G.O.A.T. Issues

By Peyton Vanhook, Peyton White, And Sean Hopkins

As was expected, the GOAT has run into a few hiccups along the way. Some problems have been solved while others remain a nuisance. We will discuss issues that we have already solved, are in the process of solving, and lastly foreseeable issues. By discussing the issues that we have come across through development of the GOAT we hope to gain a better understanding of the issues. Starting off, there are the issues that we have managed to solve since we started developing the GOAT. These issues played a role in warming us up to the problems that we will likely experience later.

At the start of the project we chose to code in Netbeans using JavaFX. When downloading Netbeans through Apache, we ran into a problem with creating JavaFX projects through the JavaFX application. Seeing as we need JavaFX code for the GOAT this was a rather important issue. The solution that we found was to implement Java using Maven. With Maven we have been able to create the projects we need to work with JavaFX and start creating the GOAT. Another difficulty came in the time consumption of typing out every line of code for the GUI and testing it. SceneBuilder was an option that we found to be extremely useful on this issue. What would have taken hours to write out in code by hand and even more hours to test is now done within minutes with ease with this visual layout tool.

Once we were able to create projects and GUI time consumption was figured out there was the issue of dealing with traversals from window to window. Without traversal to another window the GOAT would be stuck at the login page with nowhere to go. This issue was solved by using Java Class files as controllers for each specified FXML file with functions to switch from one window to the next depending on the action of the user. In these window switching functions the FXMLLoader function would be utilized and a new scene would be created based on the specified FXML file.

Another solved issue is during the process of working on the project we needed a way to send our work to each other without too much hassle so we started using Github. Initially, we had some issues getting Github to work with Netbeans. Netbeans does have Git integration, but there were several issues which caused initial problems. First, Netbeans would often have connectivity trouble with Github which would occasionally lead to loss of data. Additionally, if we wanted to remove a file there was no way to do so in Netbeans. Finally, we found the UI for resolving conflicts within netbeans to be cumbersome. Our solution was to use Github desktop to maintain the local repository and update the Github repo, and not use the Netbeans functionality at all. This proved to be much simpler than using Netbeans itself, and we have not had any problems with repo management since. Additionally, using github desktop will make it far easier to include documentation and other files than Netbeans functionality would, as Netbeans can only sync to a repo a project file.

The next solved issue involved connecting to the database in the beginning. The GO-Daddy server is private so when trying to connect to it from a php file outside the server it was not connecting because it wasn’t an admin. Even after trying to manage the users allowed to connect to the database it still would not let us. What solved this issue was creating a PHP file on the server itself. From here every time we create a page that was associated with a URL that needed to access the database we would called the PHP function “require connect.php” inside that PHP page. For example; we created a checklogin.php page that is essentially a private page online that can only be accessed through the URL itself. If this URL is activated given a post signature the PHP gets ran.

The last solved issue was involved when the user logs in, all their data from the database gets saved into a user object for faster access of the data. After saving this data the issue was how do we pass that same user object around to each window as the user goes through the application. First we tried to pass the data through the actual window as a parameter. This was done by creating a constructor that initialized a user object inside the class but it was being real buggy and throwing a nullException every time a new window was opened. So instead we researched about how to do it and the solution was similar to the constructor method. Inside each controller class which has its own window there is a method called setUser() that is public. Every time the user chooses to go to another window we load that window into the future stage and before calling window.show() to let the user see that window, we load the future controller and call setUser() and pass in the user object that is saved from the login screen. This allows one user object to be passed throughout the application.

Next, we will discuss the issues that we are currently working on solving. First, we’ve had some issues sourcing media data and processing that data. Our biggest struggle, initially, was finding datasets that had licenses that permitted us to use them and also had the data we needed for our project. Several datasets we looked at had all the data we needed, but was copyrighted or licensed in a way that prohibited us from using it for this project. Other datasets had good licenses but didn’t have the data we needed, or were not tightly formatted and would have required significant work to parse beyond the scope of this project. Thankfully we have located 3 tightly formatted and comprehensive datasets (one for books, one for movies, and one for video games) which have licenses that permitted us to use them for this project.

We have not had any issues parsing the video game or movie datasets, but we have had some issues with the books dataset. The books dataset is an approximately 10,000,000 record set sourced from the Library of Congress. While the parsing function itself has extracted the information we need without any errors, we have encountered problems running the function over the full set of records. The full algorithm extracts one record at a time from the file stream as a string and passes the string to the XML parser. The results are returned as an array which is written to the appropriate files on the local system. The algorithm, for an unknown reason, will stop consistently at a single record in the file. It doesn’t throw any error, nor is there any evidence of a memory leak; the program simply pauses at this record each time the algorithm is run on the composite file. Thankfully the Library of Congress provides this dataset as a 43 part collection of XML files, so our solution to this mystery bug is to programmatically download the dataset as individual files and parse each file on it’s own. We’re currently working on this new downloading program, but the parsing algorithm itself is already equipped to handle multiple source files and will not need significant modification to accommodate this change.

Beyond this program problem, there is still the issue of filtering the data we have received, especially the data in the books dataset. Many of the books in the books dataset are actually academic publications or government documents, which do not fit our focus on entertainment media. Additionally, in all three datasets there is plentiful explicit material and non-english material that are outside the scope of this project. Finally, each of our datasets includes genres that don’t overlap, and keywords, especially in the books dataset, which only occur in one record in the total set. We intend to limit the total set of genres and keywords to a smaller set to make our database more manageable, and to map certain similar keywords and genres to the canonical set. We will reduce the size of the set using a filtering function and use a mapping function with manually programmed mappings for the remaining data.

A current issue that has to do with the GUI is implementation of a drop down menu. The issue comes in when an option is clicked from the drop down menu. On the action, the program throws an exception but does not break or perform any actions that should be prompted by the click. This issue should be easily solved with some research into how to work with a drop down menu in JavaFX. Once the information is found, the issue should be dealt with swiftly.

Another current issue has to do with loading pictures into Java. We plan to implement pictures in a few ways. We intend to use pictures for design purposes, give the user a visual of the media they are viewing and to add individuality to a user's profile by having the ability to display a photo of their choosing. While we had issues with pictures breaking the program or not loading, there has been progress. We have figured out how to get pictures to load, but now we are working on creating the capability to change and choose profile pictures. The solution to this issue is similar to the prior issue, as it will most likely entail research into how pictures work in JavaFX and how we may work with them to create what we have envisioned.

Lastly talking about the foreseeable issues we anticipate having some issues inserting media entries into the database. Our data source files are larger than what can be stored on our web server, so we plan to parse the data locally on one of our developer machines and push that data to the web server. We anticipate some issues pushing that data because, even when parsed, the data is still several million lines long. We are considering two strategies to accomplish this. First, if we have the space on the web server to host the parsed CSV files, then we will upload the CSV files directly to the server and a conversion function will write the data into the appropriate database. If we don’t have the space to host all or any of the parsed files, however, then we will need to create an API call to which we can send some array of media records for writing. The problem with this second strategy is, of course, that there will be communication delays which might make pushing the whole dataset take a prohibitively long amount of time.

Additionally, we haven’t yet implemented the recommendation algorithm and there are several potential issues that can arise there. We’ve designed our database around what the algorithm we’re using expects as input, so we’re hoping that proactive design will reduce any potential problems in implementing the algorithm itself. However there may still be issues processing through the whole dataset, so we might have to come up with a scoping scheme for the recommendation algorithm. Finally, there’s likely to be a process of fine-tuning the algorithm to get it to return results that feel usable, however we acknowledge that some of this fine-tuning is beyond the scope of this project.

In conclusion, there have been a myriad of issues that this project has run into. There are those that we have managed to solve as a team, those that remain a problem waiting for a solution, and those that we foresee in the future. Through the discussion of the issues of this project we have gained experience in problem solving which will prepare us for once we get into the workforce.