CMPUT 291 Mini-Project #2
DOCUMENT STORE APPLICATION

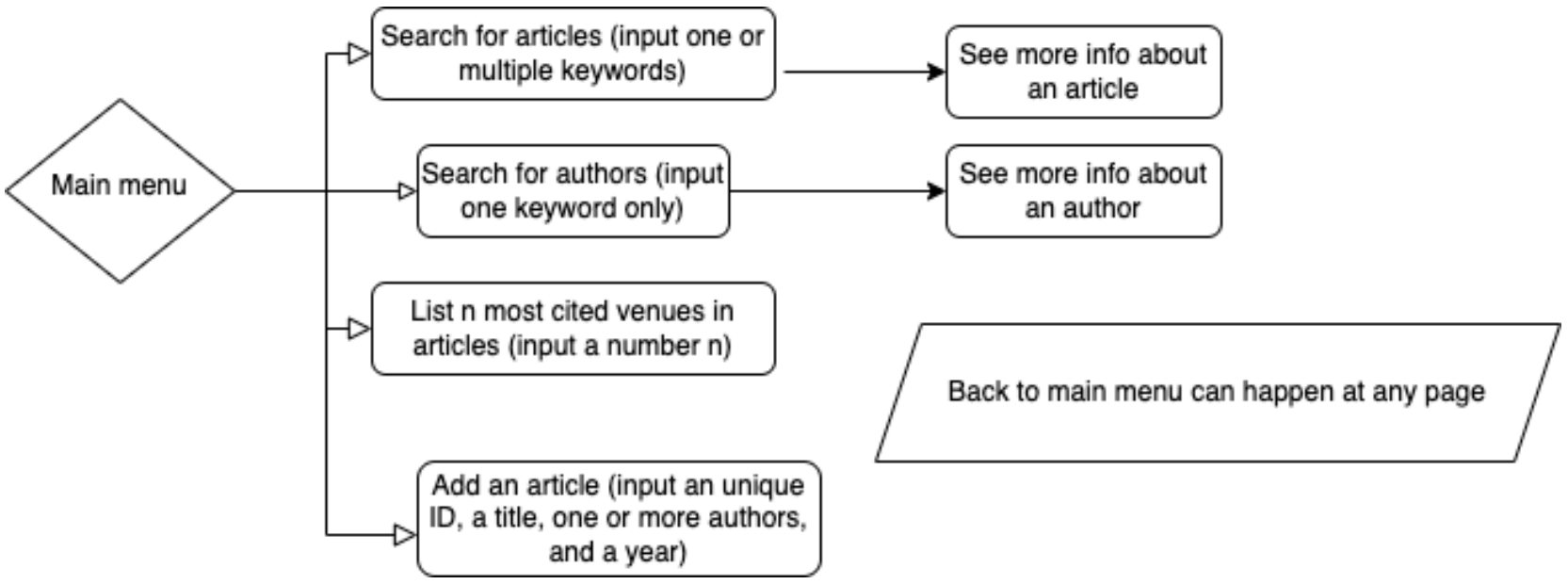
A. System Overview

Our application acquires some of an online document store/library database's basic features that provide services to the users using data from a database. It was created using Python and its standard GUI library named Tkinter for the functionalities and MongoDB for data management from a NoSQL database (data stored in JSON format). The program is similar to a document store or an online library database where the users search for their favorite articles, add their own articles or inspect which articles are the most referenced. What the users can do with the application is very simple. As soon as they open the application, they can perform several actions on the platform, as stated above.

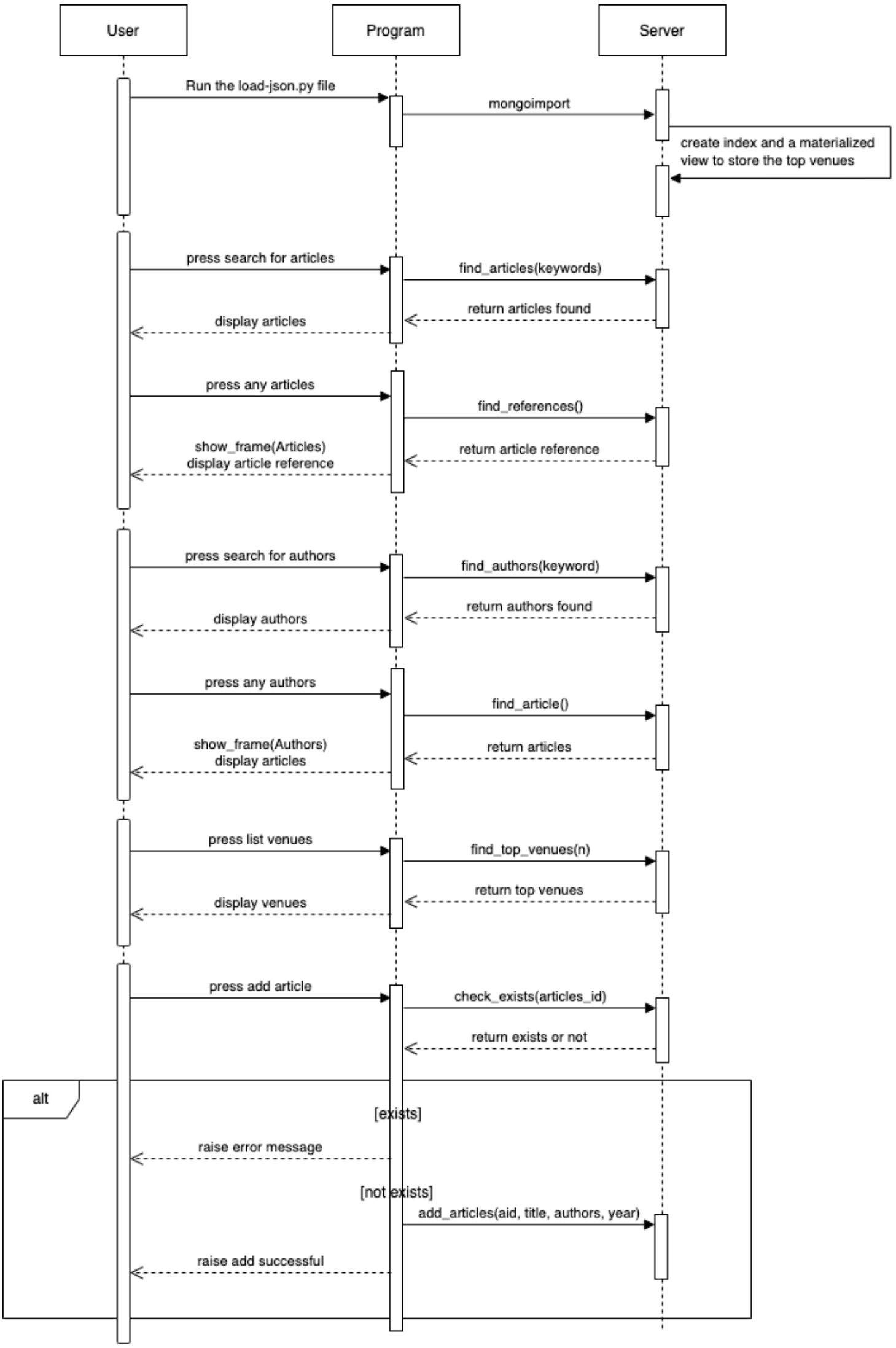
The most dominant part of the program's design lies in using Tkinter, a Python graphical user interface. The users act on the application's features through buttons, labels, and search entries on the interface, contributing to the user experience. The GUI is also functional during the creation process by helping the programmers navigate easily and thus be less likely to make mistakes due to its simplicity and visibility. Python was chosen to write the program because of its compatibility with MongoDB through the PyMongo library - the go-to language for managing data from a NoSQL database. Below is a small user guide on how to use the program.

User Guide (for Linux users):

1. Start the server using mongod.
2. Start the program using the command **python3 main.py**. You need to input the relative path of the database and a port number you would like to run the program on.
3. Then, you can perform many actions on the main menu: search for articles, search for authors, list the most n cited venues (input a number n), see information about an article and add a new article. You can also return to the main menu from any page.



B. Software Design



C. Testing Strategy

We used the same testing strategy we did for our mini-project 1. The testing strategy used for the program follows a method called “Divide and conquer” that strictly follows the software’s design. As described in part B, the program is divided into smaller components, each responsible for some specific application functionality. This way, it is better to test and debug because as we progress through the creation process, we always ensure that each component works first before moving on to the next. One example is that before we got into the coding/querying part, a structured backbone was created with empty classes and their functions with details of what they do using block comments. Mainly we tested after every class and every function (which was not always the case), but the idea is to test/debug each program's functionality before moving to the next.

Due to the nature of the strategy, it is easier to narrow down where the bugs are using the process of elimination because we test part by part, so the bugs are usually small. Here are some test cases and examples of typical bugs we faced throughout the project:

1. All MongoDB queries were tested separately before being integrated into the host application. If a bug in a data-fetching function happens, it is difficult to tell if it comes from code or the MongoDB queries written. This method reduces the debugging time because we would run the query file separately and then print out the result on the terminal or check in the database from the client side (using Mongosh) for testing purposes. Some example errors included syntax errors in MongoDB (missing brackets) and some logical errors, such as forgetting to preserve NULL and empty arrays for the $unwind aggregation would make the counting of documents incorrect (for question 3). The testing process for these MongoDB queries all boiled down to perfecting the queries written and trying not to solve the errors using the host language.
2. For the Python coding part, the testing process is similar to how we did it in the first mini-project. The assert keyword and try-except blocks combined with print functions play a big role in catching errors. These are usually syntax errors, and logical errors only sometimes occur. Exceptions are caught easily with exception handling. Some examples would be missing “self” in a class function or using incorrect logical operators. The errors in the Python code happened the least often and were usually the most insignificant due to our break-down strategy as we coded and tested function by function. The more serious errors come from the MongoDB queries instead, which would be tested separately, so we did not get confused over the errors. An example error while coding our host application is the incorrect inclusion of the escape sequences while concatenating keyword strings together (for question 1). The escape sequences “\b” were needed so that PyMongo could search the keywords as a whole, not as substrings of some other words.
3. The use of Python Tkinter could be error-prone, despite helping a lot with envisioning the process. Therefore, it is important to check the Tkinter documentation to ensure that the correct methods and syntax are being used and the debugging process is the same as above. One example error is invalid library syntax being used because the library has been updated and is different from itself ten years ago. The testing process on this aspect was much smoother because we had already gained some experience from doing the first mini-project.
4. Optimization is an important part of this project. We struggled with long running time while executing the queries on the largest JSON file until we realized that INDEXING would help and seemed to be the only solution to this optimization problem. We created a text index and normal index on every field in Phase 1 to support our search queries in Phase 2.
5. We also strived to find some edge cases. For example, in question 1, the stop words cannot be used as keywords for searching, so we needed to exclude those words. The “year” field in a document in the collection is of integer type; therefore, we were required to update these fields to string so that the search process could be done smoothly with other fields.

D. Group Work Breakdown Structure

Here is a detailed overview of the project work breakdown:

- Andy finished Phase 1, which includes loading the JSON file onto a database and creating indexes on the collection. Khac Nguyen created the program's backbone, including creating separate files for easier management, importing libraries, defining classes, functions and global variables, and finally, initializing the connection with the database created in Phase 1. Khac Nguyen created a GitHub repository for the project (we chose GitHub as a collaboration method), although we did most of the project together on a lab machine in CSC. Then, Andy would inspect the initial structure of the program, give some suggestions, and fix any problems.

- There was a meeting before starting the project after the backbone had been created. In the meet-up, we discussed the program's structure further to see if we could improve anything. We decided to only have two files for the program in Phase 2, one for the GUI and one for the data-fetching functions that contain MongoDB queries. We did mini-project 1 together, so we agreed on certain uniform styles of naming and formatting our codes to make the collaboration easier.
- In this project, differently from the first one, we all did everything together because we felt that the project did not have a lot of parts to do, and we would learn things from each other while doing together. The main and hardest part is the MongoDB querying and optimization, which would require two people to brainstorm and do research together rather than alone. We went through question 1,2,4 first, testing the correctness by using the small JSON files and then optimizing the running time using the largest JSON file. Finally, we tackled question 3 together, which was the most challenging.
- Then we all did GUI.py together, where we tested the functions we made earlier and the program and debugged everything (both in the Python and MongoDB codes), which took quite a long time.
- When we were done with the program's main components, we put down a list of small action items and insignificant details/tweaks in the program that needed to be fixed/completed. The rest of the creation process was spent tackling these little details and testing some edge cases.
- Finally, we focused on writing the report together. Throughout the project, Andy made a Google Doc where both members could track their progress on the document and let the other know which part of the project had been completed. The Google Doc combined with GitHub is a great method of coordinating that kept both of us on track throughout the creation process. Also, they helped a lot with writing this report because we did not have to think back and try to remember each progress.
- A limitation of our program is that the abstract of an article may be too long so that it cannot fit into the frame.

PROJECT TIMELINE TABLE:

DATE	WORK COMPLETED	TIME SPENT	BY
7 November	Started phase 1	30 minutes	Andy
9 November	Finished phase 1	3 hours	Andy
11 November	Started making backbone	1 hour	Khac Nguyen
14 November	Finished backbone	30 minutes	Khac Nguyen
15 November	Worked on question 1 phase 2	2 hours	Khac Nguyen and Andy
17 November	Worked on questions 1 and 2 phase 2	3 hours	Khac Nguyen and Andy
19 November	Worked on questions 2,3 and 4	5 hours	Khac Nguyen and Andy
21 November	Finished all questions	4 hours	Khac Nguyen and Andy
22 November	Optimized and worked on GUI	5 hours	Khac Nguyen and Andy
23 November	Worked on GUI and debugged	5 hours	Khac Nguyen and Andy
24 November	Finished everything and submitted	4 hours	Khac Nguyen and Andy