Team Ceres

Final Report

April 28, 2021

Sponsored By: David Trilling, Michael Gowanlock
Team Mentor: Fabio Santos
Team Members: Javier Quintana, Joseph Sirna, Miles Barrios, Zach Messenger

Table of Contents

1. Introduction

The purpose of this document is to outline the process that Team Ceres has taken in the development of the Zwicky Transient Facility Asteroid Analysis Tool (ZAAT). This document will serve as the team's final report and will reiterate much of the development process that we have taken to build our client's product. The ZAAT will be explained in detail as you progress through the document and a better explanation of the team's timeline will be written out to provide a clear narrative of what has been done and what can still be done to improve the ZAAT.

Background

Every night astronomers across the globe participate in all-sky surveys, where the night sky is recorded in hopes to gain knowledge of the galactic entities that surround the Earth. These surveys can produce very large quantities of data and are useful for cataloging notable bodies, such as asteroids. The Zwicky Transient Facility (ZTF) in San Diego, California generates nearly 2 terabytes of data every night and that data alone is very difficult to examine due to the high rate that data is collected.

It is estimated that when the Vera C. Rubin Observatory is finished being built in 2021, 20 terabytes of data will be collected every night for the next 10 years. By the end of the Rubin Observatory's participation in these all-sky surveys, 73 petabytes of data will have been collected.

Our clients, Professor David Trilling and Professor Michael Gowanlock, are interested in using this data for their personal research but many of the interfaces that use data from the ZTF, are either outdated or not user friendly. The main problem that our clients faced however, is that they had no interface available to them with functionality that is user friendly and easy to navigate. Because of this, Professor Trilling and Professor Gowanlock were interested in the development of a new interface that uses data from the ZTF and provides easy to access data that doesn't overwhelm its users on first view.

Team Ceres goal was to create this graphical user interface and to create it with all of the requested accommodations to make it a valuable tool for our sponsors.

Current Standing

As it stands now, the team has developed a new asteroid analysis interface that will save our clients from having to access the ZTF data manually and will visualize the data for them. The interface has been named the ZTF Asteroid Analysis Tool and is in the form of a web application so users can easily access the information from anywhere with a computer and internet connection. The team was able to develop an interface with full search functionality and user account tools that made saving asteroids and queries easier. While there can still be improvements made, the team was able to secure all of the client requirements and leave them with a stylish fully functional application.

2. Process Overview

This section of the report focuses on the overall development process that the team utilized to create a successful capstone project. While the whole capstone class followed a development process most similar to the waterfall approach, our team within that overhanging waterfall approach followed the agile methodology.

Our development process most resembled the agile methodology because each meeting the team would have (team, mentor, and client) represented a different major component of the agile process. Our team meetings were the equivalent of standups/sprint planning because it would allow for members to voice their current objectives as well as retrospectives because at the beginning of each of our meetings, members would have the opportunity to voice concerns and good things that occurred over the previous week.

The mentor and client meetings were our showcases as we were constantly updating both mentor and client on the progress made on the web application. These meetings involved various activities such as live demos, discussion to further understand requirements, and time to resolve issues with the web application.

The team utilized a few pieces of software that allowed for the successful completion of this project. For version control, our team used git/github to manage the project and ensure that all members of the team were working on the version of the codebase.

In order to keep track of what objectives needed to be completed, the team used discord to keep track of what features were in progress and what still needed to be completed. Discord was also the main channel of communication for the team as it made both chatting and recording design reviews extremely easy.

The roles that the team divided remained the same throughout the lifespan of the project. Javier Quintana was the team lead, Zachary Messenger was the recorder, Joseph Sirna was the release manager, and Miles Barrios was the architect. These roles worked well and played a role in the reason for the team being successful.

3. Requirements

This section of the report details all of the specific technological requirements our software must include. These requirements have been gathered through a series of ongoing meetings with our clients and analyzing the project description provided to us at the start of the semester. This section of the report will detail the key components of the software we are planning to develop. Once all of these requirements have been met, we should have a working web application that satisfies all of the needs of our clients in order for them to further their research. Requirements fall into one of three categories:

- Functional Requirements
- Performance Requirements
- Environmental Requirements

3.1 Functional Requirements

Functional requirements represent the bulk of what our product needs to be able to do. These requirements listed will be included in the final product as the base functionalities for our web application. This section represents the core components we have agreed upon implementing throughout our meetings with our clients this first semester.

3.1.1 Filter/Display Data

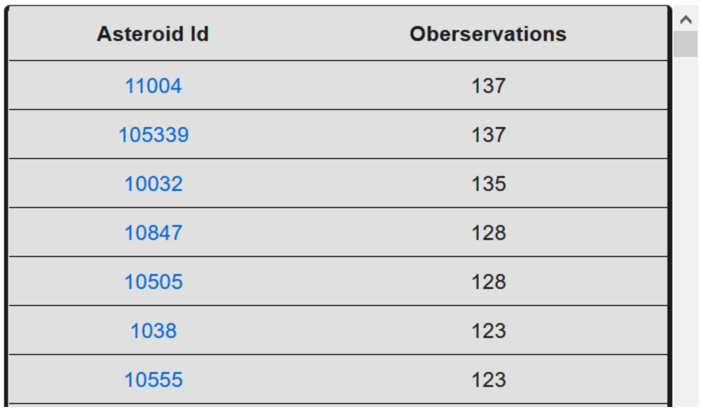
Our clients emphasized the importance of not putting off anyone not well-versed in the field of astronomy when seeing the landing page. Other similar analysis visualization tools suffer from “information overload” when someone arrives at their front pages. Our solution aims to minimize this effect by concisely displaying interesting information in an easy to understand format. This feature can be broken down into smaller components that represent this functional requirement when used together, these smaller components include filtering, selecting filtered datasets to be displayed, rendering graphs that represent the filtered and selected data

Selecting filtered datasets to be displayed

After the user is returned the filtered data that they requested, the user should be able to choose specific sets of data or ranges that are going to be displayed in various formats (table, graph, etc). This select feature needs to be user friendly and allow for all types of data points. This feature will be a big help to various researchers and users that are creating a variety of displays.

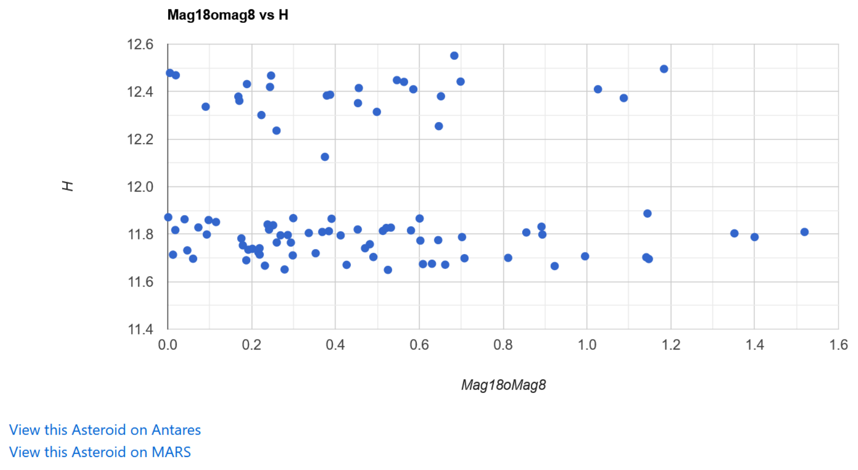
Rendering graphs that represent the filtered and selected data

After filtering and selecting data, the user will need to be able to render graphs based on the data provided. This will be done by providing Google Charts with the selected data and providing instructions on what type of graph and how it should display the data. One way to suggest a large amount of data exists without displaying it all in an overwhelming way is to rank interesting and/or well-observed asteroids in a small table, while large visualizations about interesting characteristics expressed in layman's terms take center-stage on the front page. Clicking one of these asteroids on a graph will take the user to a similar detail page. This should create an interesting and interactive experience for the average user.



Asteroid Id	Oberservations
11004	137
105339	137
10032	135
10847	128
10505	128
1038	123
10555	123

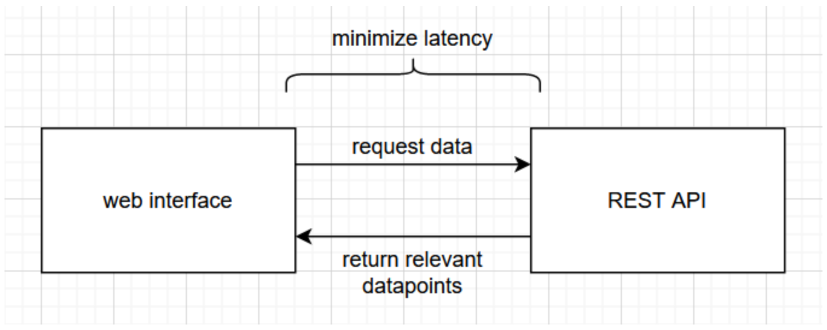
Example: a small table, sorted by observation count.



Example: clicking an asteroid Id will take the user to a page with more details.

Filtering

The ability to filter and search through large amounts of data is crucial to the core functionality of this solution. This filtering section will allow for the user to choose desired characteristics and properties of an asteroid that they require. The interactions between the web interface and the REST API must keep latency to a minimum. In addition, the REST API must be able to quickly and efficiently evaluate what data points match the constraints provided via the filtering component.



3.1.2 Export Data and Graphs

In meeting with our clients, it has been deemed necessary that users are able to easily save data and graphs to their own machines. This feature allows users to come back to data they have been analyzing previously, load datasets and figures into other softwares, or present their analysis alongside their research. On top of this, if an interesting phenomenon is observed, the astronomer/researcher will likely want to save the data/figures they found. Having the ability to download data and figures means the user does not need to recreate either the queries or figures previously generated.

Exporting

Users will have the ability to export/download files to their machines in the following formats:

- CSV
- PNG
- PDF

Data

In our web application, the user will be able to search for asteroids using filters to view specific characteristics as described in the above requirement. This will present the user with a subset of the larger database, and the user will have the ability to download this data as described by the above requirement. The user should be able to download it in the above formats.

Graphs

Similar to data, users will have the ability to download figures such as graphs created before and after applying specific analysis tools. Any of the graphs that are able to be created as described in the previous section will be able to be downloaded to the users local machine in the above mentioned formats.

3.1.3 Create User Account

Our client has shared, both verbally and written that the ability to create user accounts and set preferences is a key requirement for the web application.In addition, this would allow for any researchers using this software to modify their preferences which might relate to specific filtering or custom queries on the asteroid database. This functional requirement can be broken down into the following smaller requirements:

Account Creation/Login

The implementation of user roles and preferences relies on the creation of a create account/log in feature in the application. This login feature would be similar to that of any other web page where anyone could create an account. However, roles would be assigned by someone with the status of admin. These accounts that have roles lesser than admin would have the ability to perform certain tasks such as viewing/exporting graphs, data, and other visualizations.

User Settings/Preferences

Having a user settings feature will be essential to how users are able to interact with the web application. The people using our application are researchers attempting to pull different types of information regarding the asteroids, thus having a way for them to adjust their experience using the web application should be crucial. These settings could include preferences such as the maximum data points per graph, default export format (CSV, PNG, xslx), bookmarked asteroids/observations, and the ability to limit the number of datapoints returned from a query to name a few.

Saved Work Section

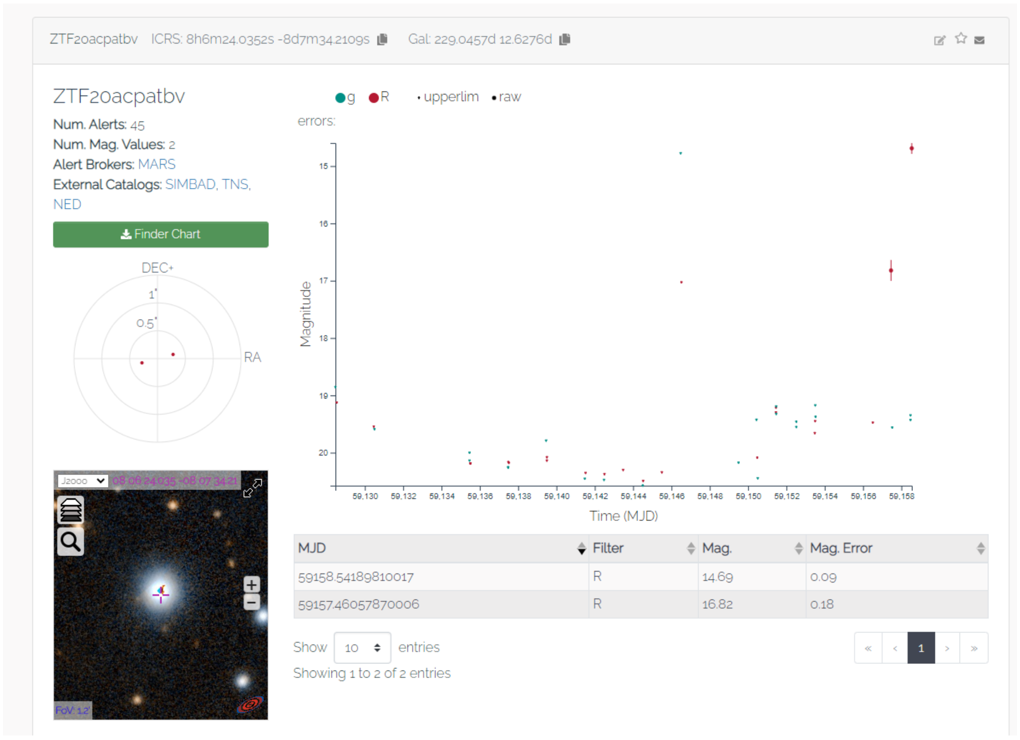
As research can take longer than a single sitting to complete, the ability for a user to save their work becomes crucial to creating an application that meets the needs of the client. This saved work section could include various types of visualizations, data tables, and common queries. This section would use the share data analytics feature mentioned below because allowing users to save their work would make sharing results easier and faster.

3.1.4 Share Data Analytics with Other Users

Another valuable feature that was requested from our clients was the ability to share data analytics with other users. The reason this is a functional requirement is because the intention our clients have is to share this data with other users across the country. Accompanied with sharing, we want to also provide links to other existing databases for asteroids in order to provide more resources readily available.

Sharing

The graphical user interface we are creating will have the option to share each asteroid's metrics with other users via email or web URL in order to ensure that the end user has access to the live data in case of any changes or developments. This will allow easy collaboration amongst peers and researchers.


ZTF20acpatbv ICRS: 8h6m24.0352s -8d7m34.2109s Gal: 229.0457d 12.6276d

ZTF20acpatbv

Num. Alerts: 45
Num. Mag. Values: 2
Alert Brokers: MARS
External Catalogs: SIMBAD, TNS, NED

MJD	Filter	Mag.	Mag. Error
59158.54189810017	R	14.69	0.09
59157.46057870006	R	16.82	0.18

Show 10 entries

Showing 1 to 2 of 2 entries

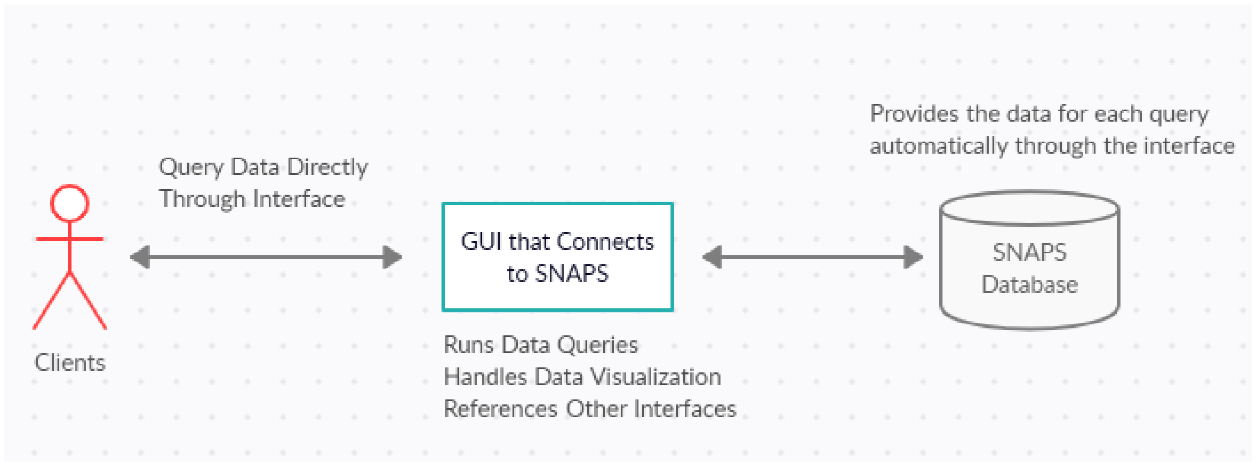
The implementation of making these resources available in our application will be done by using simple linking to the same asteroid in the other database. We were able to study the other databases to create a hyperlink that directly queries the needed asteroid.

3.1.5 Database Import Job

Another functional requirement that the team found as a major one was the database import job. In order to ensure that the application is using up to date data, the team will run database import jobs to bring in any new information introduced to the database. These import jobs need to be performed at ideal times when web application use is little to none. This will ensure that no users are interrupted in the middle of work. It is important that our application is able to receive import jobs for new data, or else the web application would require manual updates which would defeat the purpose of our application.

4. Architecture and Implementation

Architecture


Clients
Query Data Directly Through Interface
GUI that Connects to SNAPS
Runs Data Queries
Handles Data Visualization
References Other Interfaces
Provides the data for each query automatically through the interface
SNAPS Database

As seen in the diagram below, our app consists of two main components. These being the frontend user interface and the backend database. Our user interface is able to interact with the database through a rest api. The rest api will send data requests from the frontend to the database, and provide the data back to the frontend. This rest api basically acts as a middleman between the database and our web application.

When viewed in this aspect, the role of each component can very clearly be seen. The database is in charge of keeping our data accurate and consistent. This will be done through regularly scheduled batch jobs that are to import new datasets without disrupting the user's experience. The database will also be answering requests from the rest api in order to present data to our web application.

The rest api is very simply put, a middle man for our product. The rest api will receive requests from the frontend/user and then send these requests to the database or answer from cached queries.

The frontend application will be the user's main interaction with our product. This component will be visually appealing to the users and responsive with all the actions users might perform. The frontend will display data to the user, manage the user's authentication status to save personalized bookmarks and preferences, and allow for searching/querying of the database.

Overall, the data flow pattern of our application can best be seen through describing a use case of our application. Suppose a user is viewing asteroid data and wants to search for asteroids with a certain trait, let’s say asteroids that have above 20 observations in our database. This would mean that we have observed this asteroid enough for the user to want to see the different metrics collected over a series of observations in order to perform some form of analysis. Our user would start by going to the search page in our application, applying the appropriate filters, and sending out the request. This is where the data flow pattern of our application comes into play.

Our application starts by calling a frontend function to handle gathering the various search parameters the user provided. These parameters are then used to create a get/post request that will get the desired asteroids from our database. The frontend function passes the request to our API, which in turn is able to execute the actual query upon the database. From here, the database returns the results from the query: either a list of asteroids with the desired traits or an empty list. These results are passed to the API, which in turn sends data back to the frontend of the application. The frontend is able to parse the data received into an easily readable form for the user, and then displays these results.

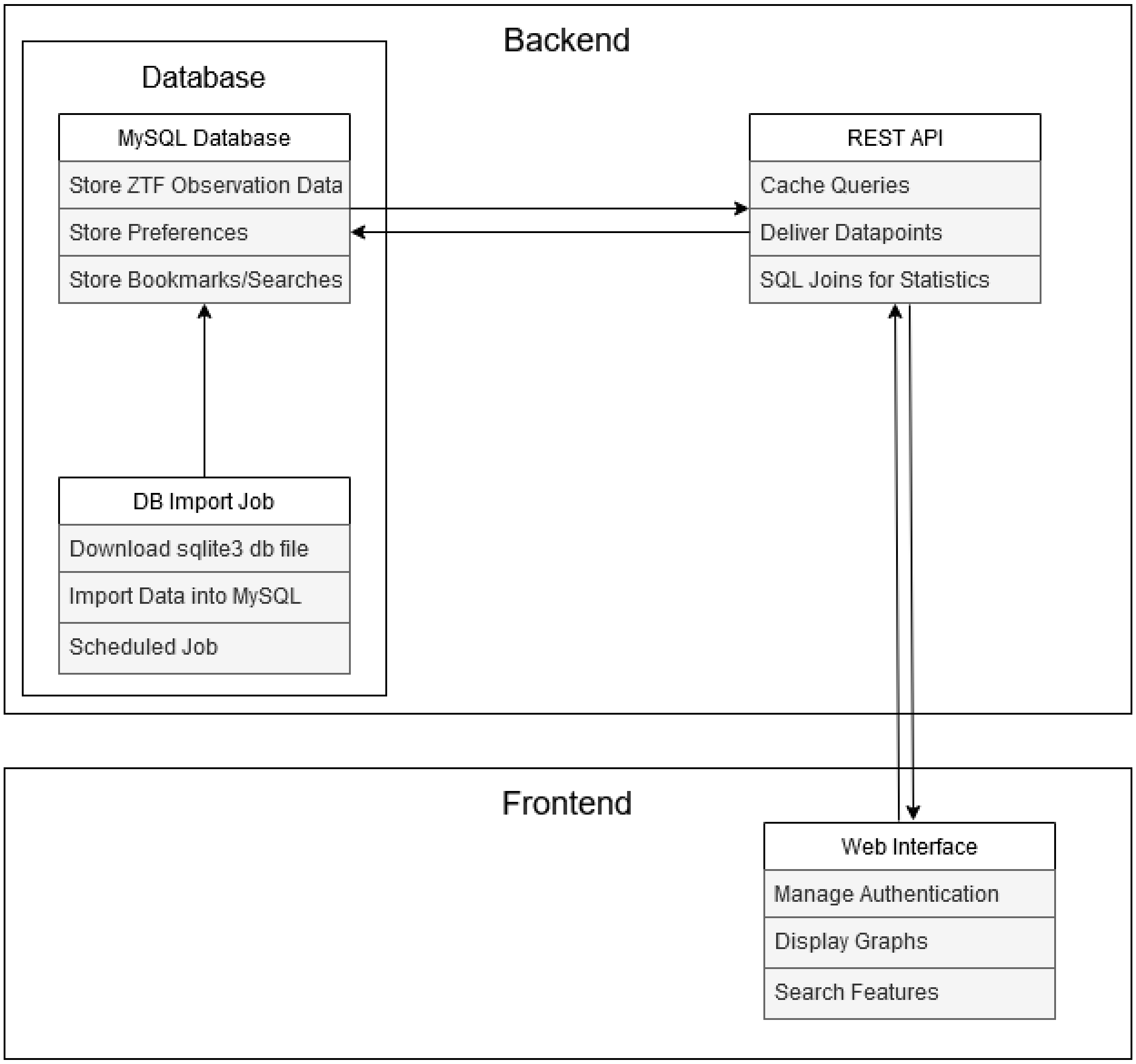


Figure 1: Architectural Layout of the Interface

Implementation

In order to implement the ZTF Asteroid Analysis Tool (ZAAT), there are two main components. We want to start by creating an interactive web application that allows for users to easily view, search, and analyze asteroid data. The second component we want is a back-end database and api to serve data from our database to the front-end interface. These two components will give the user all the functionality required for our product, and allow for easy upgrades and improvements to the interface without affecting the ability of the application to run. Although these are the two main components of our application, there was some breakdown of these technologies in order to allow for them to operate as optimally as possible.

The chosen technologies for this project are broken down below:

1. ZTF Asteroid Analysis Tool (ZAAT)
    a. Front-End React Application
        i. Various styling frameworks to allow for an excellent user experience
        ii. FirebaseJS to allow for user authentication, roles, and personalized preferences and bookmarks
    b. Back-End Database
        i. A live hosted database that is updated regularly to keep data accurate and consistent
        ii. An api to service requests to the database and send them back out to the front-end interface

This is a very general overview of the implementation of our product, and throughout this document we will further develop the concepts of our designs to formulate a blueprint for ZAAT. This should allow for incoming developers to understand our thought process with our designs and understand how the tool was built.

5. Testing

Unit Testing

The purpose for unit testing the ZAAT interface is to ensure that various specifications of the final product are functioning as intended and not performing any more or any fewer tasks than they should be. It is important to ensure that each component of the frontend and backend of this application is doing just what it is expected to do.

Especially because it is anticipated that multiple users will be potentially using the ZAAT interface at once so optimal performance is key to ensure functionality.

As mentioned previously, the unit testing of our application will be broken down into two main components of testing. The first component is the frontend of the application, which handles the user interactions and views. The second component is the backend of the application, which is responsible for managing the data being delivered to the frontend and holding user information privately.

Frontend Unit Testing

Our main user interface is built completely in React, which is a JS framework that uses node.js to run applications. Although we have been testing the functionalities of our application as we have been building it and making sure everything operates as we want it to, we're going to build more structured tests that we will be able to use for testing a larger chunk of our javascript code much more efficiently. Going along with this, we have laid out several test cases that will comprise our unit testing of the frontend.

> Note: For each of our cases we will be using Mocha.js or Jest.js as they interface with React and Node.js well for testing purposes.

<u>Test Case 1</u>: Performing a Query

One key requirement of our application is being able to perform queries on a database of asteroids based on the desired characteristics provided from our users. Without this key feature, our application would virtually not be doing anything. This is why it is important to test this functionality to ensure it works to the best of our ability.

Overall, this test should be pretty straightforward. What we need to ensure here is that our frontend can validate user provided search parameters, then using these parameters construct our query into the proper format. Once the user enters the desired search criteria, the application will validate the data entered, construct a query, and connect with the rest api. Depending on the type of query (i.e., insert vs select), we will also want to handle awaiting a response vs not receiving one.

One key problem to watch out for here improperly formatted queries and not being able to connect with the database. If either of these problems were to occur, it would block the entire application from running since we really rely on this asteroid data.

Test Case 2: Viewing Specific Data Relating to an Asteroid
Another key requirement is that we want our application to be able to access the data for individual asteroid pages. This is the bread and butter of our software, because once users have been able to query the database and get back the results they desire, it is all for naught unless they are able to view this interesting data that drew them to said asteroids.

The first component of this test would be covered by Test 1, and that is being able to query the database. From here, we can find an interesting asteroid and access its page, which pulls the data about the asteroid from the database once again. This test case should be pretty straightforward and once we validate the first test case, this one will be a piece of cake.

Test Case 3: Accessing and Saving Data to User Profiles
This case follows very similarly from the previous two cases, but will interact with new tables in our database that host information relating to the user. The first step of this test will be determining if our frontend is able to interact with our backend. Once this is verified, we'll have to test insert and select statements on this new table with user information.

This data will be relating to user bookmarks, display name, and search result preferences. All of this information will be secure and only accessible if the authentication service, Firebase, detects that a current user is signed in. One key problem to watch out for here is making sure that our queries cannot be broken/tampered with in any way that would allow unauthorized users to obtain data not relating to their user profile.

Backend Unit Testing

On the backend of our application, we have our database and REST API which are doing the heavy lifting when it comes to data storage and retrieval for our application. Again, we have spent a lot of time working with our application functioning as a whole in order to test how the backend is able to handle edge cases. We have still come up with a number of test cases to construct and work through to ensure that our application is avoiding all critical problems.

Since the REST API is written in C# .NET, the logical choice of unit test library is MSTest. We will write several unit tests using MSTest for each core functionality of the backend. These unit tests will be focused solely on internal consistency of method returns from a backend-only perspective; ensuring that non-nullable fields are, in fact, not null, for example.

<u>Test Case 1</u>: Retrieval of a Large Number of Asteroids from MySQL
One of the main focuses of the backend of the application is the ability to serve large amounts of data to the frontend. As a prerequisite, however, it must first be able to query large amounts of data from the database, which it then serializes. This unit tests involves querying every asteroid in the `asteroid` table from MySQL, as well as populating associated observations, and checking the results for anything that failed to populate.

<u>Test Case 2</u>: Updating User Preferences
The API is also responsible for being the intermediary between the frontend and the database in the context of update queries, not just data retrieval. A good example of this communication occurring in the opposite direction is the ability for a user to update their preferences (stored in SQL). In order to emulate this functionality, this unit test will call the method to update user preferences, not in the form of a post request, but a local call

in C#. The result will then be checked before and after the update to ensure that all user preferences (results per page, etc) are able to be updated individually.

Test Case 3: Registering / Deleting a User
This unit test will ensure that the functionality to register a user / delete a user works as intended. This will be accomplished as above, by calling the public-facing API method with C# and checking the results in the database before and after each creation/deletion operation on an auto generated test user.

Integration Testing

The purpose of integration testing is to take the tests done in unit testing and stack these various pieces on top of each other to make sure that everything still works as expected. When performing integration testing on a component-based front-end such as this project, the testing can be a bit tricky. The approach that Team Ceres has decided to take with integration testing is to test the smaller components individually first and begin to stack these components on top of each other to generate the overall larger components that make up the web application.

The major modules to be tested in the code include the overall app component which embodies the web application as a whole as well as the following pages (components): **Home**, **Search**, **Bookmarks**, and **Account**. These four major pages represent a high-level overview of the components that make up our front-end. Each of these pages is made up of multiple smaller components and/or components that lead to a smaller page. The plan to perform integration testing on a single-page application is to use the popular testing library for React applications known as Jest. Since React uses the state to handle variables and data, we will be able to perform integration testing on components by having components make the various state changes and API calls needed. We will figure out what data should be expected to be in that state and test to see if that data matches when the component is loaded.

The app component (overall web application) gets broken down into the 4 components mentioned above. The app component will be the final piece of the integration testing that gets tested as it is made up of a large amount of smaller components. While the app component will be a little harder to test since there is a lack of state, the current plan is to assume that by testing smaller components, we can assume that the app component works if it is built off of other tested components. In addition, the team can visually confirm if the component is rendered and acts correctly.

The Home component contains multiple histogram components that are generated using data retrieved from the database via an API call. These histogram components utilize a node package called Google Charts. This library allows for data to be sent to a component and then the corresponding component is generated and returned. In order to test our histogram component, we will make a call to the API to know what data should be returned, we will then use JEST to ensure that the data returned matches what we expected. This can be repeated for all the histogram components generated on the home page. By verifying these smaller components, we are able to verify that the home page is correctly generated.

The Search component contains a search filter column that is responsible for providing data to the frontend via a similar API call to the one used in the Home component. Once a call has been made retrieving the current stored asteroid data, it will be displayed in a sorted order (either ascending or descending based on the filter choice) based on the value. Since data is being transferred we can test this component like we do the Home component and use JEST to verify that the data we are querying is what we should be expecting. To be even more thorough, we can test each property filter in the search criteria. This will allow us to also verify that each and every value present in the query is accurate.

The Bookmark component is currently responsible for maintaining the list of favored user asteroids and has the very simple functionality of displaying a link to an asteroid data graph. While this could be tested with an automatic library, the team decided it may be best to manually test this component as it will be obvious which data needs to be delivered. Testing this component will be as simple as adding specific asteroids to our list of bookmarks, and then verifying that you can access that asteroid. The main reason being that this component does not require much digging to determine if it works.

The Account component works the most with Firebase, as Firebase is responsible for maintaining the user data and preferences. Since Firebase is responsible for the data we are using, we can test the user data storage similar to how we are able to test data provided to us from the home page using Jest. The team is already able to see who has created an account in the interface so the testing would not be as long due to the limited amount of accounts we have already. Testing would simply consist of verifying that a freshly created account matches the expected data we put in.

Usability Testing

The purpose of usability testing our interface is to ensure that the tools we have created are easily understandable and can be used by people with various degrees of technological knowledge. Testing with outside users who are not familiar with our program and have not been following along with its development makes them the perfect demographic for ensuring the usability of our program. While the users of our application will only be using the frontend to evaluate asteroid data, there are still plenty of tools on the frontend that we have created and want tested by users.

Below we have created a list of the actions and tools we want users to test during the usability trials:

User Actions

1. Navigate to the ZAAT Interface - Dependent on where the interface is hosted.
2. Navigate to the *Search* page, *Bookmarks* page, an individual *Asteroid's* page, and the *Account* page.
3. Create an account on the interface and log in.

4. Search for an asteroid by any property, then search for an asteroid by name.
5. Bookmark any asteroids then navigate to where that bookmark is located.
6. Go to an asteroid's page from the *Bookmarks* page.
7. Change the X and Y axes on the asteroid graph.
8. Download the asteroid graph and then view it on either the Antares or MARS interface web page.
9. Go back to any saved bookmarks and remove them.
10. Delete the user account being used.

The list we have created for testing has been written as such because it will ensure a number of things as the user processes the instructions. The instructions will first take the user through every single page our interface currently has to offer so they can be aware of what these pages look like for the following steps. After the user has become familiar with the location of these various pages, they will create an account so they can test the functionalities we have implemented that require a user to have an account. As such, they will continue by running queries on the interface and save bookmarks for later testing. After searching and navigating through individual pages of asteroids, the user will test undoing the bookmarks they just created and will finish off by deleting the account they have just created.

All of these instructions will expose the user to all of the functionality currently installed on the interface and have been designated as our baseline instructions to know that a user understands how to properly use our interface. And while we acknowledge that everything the user does is not the entirety of the functionality we have built-in, the important thing that the team is mainly concerned with is a user's ability to find these tools on their own. Once we have had users test these components, we can look for tester feedback to improve the interface.

6. Project Timeline

6.1. Semester 1 Timeline

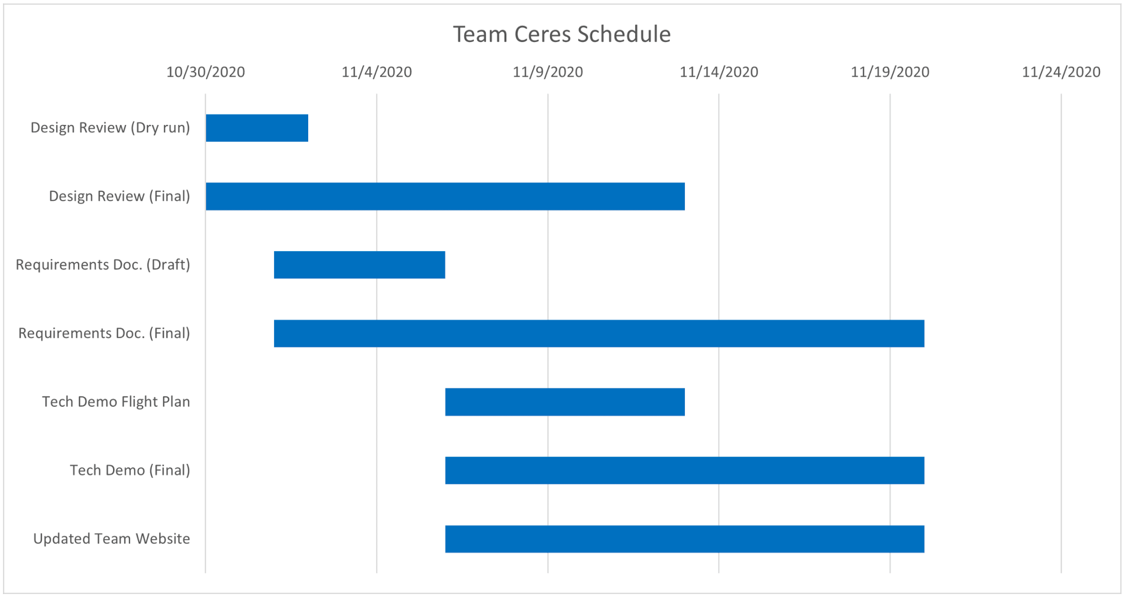
During our first semester, we spent a lot of time gathering requirements and making sure that we had a good understanding of the vision that our clients had in mind when proposing the project. From here, we began production on our documentation and application in early October with our first due date being in mid November.

To start off with our project, we created a design review presentation that showed we efficiently and effectively communicated with our clients to understand their requirements for the project. From here, we then turned around and put these requirements into a more official document that related the required features of our application to the technologies we chose to use for our building the application for our clients.

Once all of these requirements were gathered and set into stone, we began to produce a technical demonstration to show that our application would be able to get the job done with the technologies we had chosen. This was the most fun we had as a team during the first semester. It was an amazing experience to actually get to build out the application as a team and put code together that worked for the exact purposes our clients had in mind.

From here, we moved into the second semester where we focussed very much on finalizing the application, working out any weird kinks, and making sure that all features were working as desired.

A detailed rundown of the key tasks we accomplished can be found in the chart below.



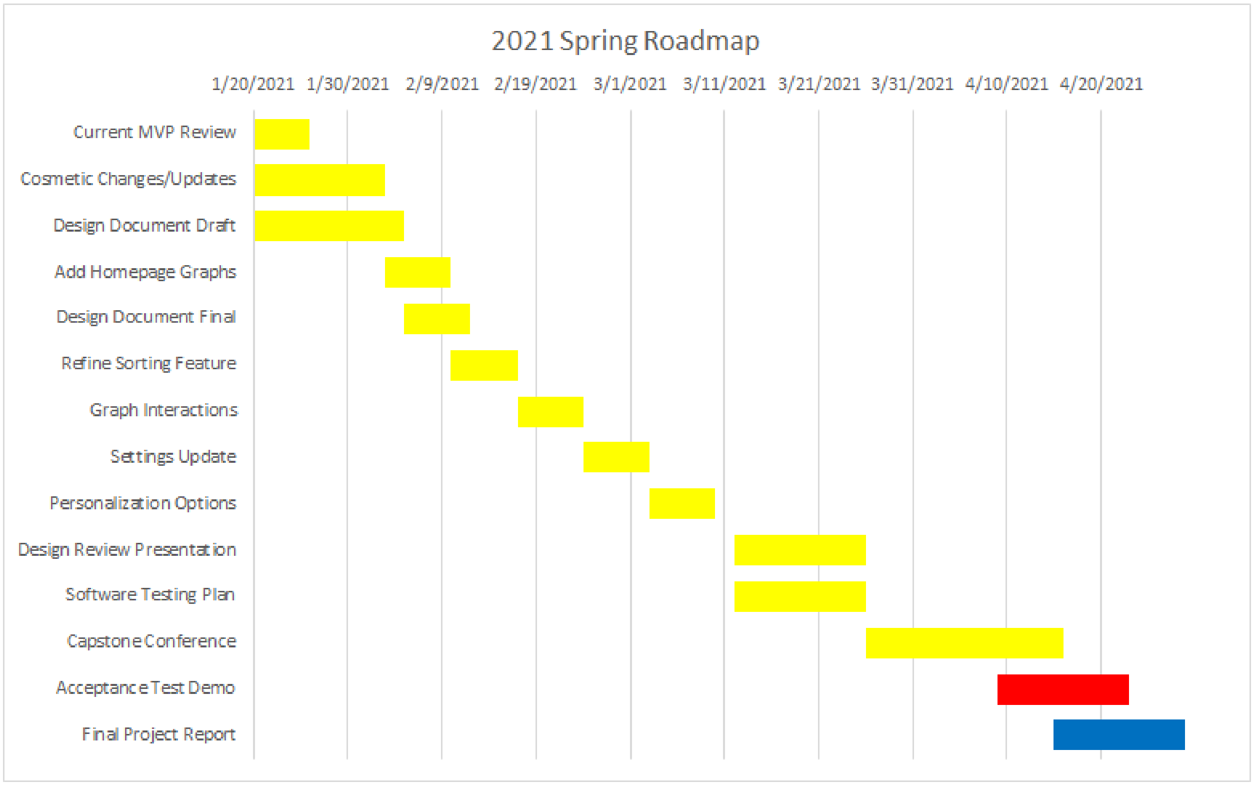
6.2. Semester 2 Timeline

In our second semester, we had a lot more focus on developing our application and documenting how exactly we were developing it so that it would continue to be usable and scalable long after we leave and hand off the project.

To start the semester, we reviewed the MVP that we were planning to build for the client and the features on top of this MVP that would make our project stand out from the rest of the capstone projects this year. Initially, we had a very great foundation for our application from last semester’s technological demonstration. All of the key components of our application were already integrated within one another, and from here we just had to build out more features. This included making desired aesthetic changes to the application, adding more graphing features, personalization updates, etc…

On top of this, we had to prepare a couple of documents to record our design process with our application and create presentations to commemorate our great work.

A detailed rundown of the tasks we accomplished can be found in the chart below.



7. Future Work

As far as future work goes, the application is still something that can be improved over time. To start the querying of asteroid data could also incorporate the use of mathematics or multiple metrics and features. There is the potential for users who are not signed in or using an account to have bookmarks saved using cookies or browser settings. Additionally the database could be expanded to include data about other extra terrestrial bodies besides just asteroids, such as stars or planets. A final key example would be to make modifications to some of the google charts sylings or override these stylings to improve readability and scalability of the charts within the application.

These are just a few small ideas that we came up with our clients in some of our final meetings and some ideas that we heard would be nice throughout the semester. We believe that we have provided ample documentation and insight into our design process so that future developers on this project will be able to easily understand our ideas and implementation process in order to continue developing this application into something that can help astronomers all around the world.

8. Conclusion

The big data revolution is coming to astronomy, and there are very few computational methods for analyzing this data as it currently stands. The knowledge to be gained from this data could lead to many powerful discoveries, but most importantly it could help us to prepare for the inevitability of an asteroid impacting Earth. So Team Ceres plans to address this problem. We want to build a scalable, responsive web application in order to aid in this revolution. We plan to use a comprehensive data visualization framework, user authentication, and importing data to a web-accessible database system in order to reach our goals.

We spent a good deal of time researching and testing the feasibility of the various technologies that we used for this project, and we feel confident in the decisions that were made along the way. The finished application that team Ceres delivered to the client met all of the requirements specified back in the fall semester. Some of the major features that should be noted include:

- Fully functional search page
- Bookmarks page that is linked to user accounts
- User Settings page
- Efficient Rest API to deliver data from database to front-end

As discussed above, this project was created to allow for Professor Trilling and Professor Gowanlock to be able to quickly and effectively gather information on the asteroid data being transferred from the ZTF. Our team achieved just that by creating a responsive and reliable front-end which was connected to the database using a REST API which allowed for fast transfer of data. Due to the application, our client will no longer have a need to go to their database guy as they can now easily perform database queries on their own and receive the data and graphs they want quickly.

During the semester, our client discussed that this application was not just for them as they work with a number of other researchers who would benefit from this application as well. The web application is hosted on AWS which makes for universal access for all. Should another team pick up this application in the future, this GUI could be useful for a very large audience.

Overall, the team performed extremely well and was able to provide a complete web application to our client. The trick to this was communicating effectively and having trust that each other would complete the assigned task on time. The utilization of the agile methodology proved to be a key part to the success as requirements are constantly changing. This whole year of capstone provided the team with an opportunity to grow significantly as developers as we look to enter the professional world of software development.

Glossary

All Sky Survey. A general map or image of a region of the sky that lacks a specific observational target.

Asteroid. a small rocky body orbiting the sun. Large numbers of these, ranging in size from nearly 600 miles (1,000 km) across (Ceres) to dust particles, are found (as the asteroid belt) especially between the orbits of Mars and Jupiter, though some have more eccentric orbits, and a few pass close to the earth or enter the atmosphere as meteors.

Google Charts. A graphing library developed by Google to help visualize data with various graph types and interactions.

Query. To search or browse through data, in this instance, asteroid data.

REST API. An application programming interface (API or web API) that conforms to the constraints of REST architectural style and allows for interaction with RESTful web services. If you want to interact with a computer or system to retrieve information or perform a function, an API helps you communicate what you want to that system so it can understand and fulfill the request.

ZAAT. ZTF Asteroid Analysis Tool, the name of the interface Team Ceres has created that helps visualize data from the Zwicky Transient Facility.

ZTF. Zwicky Transient Facility, the origin of the database where the ZAAT pulls asteroid data from.

Appendix A

Hardware

The **ZAAT** Interface is a web application that is currently being hosted on an instance of an Amazon Web Services server, currently being paid for by our clients. The server has a total of 4 gigabytes of memory, 80 gigabytes of storage, an Intel Xeon E5-2686 CPU and uses AMD64 architecture. More specifically, the server is Amazon's t2.medium plan and uses Windows 2019 Server to host all of the information for our web application.

Hardware Requirements

After an excessive amount of testing and research, the team has determined that there are minimum server requirements to have this web application properly hosted. In order to complete processes and data transfer, the team has found a requirement of 4 GB of memory on the server as it has to deal with the bloatware on Windows and handle live web operations. Along with this, the team also suggests at least 2 CPU cores on the server to get the tasks completed in a fair amount of time. Any less presented more issues with functionality and made the application very difficult to use. Finally, the web application needs at least 60 GB of storage space to hold the data from the ZTF that is being sent to the web application. This ensures that there is enough space for the data on there and will allow some room for new data to be held while old information is cycled out.

Toolchain

In this section of the appendix, we will outline all of the actual software tools we used in developing this application. As far as environments go, we did not use any specific development IDE other than each person's preferred text editor. There was no real need for everyone to jump on board with a single IDE for this application because we established some good and practical coding practices to keep our codebase clean.

On top of this though, we did have several useful tools to the development cycle we believe are notable. Starting with the frontend component of our application, we built our web interface using React JS and Node.js. These components were very nice when sharing code around our team because Node.js has a built in Node Package Manager (NPM) which was able to track and install new packages each person added to our JS libraries. With the combination of these two technologies, package management and installation was basically taken care of.

Moving onto the backend of our application, we used two key components for our application. A Rest API and a MySQL database instance. The Rest API was the middleman for our application that would receive data requests from the frontend of the application, perform queries on the database, and then return the results formatted using JSON to the frontend of our application. The MySQL database instance was used simply to host our data, support constant updates, and hold some notable user keys and information.

With all of the noted technologies above, we believe that our application should be able to run on any machine with the appropriate hardware specifications.

Setup

Required on windows server:

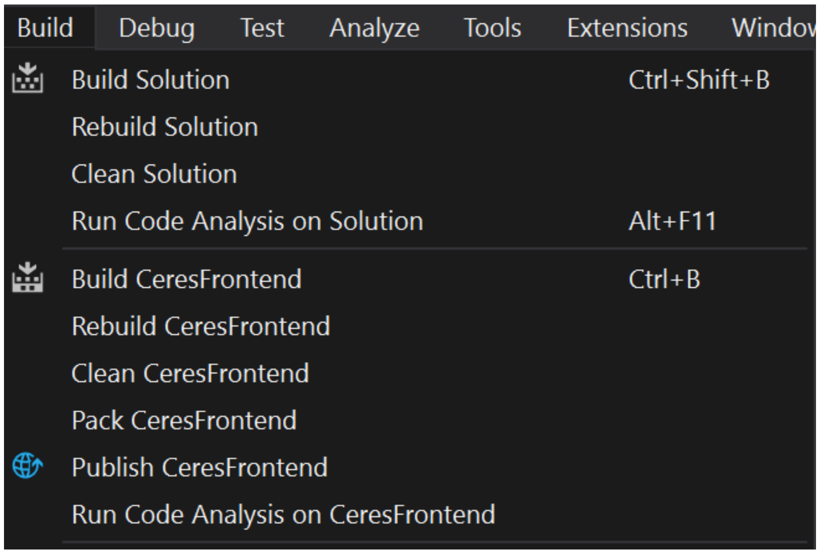
- .NET Core 3.1 Hosting Bundle
- Node.js
- MySQL Server
- IIS 7 / 8

If compiling from source code, Visual Studio 2019 is also required.

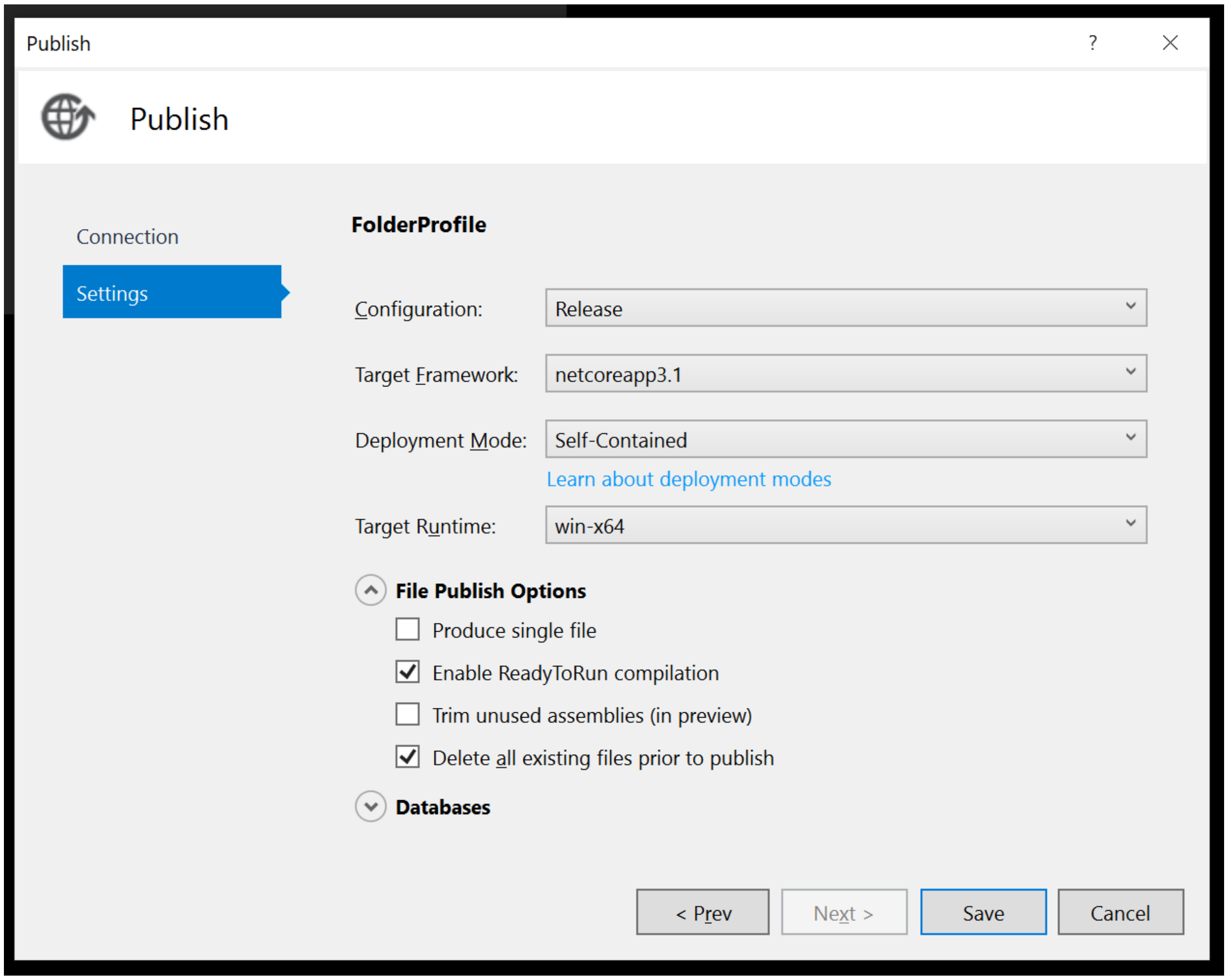
Building From Source

Visual Studio 2019 is required to build all 3 parts of the source code (import, api, and frontend). For all 3 projects, the initial build configuration is very similar.

Opening the Build menu from Visual Studio, click “Publish [project name]”



For the frontend, the build is configured as follows:



FolderProfile

Configuration: Release

Target Framework: netcoreapp3.1

Deployment Mode: Self-Contained

Learn about deployment modes

Target Runtime: win-x64

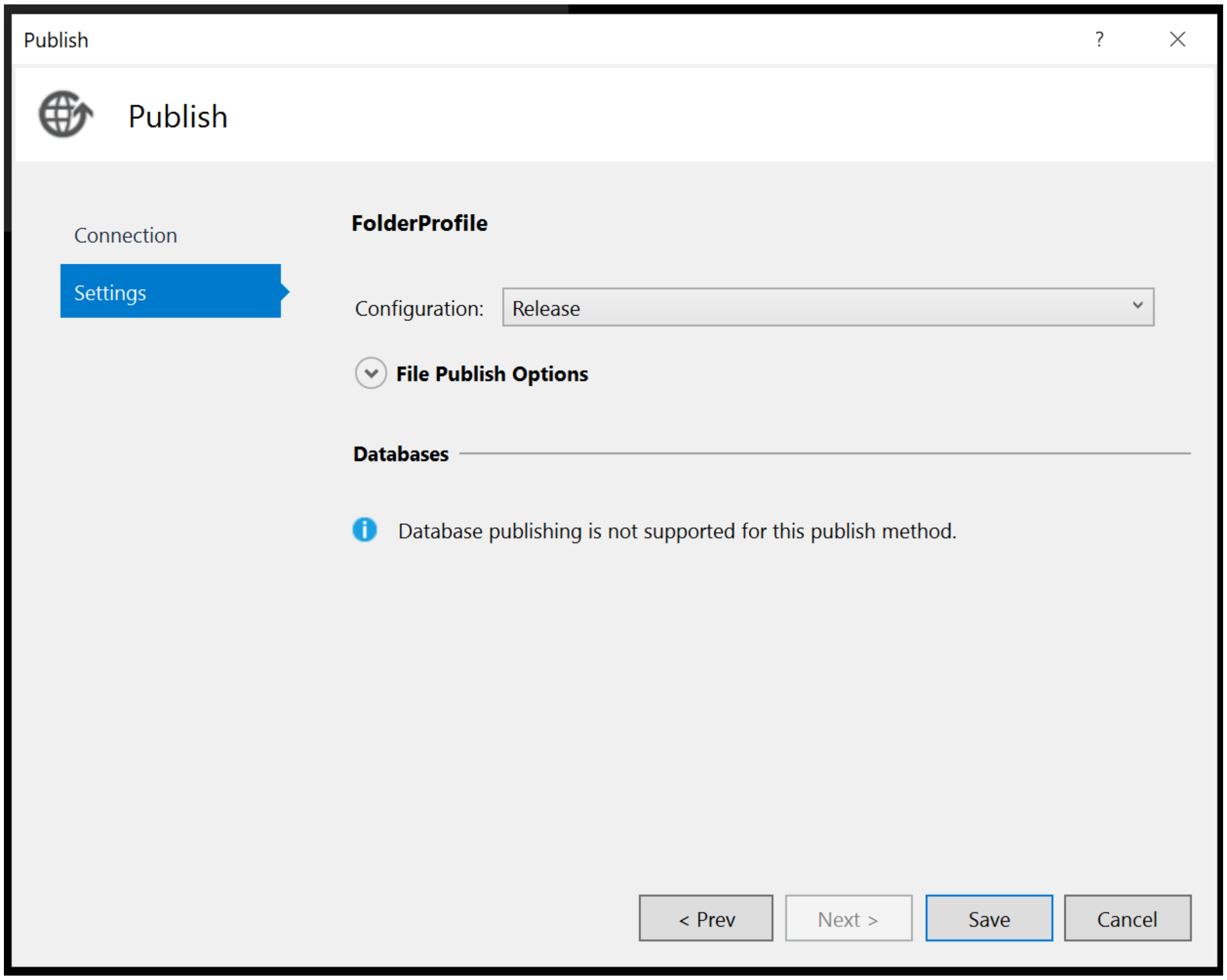
File Publish Options

- [ ] Produce single file
- [x] Enable ReadyToRun compilation
- [ ] Trim unused assemblies (in preview)
- [x] Delete all existing files prior to publish

Databases

For the database import tool, the build configuration is set up in the .sln file by default and should publish the .exe and any required Dlls to "<project location on disk>\bin\Release\netcoreapp3.1\publish\"

For the API, the build configuration looks as follows:



FolderProfile

Configuration: Release

File Publish Options

Databases

Database publishing is not supported for this publish method.

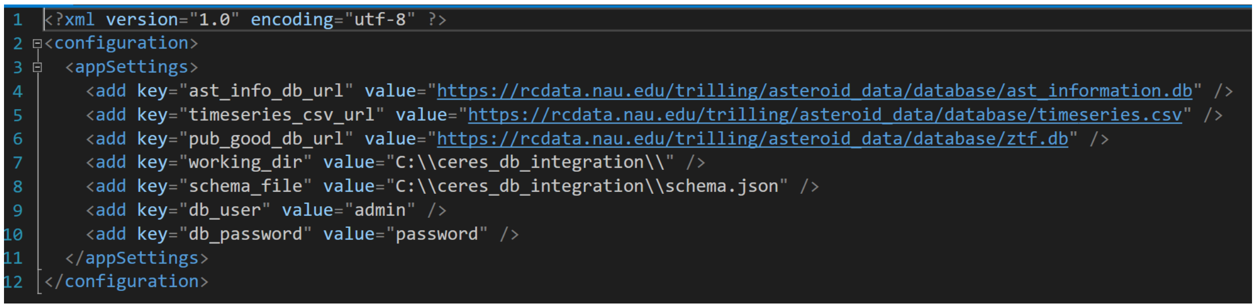
Setting up the Database

Once MySQL has been installed and an administrative user configured, the following tables must be created manually: `users` and `bookmarks`. The .sql file to create these tables is located on the Desktop on the hosted server instance provided, but also along with the database import job repository just in case.

Running the Import Tool

Once the MySQL database has been created and configured according to the above step, the 3 main tables can be imported: `ztf`, `timeseries`, and `asteroids`. These tables can be selectively imported by way of command line arguments supplied to the import utility executable, which are “-ztf”, “-timeseries”, and “-ast”, respectively.

Note that the import tool's schema configuration file must be changed if the schema of any of the corresponding .db or .csv files change. This file should be located in the specified directory within DBImport.dll.config (screenshot below).

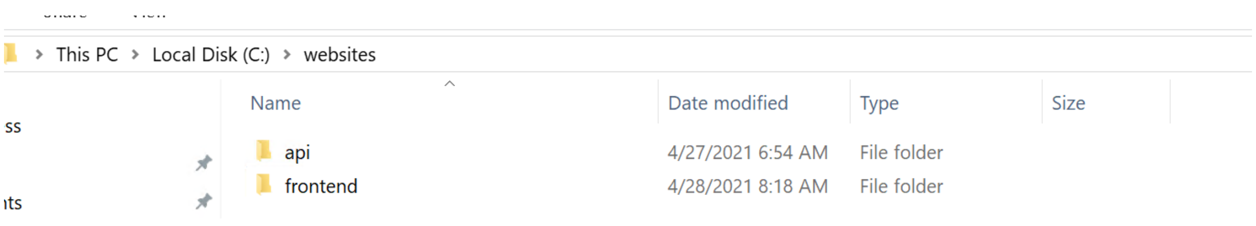


```xml
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="ast_info_db_url" value="https://rcdata.nau.edu/trilling/asteroid_data/database/ast_information.db" />
    <add key="timeseries_csv_url" value="https://rcdata.nau.edu/trilling/asteroid_data/database/timeseries.csv" />
    <add key="pub_good_db_url" value="https://rcdata.nau.edu/trilling/asteroid_data/database/ztf.db" />
    <add key="working_dir" value="C:\\ceres_db_integration\\" />
    <add key="schema_file" value="C:\\ceres_db_integration\\schema.json" />
    <add key="db_user" value="admin" />
    <add key="db_password" value="password" />
  </appSettings>
</configuration>
```

In addition, the URLs for each of the source .db and .csv files needs to be specified, as well as the MySQL username and password.

Hosting the API / Frontend

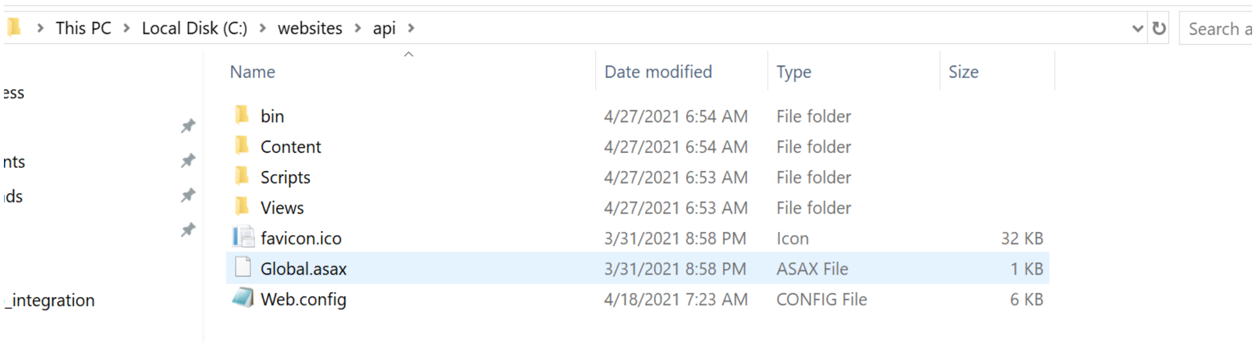
Once the API and/or Frontend project(s) have been built from source according to the instructions above, the process for hosting them is identical. First, create a directory somewhere accessible by the IIS worker process for each of the websites.
In this example, C:\websites\api and C:\websites\frontend were chosen (see below).



This PC › Local Disk (C:) › websites

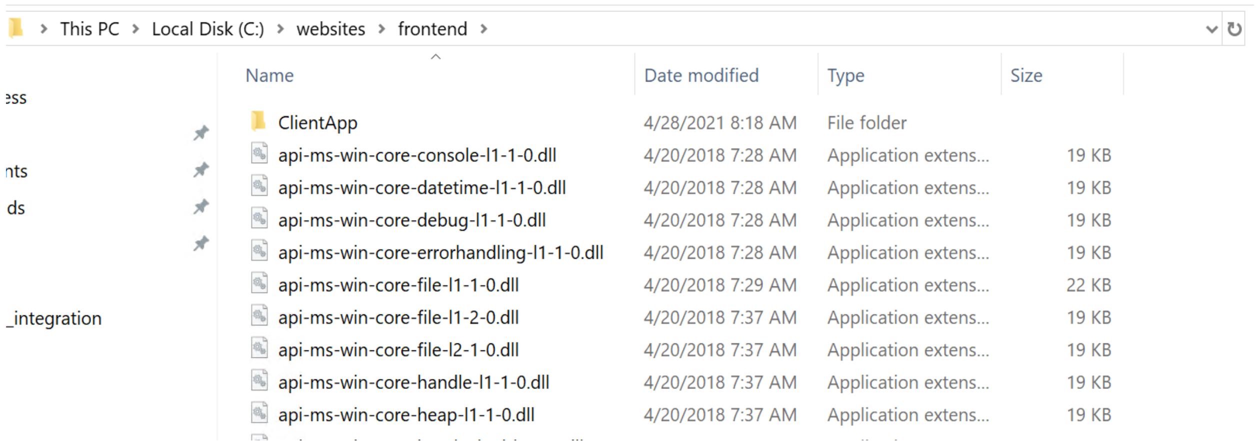
Name	Date modified	Type	Size
api	4/27/2021 6:54 AM	File folder	
frontend	4/28/2021 8:18 AM	File folder	

Next, copy the respective binaries that were compiled by Visual Studio to these directories.



This PC › Local Disk (C:) › websites › api ›

Name	Date modified	Type	Size
bin	4/27/2021 6:54 AM	File folder	
Content	4/27/2021 6:54 AM	File folder	
Scripts	4/27/2021 6:53 AM	File folder	
Views	4/27/2021 6:53 AM	File folder	
favicon.ico	3/31/2021 8:58 PM	Icon	32 KB
Global.asax	3/31/2021 8:58 PM	ASAX File	1 KB
Web.config	4/18/2021 7:23 AM	CONFIG File	6 KB

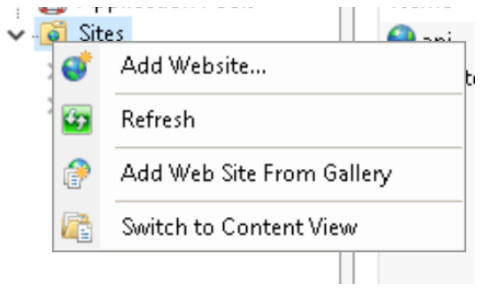


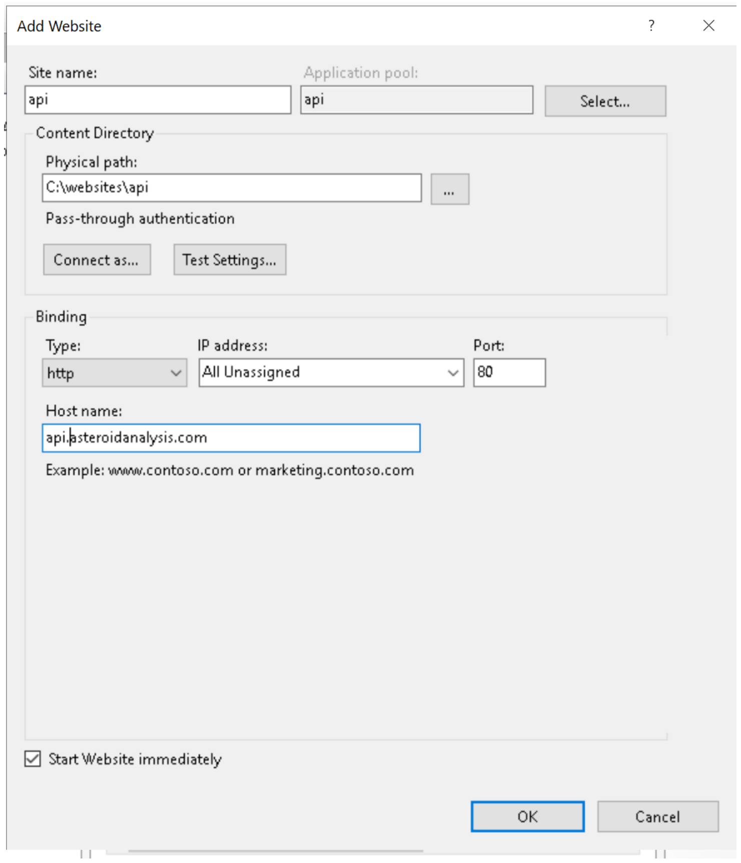
This PC › Local Disk (C:) › websites › frontend ›

Name	Date modified	Type	Size
ClientApp	4/28/2021 8:18 AM	File folder	
api-ms-win-core-console-l1-1-0.dll	4/20/2018 7:28 AM	Application extens...	19 KB
api-ms-win-core-datetime-l1-1-0.dll	4/20/2018 7:28 AM	Application extens...	19 KB
api-ms-win-core-debug-l1-1-0.dll	4/20/2018 7:28 AM	Application extens...	19 KB
api-ms-win-core-errorhandling-l1-1-0.dll	4/20/2018 7:28 AM	Application extens...	19 KB
api-ms-win-core-file-l1-1-0.dll	4/20/2018 7:29 AM	Application extens...	22 KB
api-ms-win-core-file-l1-2-0.dll	4/20/2018 7:37 AM	Application extens...	19 KB
api-ms-win-core-file-l2-1-0.dll	4/20/2018 7:37 AM	Application extens...	19 KB
api-ms-win-core-handle-l1-1-0.dll	4/20/2018 7:37 AM	Application extens...	19 KB
api-ms-win-core-heap-l1-1-0.dll	4/20/2018 7:37 AM	Application extens...	19 KB

Make sure to configure the Web.config file for the API to use the appropriate MySQL username/password.

The next step is creating two website instances in IIS.
Right click the “Sites” folder, then “Add website”.




Add Website

Site name: api

Application pool: api

Content Directory

Physical path: C:\websites\api

Pass-through authentication

Binding

Type: http

IP address: All Unassigned

Port: 80

Host name: api.asteroidanalysis.com

Example: www.contoso.com or marketing.contoso.com

☑ Start Website immediately

Specifying a user account to connect as is critical for the website to function properly.

Issuing an SSL Certificate

For this project, we used the CA at letsencrypt.org

For this (optional) step, the following are required:

- openssl binaries (https://www.openssl.org/)
- certbot binaries (https://certbot.eff.org/)

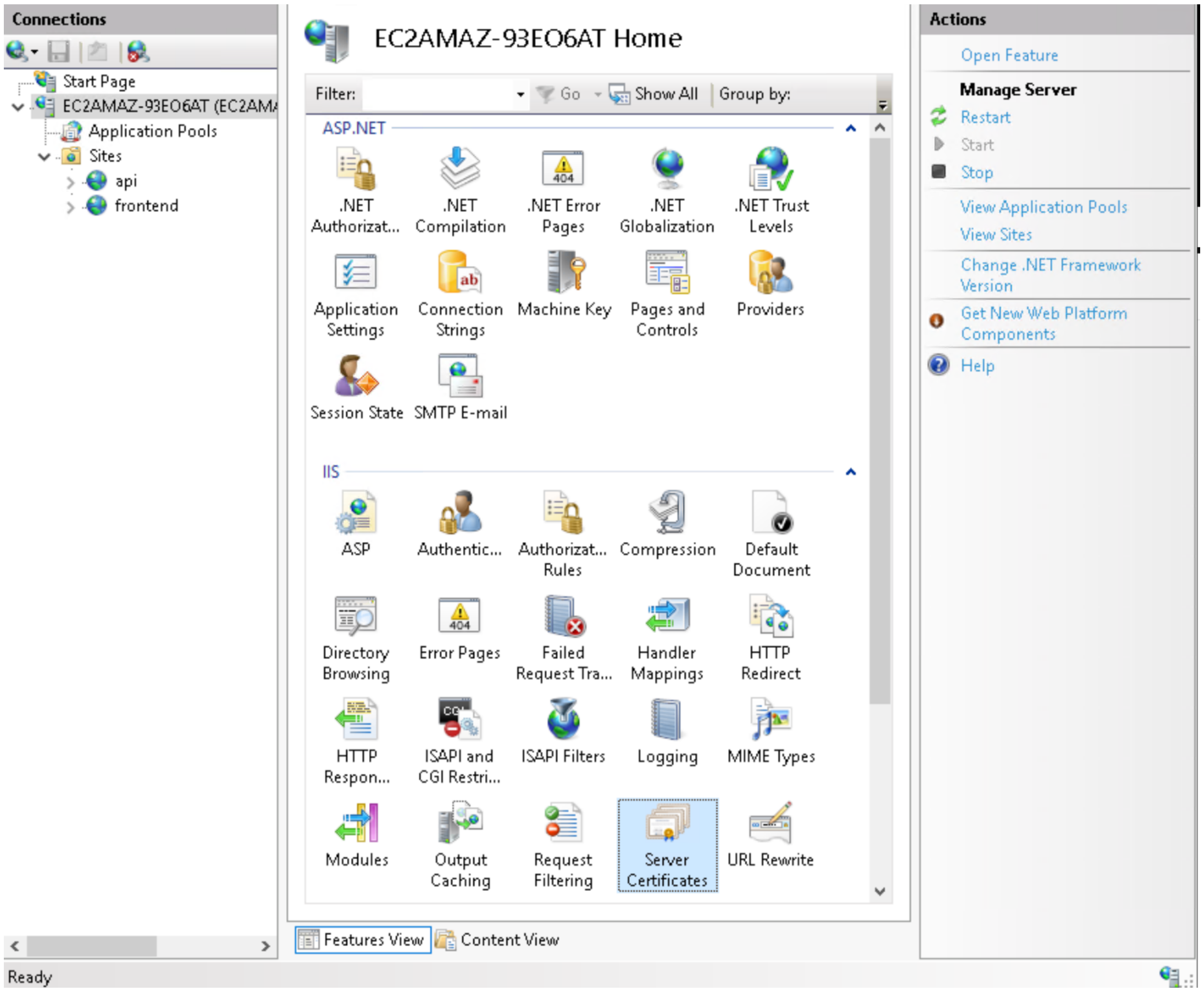
First, stop the two websites set up in the previous step.

Run certbot from the command line, “certbot certonly”.

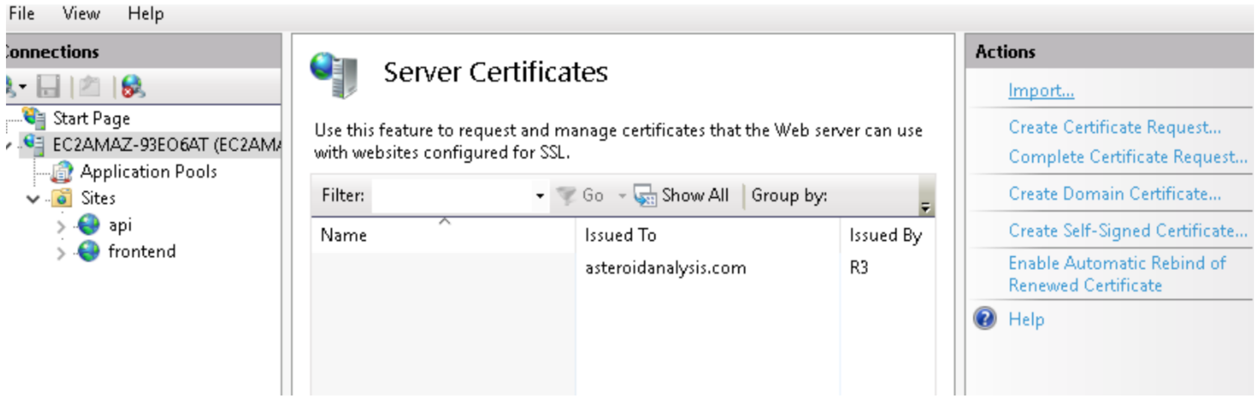
Follow the instructions prompted, entering the name of the domain(s) you want an SSL certificate for.

Then, run openssl and convert the certr1.pem and privkey1.pem into a .pfx file to be imported into IIS.

Once the .pfx file has been created, open IIS and click the machine name, then “Server Certificates”.



Next, “Import...”

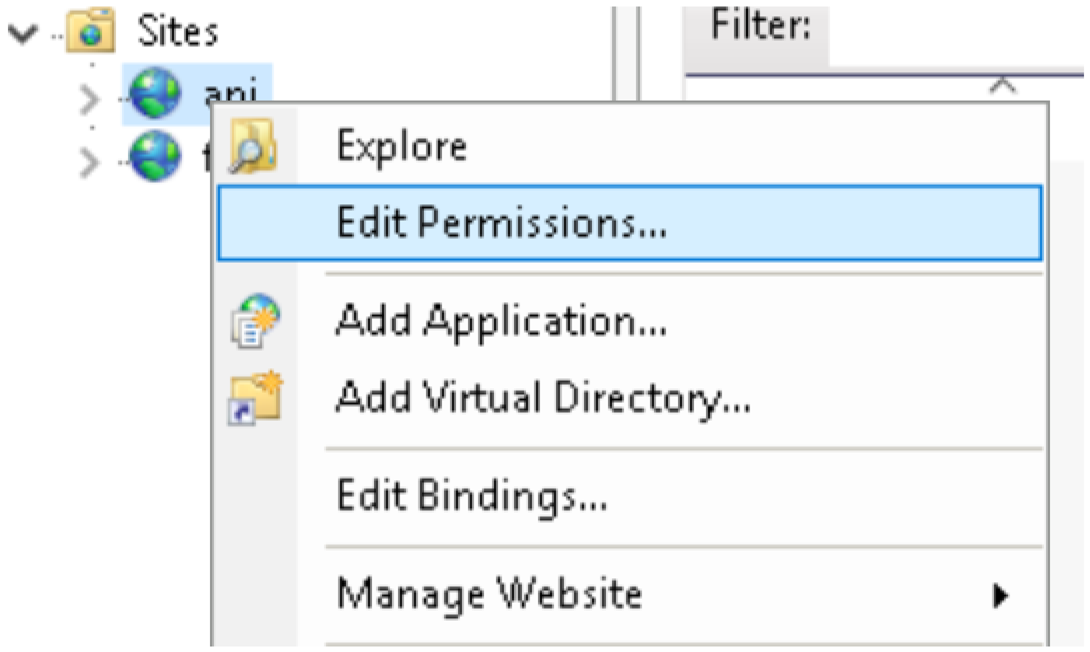

File
View
Help
Connections
Start Page
EC2AMAZ-93EO6AT (EC2AMA
Application Pools
Sites
api
frontend
Server Certificates
Filter:
Go
Show All
Group by:
Actions
Import...
Create Certificate Request...
Complete Certificate Request...
Create Domain Certificate...
Create Self-Signed Certificate...
Enable Automatic Rebind of Renewed Certificate
Help

Use this feature to request and manage certificates that the Web server can use with websites configured for SSL.

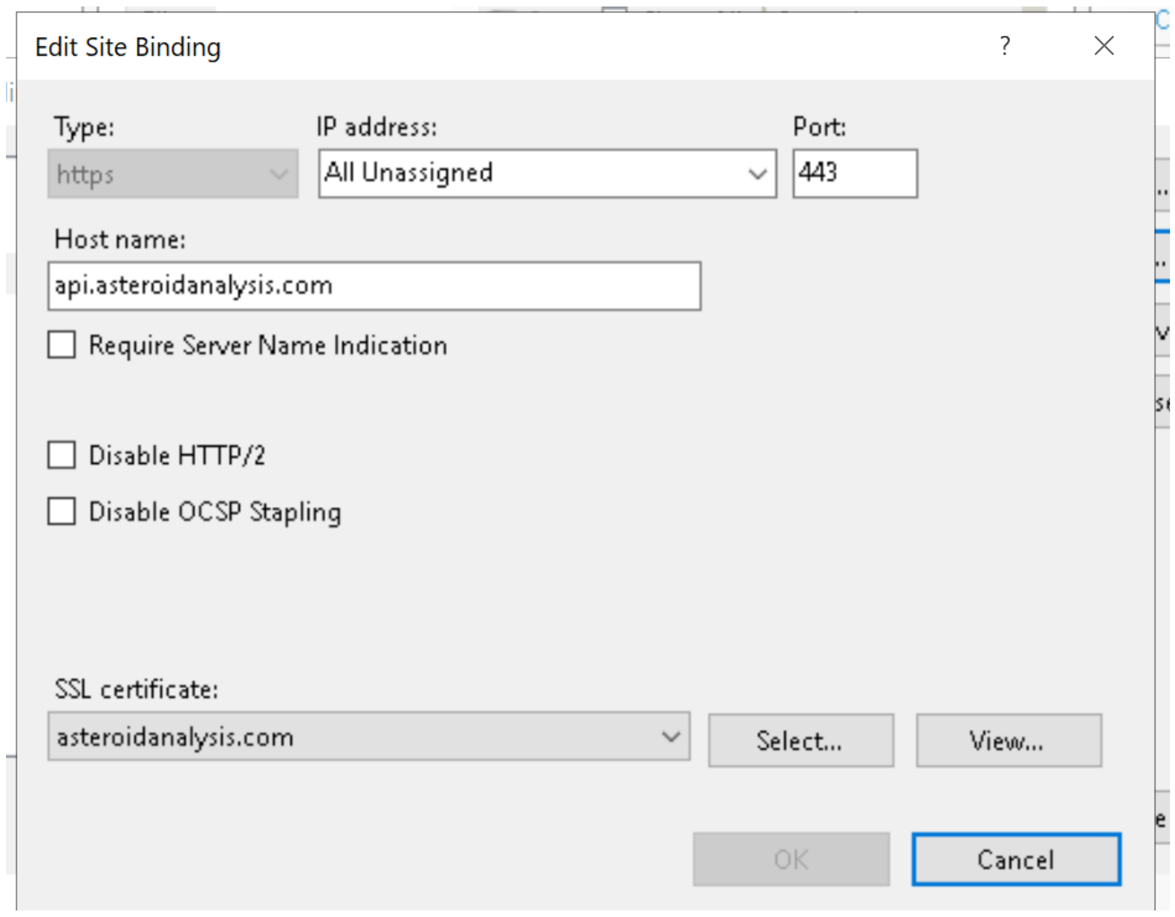
Name	Issued To	Issued By
	asteroidanalysis.com	R3

Specify the .pfx and password used in the command line for openssl, then click “ok”.

Once imported, click each website and then “Edit Bindings...”



Then select “https”, enter the hostname, and select the imported certificate.


Edit Site Binding

Type: https

IP address: All Unassigned

Port: 443

Host name: api.asteroidanalysis.com

☐ Require Server Name Indication

☐ Disable HTTP/2

☐ Disable OCSP Stapling

SSL certificate: asteroidanalysis.com

Repeat for each website you wish to issue an SSL certificate for.

Production Cycle

Once all of the above steps have been completed, any new developer should be able to actively contribute to the application. Now let’s get into the details of how one might get started with this. We created our codebase to live inside of a git repository, so it is very easy and intuitive to navigate the directories and see the file structure that we implemented. Something that might be beneficial would be to review how React JS applications are structured with components.

From here, the code base can be opened in any text editor of choice. Our team decided to use free to use versions of Visual Studio and Visual Studio Community for simplicity of showing code to each other and maintaining formatting. But again, any text editor will work. In order to make changes to the application, changes can be made directly to the JS files contained in the code base, and since the application is run using node.js, once any changes are made and saved, the changes will be reflected in the locally running version of the application.

In order to publish changes to the server (AWS), we decided to use the Build and Publish features of Visual Studio to make a more compiled runtime version of the application. Simply use these commands, save the compiled files to a local directory, and upload them to the AWS server in order to have changes be made to the live running application.