Angela Chen
Jeremy Feininger
David Gracia
Fin Headley
Dahyun Hong
Anika Walia

A4-Group 2 CS CS 411 Documentation

Here, we discuss how our project has evolved over the course of the semester. Changes and modifications are detailed in red below. The first two deliverables are in the docs folder and the prototype is in its own folder.

Link to repository: https://github.com/d-gracia/A4-Group2

Meeting Dates and Goals

In addition to meeting weekly in discussion as well as regular back and forth text discourse, we met on zoom or in person on the following dates:

2/23 - meet on zoom to discuss project proposal topics
4/6 - meet on zoom to discuss and work on the initial draft with a working api call
4/10 - meet on zoom to discuss further how to get the api calls to work
4/30 - meet in person to work on project
5/1 - meet in person to work on project
5/2 - meet in person to finish project
5/3 - meet in person to record video, make finishing touches on project

Project roles

The project roles were not strictly defined and we all worked a bit on each aspect, but generally we had the following roles as the focus of each member:

Frontend and Backend - Anika, Dahyun, Fin, David
Research, database, misc. - Jeremy, Angela

Deliverables

Deliverable 1: Idea proposals - assignment 1
Deliverable 2: User stories- assignment 2
How our tech stack choice changed
Deliverable 3: Prototype

Deliverable 1 - Idea Proposals

Idea 1: Stargazing report in your area

Takes user location and finds stars/constellations/planets/moons that would be good to look at on any given night. Checks weather API to see if it will be a clear/cloudy night. Checks light pollution API to see what is visible. Lists whether objects can be seen with the naked eye or needs a telescope. Possible AR night sky finder component? Could allow easy copy-paste of objects to AR app. A feature that tells you where the best location is to view the night sky in your immediate area based on user input for how far the user is willing to travel. Could also give the best time to view objects and the best date to view them. The database could store users as well as which objects they are interested in and their log-in locations.

Some possible APIS
http://simbad.cds.unistra.fr/guide/sim-url.htx
http://www.sky-map.org/
https://vizier.cfa.harvard.edu/viz-bin/VizieR
https://stellarium-web.org/

Idea 2: Finances

Recommends different investments to add to your portfolio. Given a user's portfolio, we use an API to see how to further diversify investments. Another API can be used to recommend what the user should invest in that specific market. A database is needed to store user-profiles and information. OAuth would be needed to log into the user's portfolio with a given brokerage.

Some possible APIs
https://polygon.io/
https://portfoliooptimizer.io/

Process notes: We were interested in creating an app with functionality related to stargazing as two of our group members are astronomy majors. Since it was our first option and it was approved, we decided to proceed as it seems to be a slightly more unique idea.

Deliverable 2 – User Stories

Hardware
This user story is necessary because users will want and need to know what hardware they need to view certain objects in the night sky.

As a user, I want to know what hardware I need to see certain objects. When picking objects to return to the user, the app will check the magnitude of the objects that would be in the sky at the time the user specifies. Depending on the magnitude of the object the app will determine what hardware the user will need to see the object. When the app returns a table to the user, it will

include what hardware is needed to see each object on the table (eyes, binoculars, a hobbyist telescope, a scientific telescope, etc.).

During a regular query (check normal output user story), hardware that is required to view an object should be listed next to each object by default. To only see objects that can be viewed with certain hardware, set a hardware filter during a regular user query.

Revision:

- Rather than showing what hardware is needed for each object, we instead allow the user to decide which objects they would like to see depending on what tools they have
- If the user would like to see objects that can be seen with naked eye, we allow the user to input that along with the zipcode. Similarly, if the user would like to see objects that can be seen with a telescope , we allow the user to input that along with the zipcode.
- This is because we were able to use components of the API correlated to visibility, and did not have enough time to set a filter to show all objects and what tools were needed for each object.

Login and Signout

As a user, I want to log in to my account to store what locations I have observed at and what objects I have tried to view.

I want to be able to log into the software with my username and password or use OAuth to log in. The user's account is used to maintain the preferences of the user, as well as store what objects the user wants to view/ has viewed in a database. Logging in will also allow the software to access the user's location and time to output objects (refer to other user stories on how location and time will be used).

Log in to the application with your username and password. If you do not have an account already, sign up for one. Signing up will consist of picking a username/email address that is not already being used by another user, as well as a password. There should also be a way to create an account by signing in with a third-party account like Google or Facebook using OAuth. If the username/email is already registered the account will not be allowed to be made. If you are having trouble logging in, click the “forgot username/password” button for assistance. An email will be sent to the given email to reset your password.

Upon successful login, you will be brought to the applications home page. When you are done with your session, be sure to log out properly by clicking the “logout” button. Upon successful logout, you will be brought to the login page.

Revisions:

- Locations tied to specific users are not saved, but each user who enters through google oauth is saved to a database and each zipcode that is enters are stored in a separate database. Additionally, objects that were visible were not stored.

- We would like to connect zipcode to user information, but because they were separate post requests, the database information was getting overwritten by the newest command, so integrating the two is the next step.
- Only way to log in is through google, but not required to have an account to use this web application
- We initially wanted to add functionality for users to save constellation data, etc. on the app, but we realized this was out of the scope of our constraints. This is probably the first feature we would add when making improvements.

Normal Output

As a user, I want to be able to be told what stars/constellations/planets/moons would be good to view at a given time and location. After log in (see log in user story) the software will ask for time and location data to look for possible objects for the user to stargaze at. If this permission has been granted, then the app automatically generates suggestions based on the user's current time and location. If not, then the user has the option to enter any location/time of their choice.

Suggestions can also be generated based on a future time inputted by the user (for either the user's current location or a manually inputted location) to allow the user to know what will be visible during a given night. Depending on the location and time given to the software, the app will return a table of objects that are visible to the user based on how likely the user will be able to see the object and how popular the object is to look at.

The output table will include a list of objects, the best time to view them, the RA and Dec of the object (the coordinates of where it is in the sky), the hardware needed to view the object, and the magnitude of the object. There will also be an ability to search for specific objects in the sky, where based on the user's location and time, the software will output if it is visible when it will become visible if it isn't already, as well as the other information in the table. The user will also be able to save what objects they have viewed or save objects they are interested in viewing. They will have access to their saved objects so that they can quickly search for them later if they want. The suggestion should also output good spots to view the sky in the area. A radius can also be inputted by the user in settings, to say how far the user is able to move. These take into account light pollution and scenery (parks, etc.)

Revisions**:**

- No information is taken automatically from the user's device to provide an automatic result. The only way to get predicted weather and objects visible are through entering a zipcode.
- A table is created to show objects visible from the zipcode given, but suggestions are not able to be made based on a future time. The objects displayed are what are visible for the same day.
- Though we could automatically provide the desired information using permissions, we figured it may be better for the user to input their zipcode as it would streamline the process of having to ask for permissions.

Seeing Conditions

In order to maximize the user experience of viewing desired constellations and stars, checking if conditions nearby are conducive to the outcome is imperative. Light pollution in the area will be checked to determine what objects are visible. Additionally, providing direct access to weather data in the desired location will allow the user to determine if they would like to go view the objects. Recommendations will be made as to when the user should view certain objects.

The app can then look for locations where there are better viewing conditions, such as better weather and less light pollution. The location can then be given to the user via google maps. The app will also look for the best time to view objects by seeing when they would be visible in the night sky and again checking other APIs to check for a time with better conditions.

To see the light pollution in a given area, start a light pollution query. Search for light pollution by place name (city/town/etc.) or by coordinates in latitude, longitude format. To check the weather in your area, start a weather query. To enable notifications for good seeing conditions, set a notification with necessary weather conditions filters.

Revisions**:**

- Light pollution is not checked and returned because we could not find a viable API or dataset to call this information from.
- Weather data is provided for 48 hours for a given zipcode, and based on the status of clouds and the sky from the weather conditions, recommendations are given for every 10 hours from now if it is recommended to go view stars.
- Nearby locations were not implemented as we decided to focus on our objects and weather APIs rather than adding a new API

Time and Location

This user story is necessary so the user can get a list of visible objects in their location or a specific location. The app will need the time and location of the user, either by taking the user's location data, or from the user inputting a time and location. The app will use this to calculate what objects are visible in the night sky at that time and location. The app can then use the list of possible objects as well as data from weather and light pollution APIs to find what objects would be optimal for viewing at the time and location.

Revisions**:**

- Zipcode is the only component needed from the user. Objects are shown that are visible in the sky that night from that zipcode. All Objects that are visible are shown, but recommendations are given at the bottom as to what time the user should go at based on the weather.

Tech stack choice

For our prototype we chose to use Flask for the backend and React for the frontend.

For the back end we were considering using Django as it was python based and has lots of useful integrations. Ultimately we wanted something python based as it was the language that all of us were comfortable using and could most easily collaborate in. While considering how we would connect the front end and back end, we decided that we wanted to use Axios to link our front and back end. In talking with Dev, he told us that he knew that Axios worked with Flask, which was also python based. We looked into Flask and decided that it would be better for our group, as it had well documented examples of how to use Axios to link it, it was python based, and it seemed better suited for smaller applications.

For the front end we wanted to use javascript. We did not want to use typescript as none of us were comfortable with it. We looked at both Vue.JS and React. In the end we chose React as there was a lot of documentation on how to connect Flask to React.

Deliverable 3 - Prototype
https://github.com/d-gracia/A4-Group2/tree/main/prototype

Process notes: We initially ran into some major issues with getting API post/get calls to fully integrate with the frontend. Figuring this out ultimately involved looking at many articles and also seeking the kind assistance of Dev during discussion section.