

3.4.1 jQuery

For JavaScript development we chose to use jQuery, it's a library designed to decrease the time spent writing JavaScript code to be able to focus on bigger pieces of the project. The goal of using jQuery is to reduce the amount of time the team needs to spend on writing any form of helper function and should increase the overall productivity of the team on the project.

3.4.2 Bootstrap

For CSS formatting we chose to use Bootstrap, the most popular CSS framework. It contains a multitude of styles to be applied across the dashboard to keep an overall theme. The implementation of these styles should drastically save time since the team will not have to create a large swath of CSS classes for the elements we will be using.

3.4.3 Deployment Method

From the team’s prior experience with externally referenced libraries, updates can occasionally break the functionality of pieces of the web application which results in a decent amount of unnecessary refactoring. Local libraries are a much safer option since they will never change how they work, this protects against the updating issue and there typically are not massive overhauls done to such long standing technologies that will cause a deprecation of the functionality. Our chosen approach is to implement the libraries locally so new updates do not have a chance of breaking the application. The only feasibility that will need to be considered is integrating these technologies onto a basic Django web application and testing that they function as intended.

4. Technology Integration

This section will cover how our chosen technologies will integrate together to create a solution that addresses the challenges of the project. In the diagram below, it showcases how the user will be able to utilize our dashboard and displays how the chosen technologies will work together to solve the clients problem.

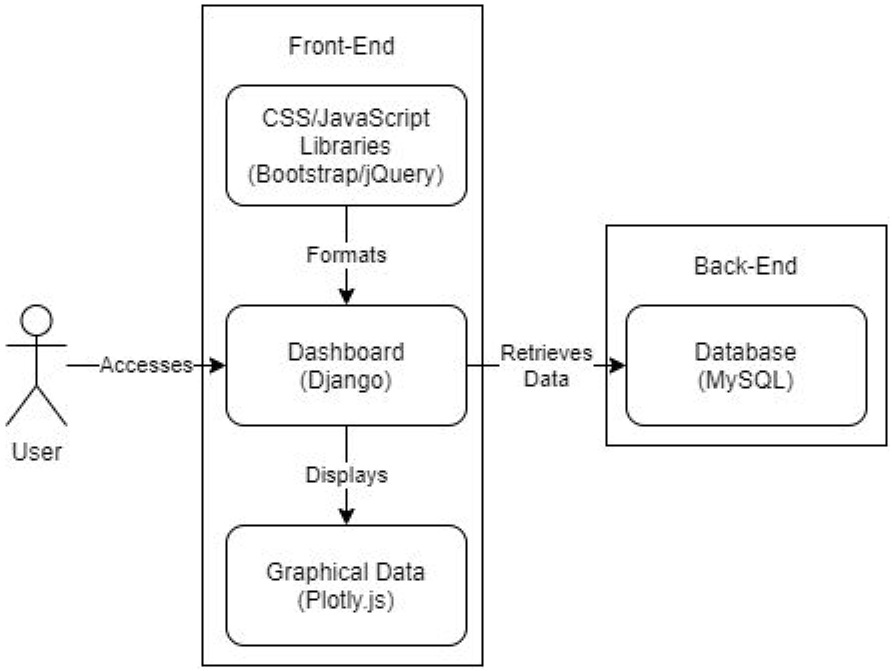

Figure 3: A diagram showing how the chosen technologies will integrate together

As shown in figure 3, when the user accesses the dashboard, they will be viewing the front-end web pages constructed through the web framework Django. For security purposes, the dashboard will be stored on a local machine, which will be located on site for the client. Then when the user loads the dashboard it will pull stored data from the back-end database that will be built in MySQL. Then after pulling the data from the database, the dashboard will display informational graphs about the NPOI tools and observational data through the use of our chosen graphing library, Plotly.js. For sake the usability, the dashboard and displayed data will all be formatted by a CSS library to help organize the page. Also, to help make the graphs and pages more interactive and dynamic, the dashboard will use a JavaScript library, jQuery. With all of these technologies working together, the dashboard will display simple but highly informative graphs to help monitor the delicate instrumentation at the NPOI and easily parse through collected observational data.

5. Conclusion

The goal of this project is to help the clients at NPOI build a web-based dashboard that will allow them to view and monitor instrument data of the tools working to collect observational data. Currently, the client at NPOI has no easy or viable method to view the health and status of the various tools working in unison on site. Monitoring each of the components is critical because if one of the tools is even slightly out of calibration, it can lead to errors in the observed data which defeats the purpose of collecting the data in the first place. Team Sirius hopes to build a solution to address this critical issue.

The team has researched and selected a number of technologies to implement this web-based dashboard for NPOI. The team has considered multiple factors when selecting these technologies such as security, longevity, integration compatibility, and learnability.

Technological Challenge	**Chosen Solution**
Web Framework	Django
Database Technology	MySQL
Graphing Library	Plotly.js

Figure 4: A table showing our selected technology for each technological challenge

To build a dashboard for NPOI, we have selected Django as our web framework which provides us with the integration capabilities and level of security required by Lowell observatory and the Naval Research Laboratory. For storing and retrieving the sensitive data from the NPOI, we are using MySQL for the database technology. Finally, to display and visualize the data in the dashboard we will be using Plotly.js as it has the ability to construct the custom diagrams we are looking to build.

With these selected technologies, we are confident that the team will be able to successfully build a web-based dashboard for the NPOI. This dashboard will be vital in helping ensure that the tools at NPOI stay properly calibrated and guarantee that collected observational data is accurate. This dashboard will streamline the process of monitoring the NPOI and help astronomers visualize the data collected.