For my third enhancement to my custom app, MyMessierTracker, I focused on databases where I completed the CRUD operations by adding functionality for the user to edit and remove journal entries. The user could already create records by adding new object entries to their Messier Journal and inserting them into the database and they could also read from the database all of the data they had previously loaded, plus some additional object information. However, they were unable to edit their entries or completely remove an entry until this enhancement. Even though this enhancement was focused on databases, all of my enhancements have had a heavy focus on proper database schema design, scalability and ensuring that the application has secure interactions with the database.

A screenshot of a video game

AI-generated content may be incorrect.

I selected this enhancement to showcase a secure and computationally efficient mindset when designing and interacting with databases. When a user clicks on either edit or delete buttons, the system collects their *user\_id* and *entry\_id* as variables to be used in the respective queries that will ensure the correct record is being edited or deleted. This setup will also handle in cases where there may be a system error and will not allow any further steps in the process to take place if the entry id cannot be confirmed.

A screen shot of a computer program

AI-generated content may be incorrect.

If the user clicks on the edit icon then the user get the option to edit one or all elements of their original entry

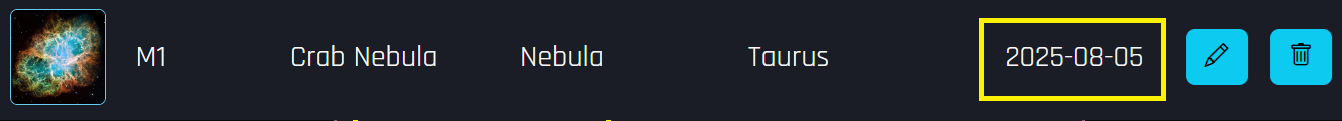
A screenshot of a computer

AI-generated content may be incorrect.

One thing I came across here was needing to make sure that I also passed all of the previous data elements that were loaded into the system already instead of blank by default, else when the user when to save their entry but say only updated the date field, the the journal notes would be completely erased as the original notes were not persisting. However I made sure to make that change and as you can see in the image above these are the previous values pop-up correctly allowing the user to review their data before saving any changes to ensure nothing is lost in the process. Once the user has confirmed their changes, then the system performs an UPDATE statement and brings the user back to their dashboard.

A computer screen with text and numbers

AI-generated content may be incorrect.



If the user clicks on the delete icon then they are prompted with a warning similar to the user account deletion process for consistency. This ensures that they are well-informed that this action is permanent and irreversible.

A screenshot of a computer

AI-generated content may be incorrect.

The code for this process collects the entry id and passes it along with the user id within a DELETE statement to remove the journal entry record as well as related records for images then also removes the image from storage.

A computer screen shot of a program

AI-generated content may be incorrect.

Then it will also ensure the correponding img is deleted from the file storage as well

A screen shot of a computer program

AI-generated content may be incorrect.

This enhancement met all of the outcomes I had originally planned to meet for this enhancement:

**#1** Completed through allowing the audience the ability to make key decisions in the data they load such as how to update it and/or remove it entirely, the power is in their hands

**#4** Integrated Postgres > Python > Flask/Jinja framework that uses Highcharts for user facing analytics. This provides a smooth and clear UI for the end user that provides snappy data and interactions. As the user interacts with edit/delete buttons they see that the analytics on the page also update accordingly.

**#5** Ensured a secure application through the use of UUIDs, sanitizing user input and validating data throughout the entire process. All edit/delete actions are scoped to only the authenticated user and record id in question and use server side validation to prevent tampering or unintentional leaking of other user data.

As far as challenges this week, I did run into that oddity where I had not accounted for populating the edit modal with the current information the user had uploaded and so it was unintentionally deleting a lot of data, but once I figured it out it was an easy fix where I just needed to read and pass in that data first so it populated that modal and the user could easily infer that that was the data that would be saved unless edited.

Another issue I’ve been encountering this week is getting my web app to show on GitHub pages for my ePortfolio. Apparently, it has something to do with Flask and Jina and it not being a static page. Therefore, this forced me to start testing web serves earlier than I had planned so I now have it hosted on Render (through my GitHub where I have also created a Postgres database there following the exact same schema as in my GitHub). This is not ready for live web deployment in my opinion as there are many things I would still like to do such as get a Google account API connection implemented for cleaner and more secure logins, and now I need to also re-seed the app with more user info so that the rarity metric on the tooltips feel more realistic, however this is currently live if you would like to take a look at it and provide any feedback - I’m open to any suggestions! <https://www.mymessiertracker.com/>