Recommending Alpha

Software Engineering Spring 2019

Group #16

Group Members:
Vedanta Dhobley: vjd41@scarletmail.rutgers.edu
Avani Bhardwaj: ab1572@scarletmail.rutgers.edu
Shazidul Islam: si194@scarletmail.rutgers.edu
Akshat Shah: avs91@scarletmail.rutgers.edu
Kutay Kerimoglu: kk851@scarletmail.rutgers.edu
Alan Patel: akp122@scarletmail.rutgers.edu
Anthony Matos: amm720@scarletmail.rutgers.edu
Joel Cruz: jc2125@scarletmail.rutgers.edu

URL of our projects web-site:
Github Link: https://github.com/vedantadhobley/SoftwareEngineeringProject2019
We will be using this repository to commit all changes to our project. All form of communication will be through weekly group meetings, messages, and smaller group meetings where smaller groups of people will meet to finish mini-projects.

Team Profile:

a. **Individual Qualifications and Strengths:**

Avani: C++, data organization, documentation, presentation, management
Vedanta: Java, Python, AWS, design, presentation, management
Shazidul: C++, Java, Python, SQL, design, management
Alan: Java, Python, SQL, design, data analysis, documentation
Kutay: C++, some Java, programming error checking, documentation
Anthony: Java, C++, presentation, documentation
Joel: C++, presentation, do cumentation
Akshat: C++, Java, Python, SQL, design, documentation

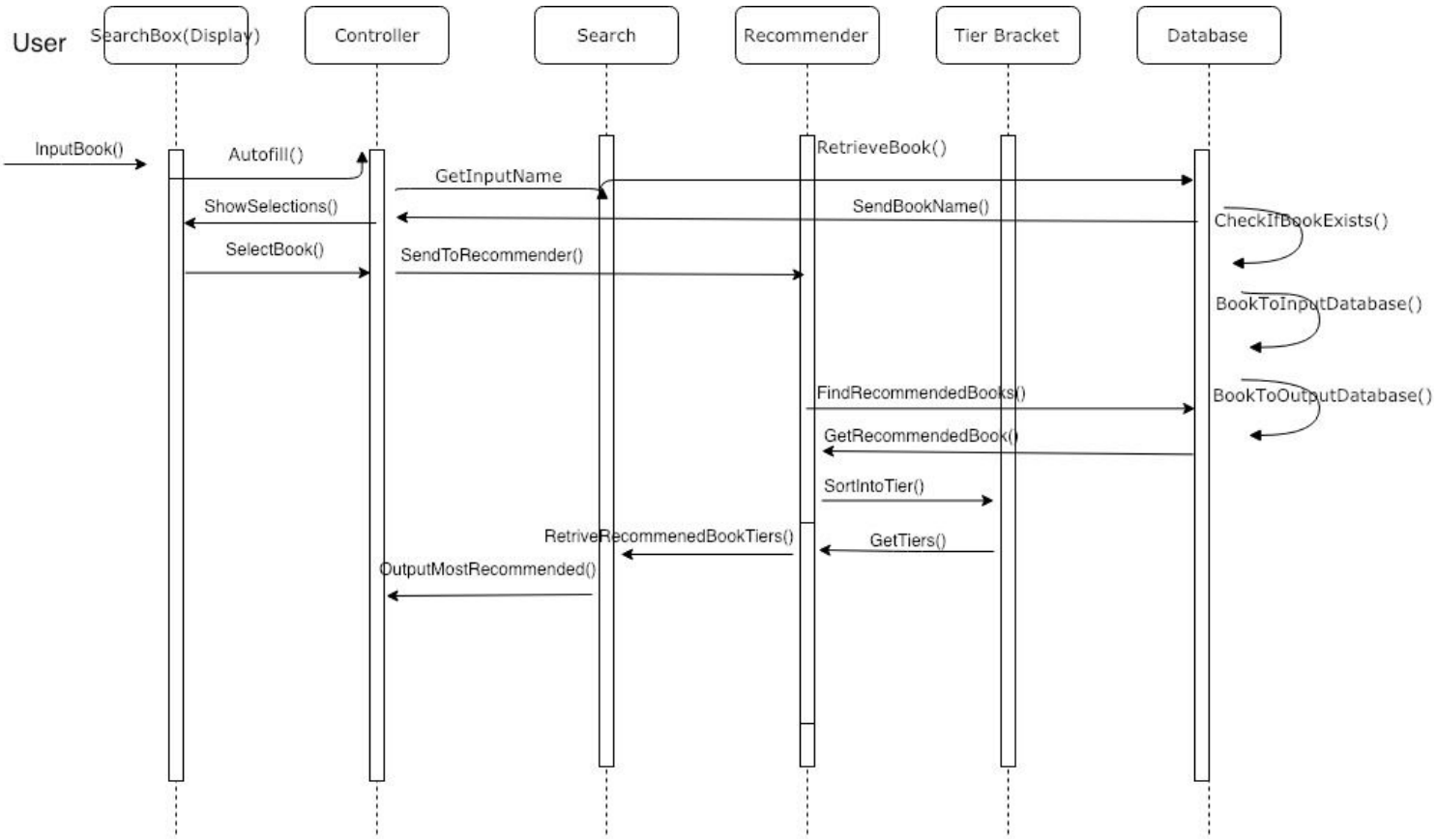
b. **Team Leader:** Vedanta Dhobley

Table Of Contents

Interaction Diagrams

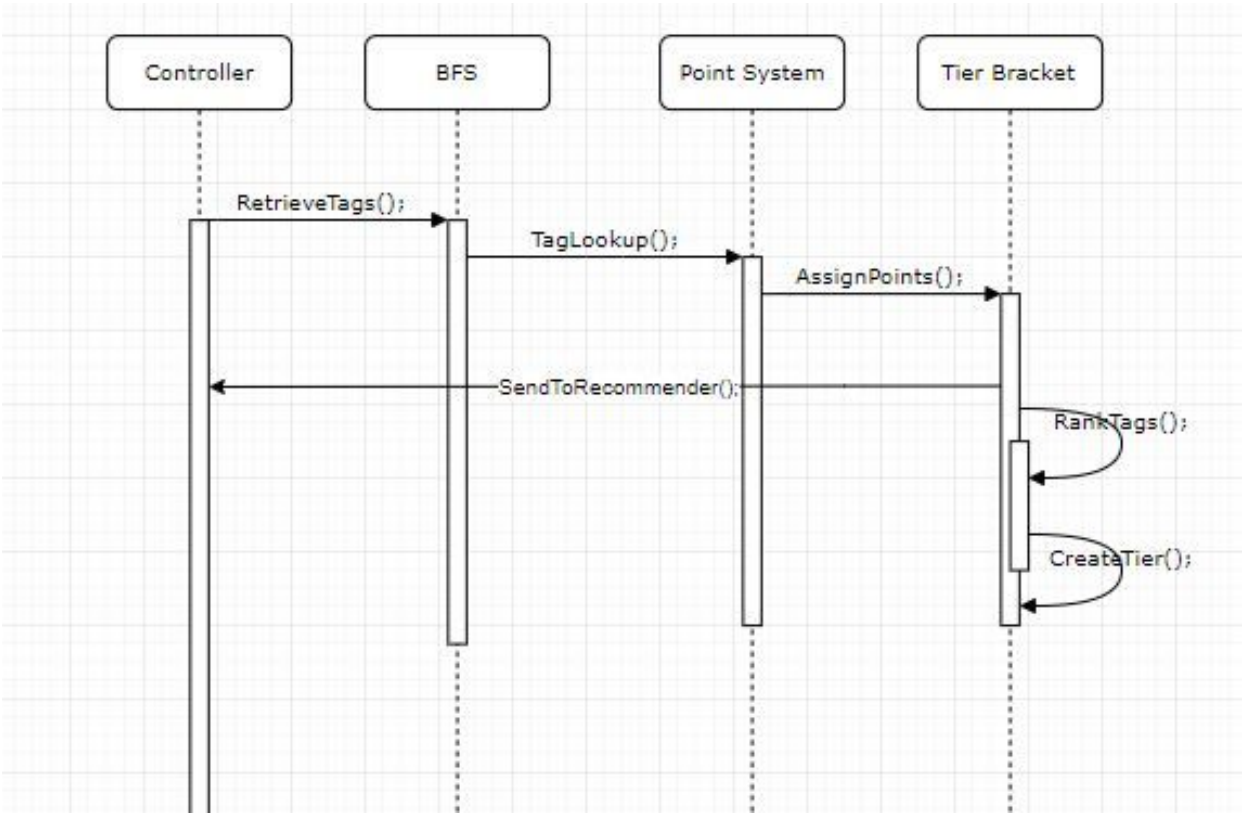
These are the use case diagrams for the two use cases we have chosen to fully address.

Use Case 3



For use case 3, which is the most dense use case in regards to tasks that need to be done, the main focus was making many distinct objects who allows for use case 3's tasks to be broken down into as many singular tasks as possible. The main design principle that allowed for us to assign responsibilities to all the objects was the high cohesion principle, since our use case 3 involved the most computational and data manipulation out of all the use cases. The single responsibility principle from SOLID also helped us in breaking down the task into classes, followed by the high cohesion principle assisting in helping break down responsibilities of the concepts in each class. The goal was to make sure not a single class had objects responsible for too much at once, by creating several objects interacting with each other in order to make up that class.

Use Case 4



Use case 4 had a lot more to do with the user interface, and communicating the correct results to the user. Given the task of use case 4, it was a dead giveaway to us to look into the low coupling principle when assigning responsibilities to objects because the low coupling principle mostly deals with communications in regards to one object being the primary one to delegate responsibilities to other objects. The goal here was to make sure that any object isn't responsible of communicating too many things independently, and that we made sure different things that needed to be communicated were each handled by single objects. Liskov's Substitution Principle also assisted with this use case because it allowed us to make sure that each object should operate independently, but should still interact with other objects. No changes in our software, system, or objects around a certain object should directly affect any object, only changes directly made to an object should affect its performance and outcome.

Interaction diagrams are most effective when they're manufactured off the basis of a design system sequence diagrams. What a design system sequence diagram does is it looks at all of the objects that help drive a system, and breaks down how exactly each object interacts with other objects. These interactions conclusively defines how each object contributes to execute

what the system is intended to do. They are very in depth and much more detailed visual representations of how each object plays a certain role in getting a task completed.

Objects are the different functions inside of a class (programming definition of class), which make the class fulfill its job. So we had to look at how each object interacts with one another, and break down a task into pieces and make sure each piece is given attention from objects.

When assigning our objects to responsibilities, we used certain design principles in order to make sure that each object had a fair share of responsibilities and that the work between the system was fairly spread apart so no single object had too many tasks. We also used some of the techniques provided through the SOLID design principle to make sure that we were forming a diagram that does necessary tasks, and nothing was redundant or irrelevant.

What use cases have been implemented?

So far, all of the test cases have been finished.

What is already functional, what is currently being tackled?

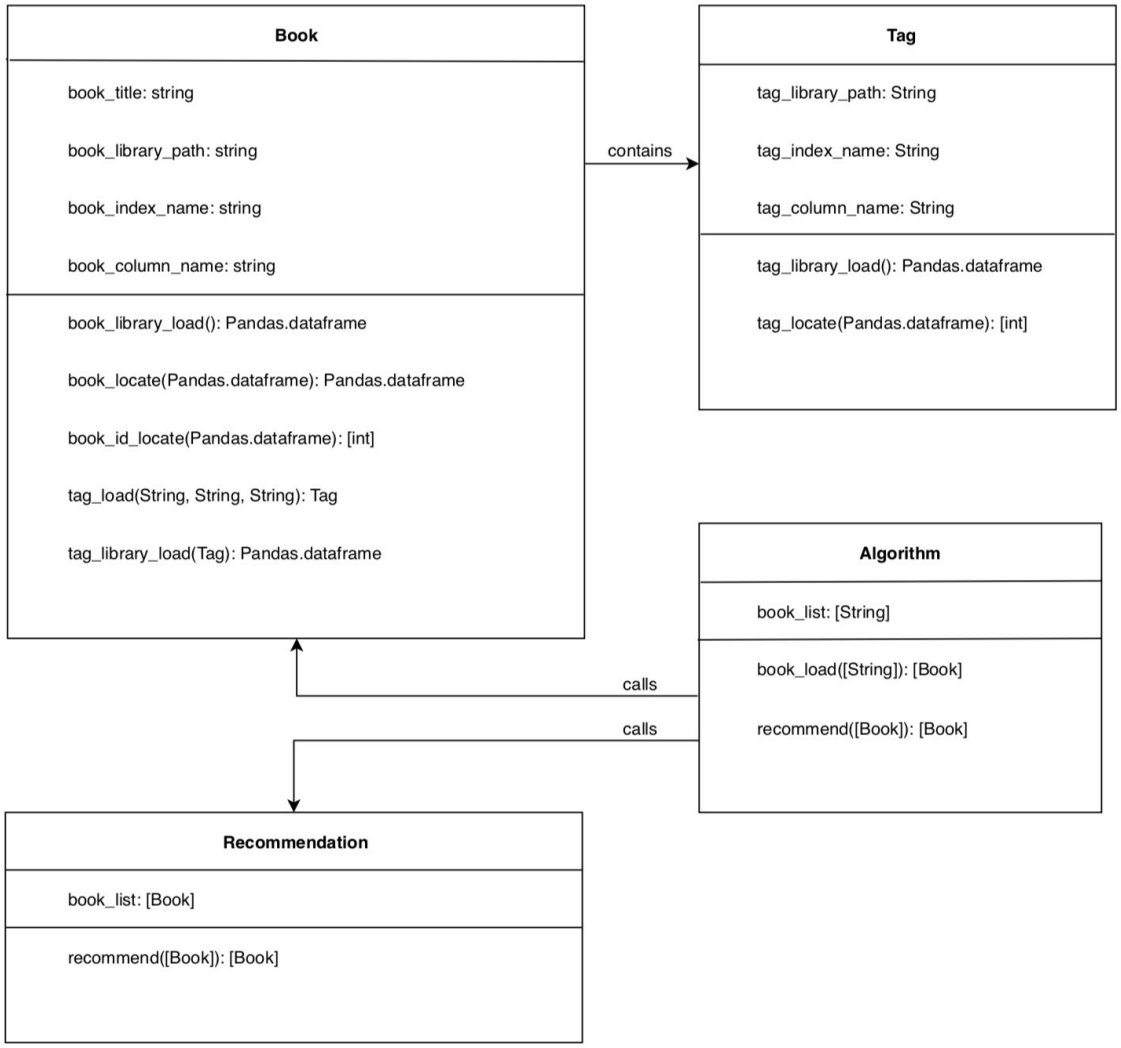
So far, the search, database (static, local) and retrieval for the recommender have been implemented abstractly. We've begun work on creating the object classes in order to store this information logically. The recommendation algorithm has just been finished and is ready to be used.

Other relevant project management activities:

Branches and folders have been created for each development team (backend, recommendation, frontend, database) with ground broken in backend regarding retrieval and object creation. As far as project management is going, everyone has been contributing equally and uploading weekly information to our github repository. With upcoming deadlines, the use cases will be implemented soon and the sorting algorithm will be set in motion in about a week or so. After that has been done, we will start bug testing and checking the database to see that the information can be retrieved with ease. As for now, there are a lot of unreadable characters in the excel spreadsheet that need to be fixed for the search to work properly. We will go through the database to fix the errors which will be done during the bug testing portion of our project.

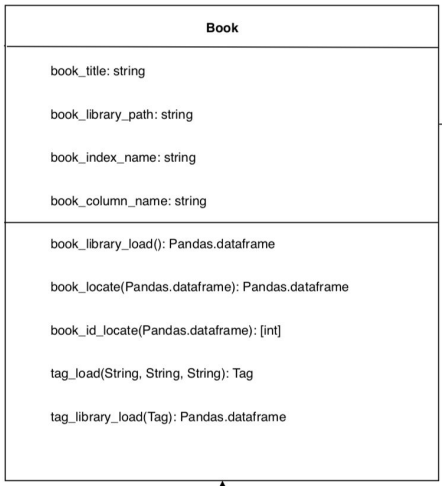
Class Diagram and Interface Specification

Class Diagram

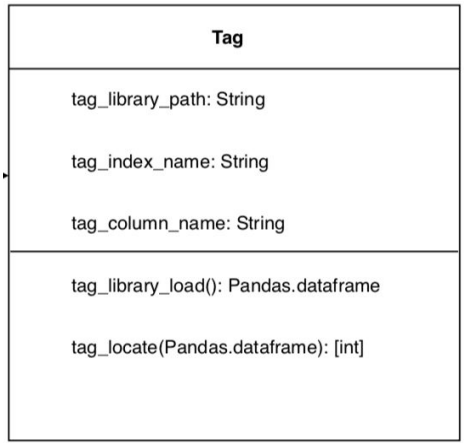


Data Types and Operation Signatures

- Book class: an object class storing data for each inputted book (including the tags for each book, stored as Tag objects).



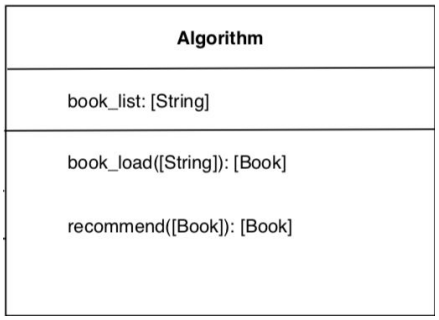
    - book_library_load(): creates a dataframe of our csv file containing the book data.
    - book_locate(Pandas.dataframe): creates a dataframe containing only the searched books.
    - book_id_locate(Pandas.dataframe): creates a list of book_ids pertaining to each book searched.
    - tag_library_load(): creates a dataframe of our csv file containing the tag data.
    - tag_load(String, String, String): creates a list of Tag objects for for each inputted book.
- Tag class: an object class storing data for each tag pertaining to each inputted book.



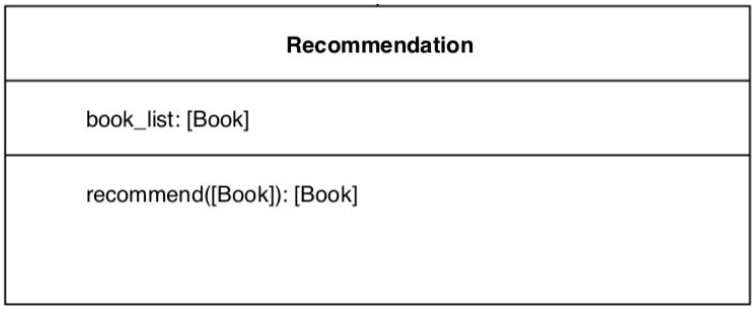
    - tag_library_load(): creates a dataframe of our csv file containing the tag data.

    - tag_locate(): creates a dataframe containing only the tags pertaining to the book in which this method is called.

- Algorithm class: the class used to call each of the others.



    - book_load([String]): returns a list of Book objects each for a book title searched.
    - recommend([Book]): uses a list of Book objects to return another list of Book objects, a result of the recommendation.
- Recommendation([Book]): the class used to return a list of recommended books.



    - recommend([Book]): the method used to return a list of recommended books.

Traceability Matrix

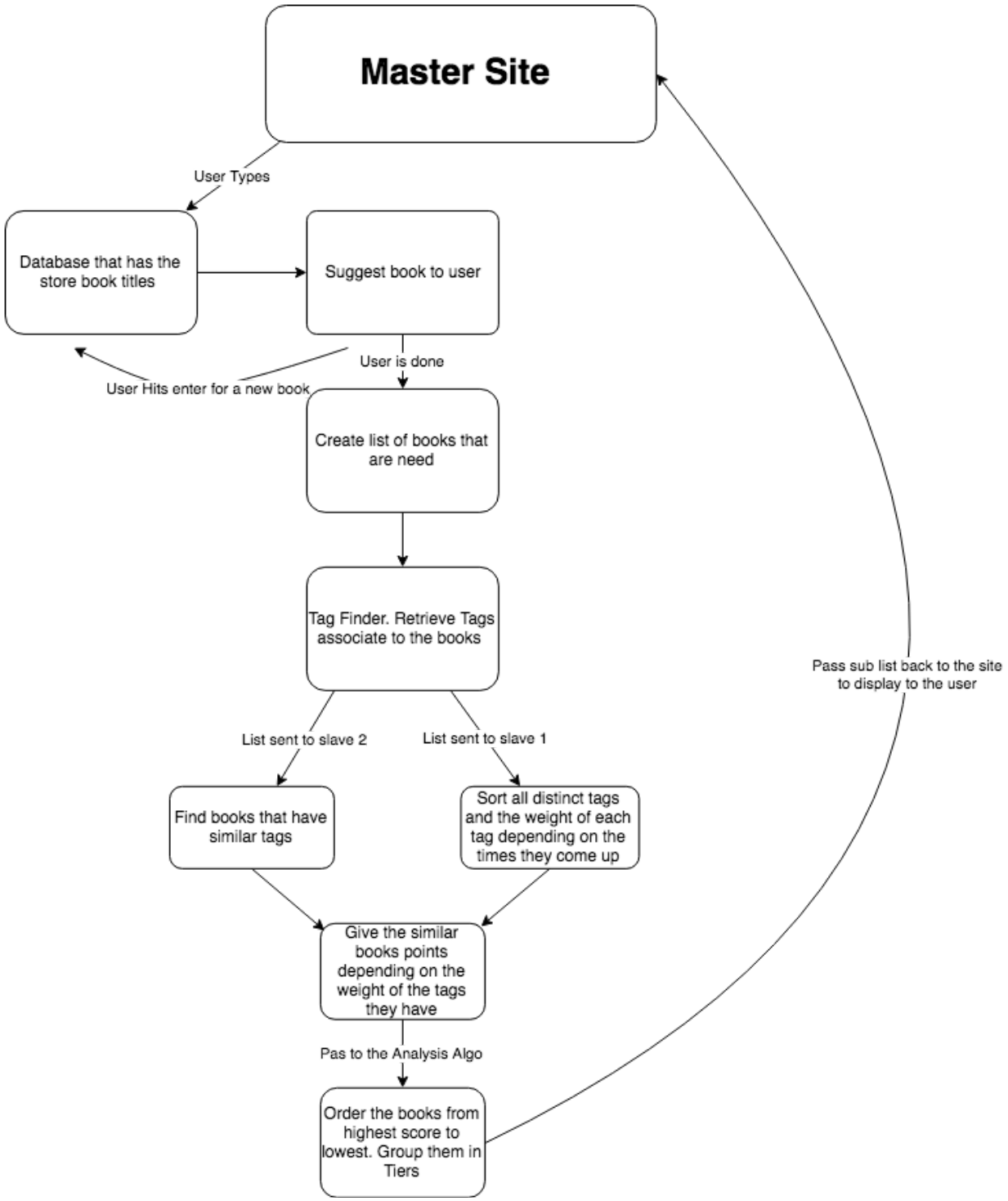
Class	Method	Identifier	P.W.	Definition
Recommendation	recommend([Book])	**REQ-1**	**2**	Use a breadth for search for optimized time
Tag	tag_locate(int)	**REQ-2**	**5**	The tags will be organized in a link list type of fashion that allows for easier more organized traversing.
Recommendation	recommend([Book])	**REQ-3**	**4**	Account for all distinct tags that associate with all the books entered
Algorithm	book_load([String])	**REQ-4**	**4**	Create a new table every time a new user enters the list of books (New user just means a new list. Users are not recorded)
Recommendation	recommend([Book])	**REQ-5**	**5**	Spreadsheet will hold a list of books that have tags that are matching the distinct tags list
Recommendation	recommend([Book])	**REQ-6**	**5**	The list will include all books that have at least Age and Genre matching.
N/A	N/A	**REQ-7**	**1**	(Optional) Delete the last sheet to save memory allocation
Book	book_locate(Pandas.dataframe)	**REQ-8**	**4**	System will begin predicting user input
Book	book_locate(Pandas.dataframe)	**REQ-9**	**2**	Update prediction with ever letter input
Recommendation	recommend([Book])	**REQ-10**	**4**	The spreadsheet list will be organized via point system to rank them
Recommendation	recommend([Book])	**REQ-11**	**4**	Ranking will be broken via percentage tile of the highest point in that list 95%+ =S, 90%-94%=A, 80%-89% =B, 70%-80%=C
Recommendation	recommend([Book])	**REQ-12**	**2**	System will have a short description about the tier in the box located next to the letter
Algorithm	book_load(String)	**REQ-13**	**4**	List should hold tags about the books

The classes we currently plan on using are a Book class, Tag class, Algorithm class, and Recommendation class. Originally, We believed that we would only have a single object class for each book (Book class) but quickly realized that having a subclass (Tag) which was contained inside book allowed for more detailed and descriptive objects.

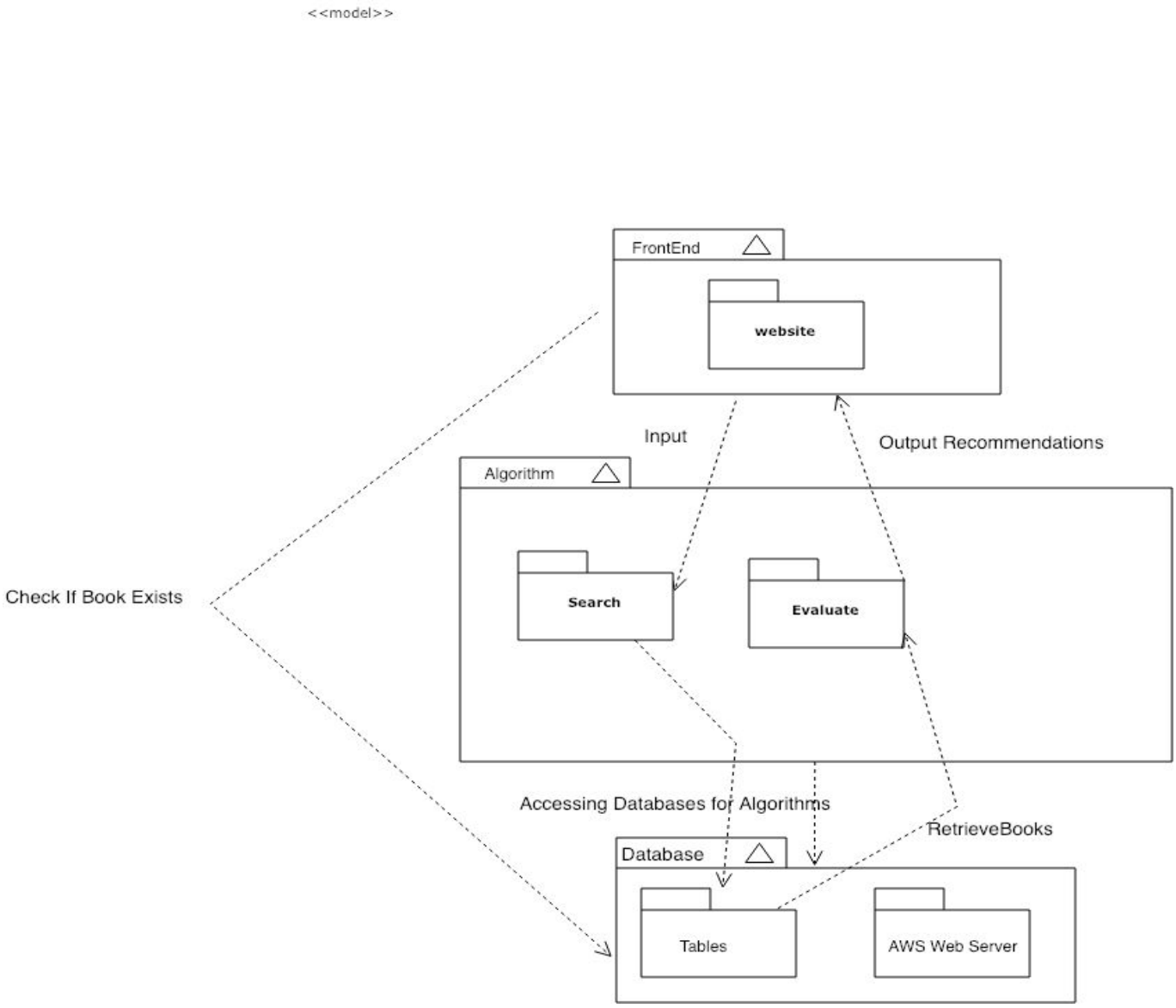
System Architecture and System Design

Architecture Styles:

We basically used two types of architecture styles. The main style that we used would have to be the Master-slave style. The master in our case is the site itself that takes the user input and thus the main type of data that we need. After that, the data is shared among the slaves (Tag retriever, Score giver, Sorter, Grouper). When data gets sent to the slaves, there are layers and order in which data goes through to reach the final result that we want. That is the layer portion. We also have a version of Client-Server pattern which is used for out input front end system. As the user puts in the book that they read, the data will be sent to the database and begin searching that book. Then that data will come back and present itself in the form of a dropdown menu.


Master Site
User Types
Database that has the store book titles
Suggest book to user
User is done
User Hits enter for a new book
Create list of books that are need
Tag Finder. Retrieve Tags associate to the books
List sent to slave 2
List sent to slave 1
Find books that have similar tags
Sort all distinct tags and the weight of each tag depending on the times they come up
Give the similar books points depending on the weight of the tags they have
Pas to the Analysis Algo
Order the books from highest score to lowest. Group them in Tiers
Pass sub list back to the site to display to the user

Identifying Subsystems

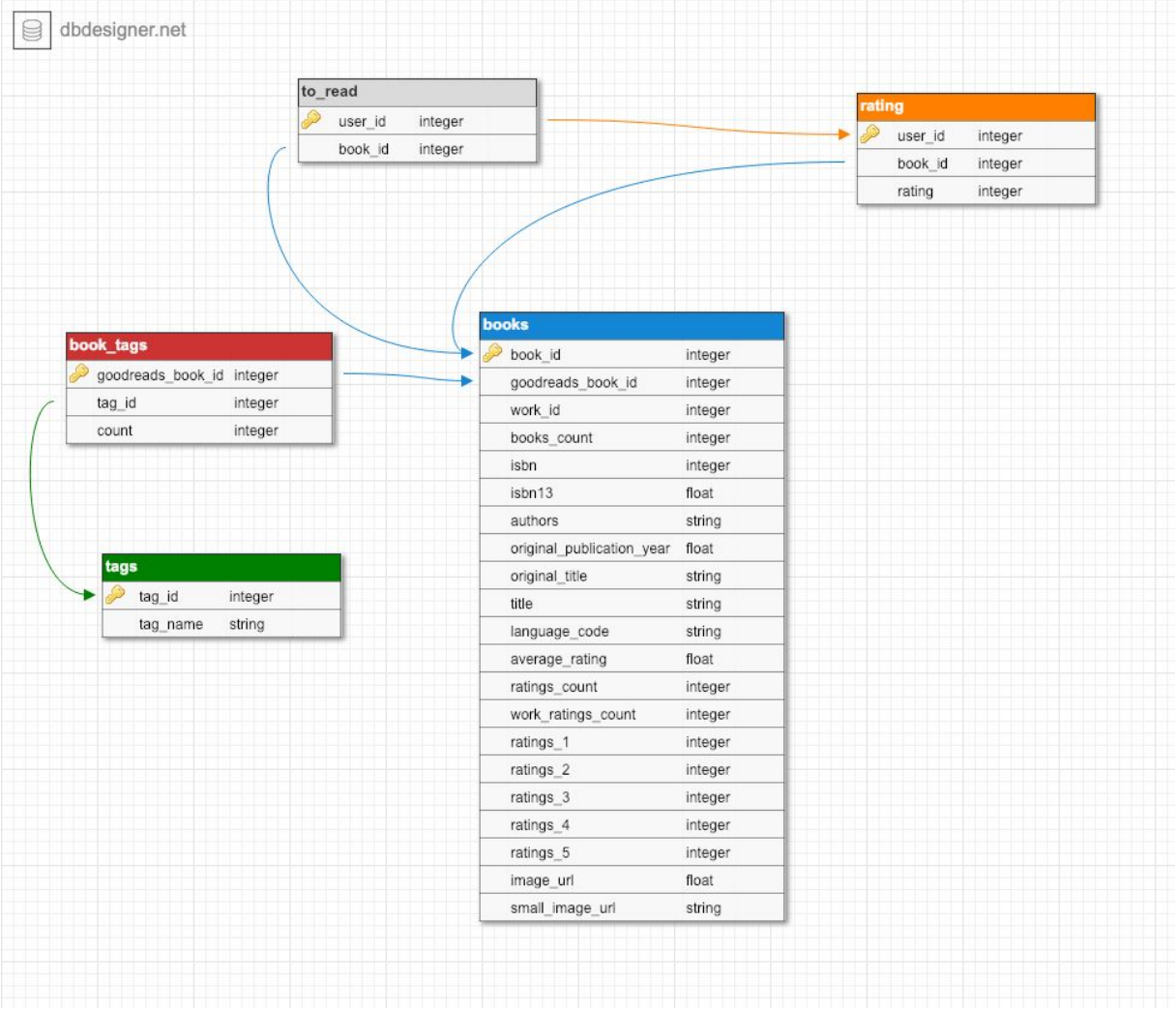


Mapping Systems to Hardware

Our system is expected to run on multiple computers since our book recommendation is deployed on a web-browser through AWS (Web Server). For frontend, we are implementing using Bootstrap to make a user-friendly and dynamic site. From the frontend site itself, the system is accessing the database to check for book's existence and if it exists then the system accesses the backend part to run the

recommendation algorithms. The Search subsystem accesses the database and finds out all the similar books to recommend. At database level, the system communicates to Evaluate sub-system to display the recommended users on the frontend to the user.

Persistent Data Storage



Global Control Flow

Execution Orderliness:

Our system is event based since it waits for user's book title inputs before searching the appropriate books for the user. Different kinds of book genres, titles, author relevances, and other

various attributes will generate different types of recommendations. Therefore, the whole functioning of the system is driven by what the user inputs.

Time Dependency:

Our system has no time dependency since we are updating are not updating the book database.

Concurrency:

No, our system does not use multiple threads, but if we have time to finish the main program then, we would implement multi-threading to optimize time for faster screen time execution.

Hardware Requirements

- We are using a dynamic website being built with bootstrap framework and Python based back-end code. So the minimal RAM that a user may fluently use our website with, is 2GB of RAM. This may change if our databases grow to substantial sizes or our website becomes more complex. But at operational levels 2GB of user RAM is definitely enough
- We are using cloud storage, which will ultimately be our host servers. So their processors used, and storage sizes in their server centers aren't directly linked to us. But due to cloud processing, the user must have a processor capable of 2 gigahertz (2 GHz) frequency at the minimum, and must have at the most 40 GB of available disk drive space (which is a very low amount in practical hardware) and minimum 2GB of hard disk space.
- We require monitor resolution of 1024 X 768 or higher, which is the minimal standard for most website specifications today, therefore should not be an issue for users.
- Our network is being hosted by a cloud server as mentioned above, so is requires a particular user network bandwidth, and must be capable of data transfer (measured by data transferred monthly). Data transferred indicates how much data can be transferred by our website, and the bandwidth indicates how fast the data can be transferred, both with are dependent of a users speed for minimal website response (aka there is a minimal required user bandwidth in order use the website in its slowest form). For websites who do not engage in streaming activities and do not have large media files (like ours), 10 GB+ per month, as 10mbps will be in the range of our operational bandwidth, making our minimal user bandwidth requirement at 4kbps (average speed of dial up internet, maximum being 56kbps for dial up internet).

Algorithms and Data Structures

Algorithms

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

For the recommending algorithm, we used a common similarity algorithm called the cosine similarity algorithm. It essentially maps all the genre and author tags contained in our database on a vector coordinate system. To see how two or more books may be similar to each other it measures the angle between the two vectors and then uses the equation above to measure how close they are two each other. If the books are highly similar the value will be close to 1, but the farther away the book or vector is the score will get closer to 0. To find the set of books that are similar to a set of books inputs, we find the matrix of similarity scores for each input book and then add up the all the scores. We then only have to sort it and give back the top results that are the most similar. This seems like a lot of steps, but using the pandas library in python a lot of the brute math is done for us using the built in methods.

Data Structures

For our data structure we are using Lists from python. The reason that we are using lists is because lists provide us with the ability to add elements as we go. This feature is very useful due to the flexibility it allows since part of our algorithm searches the books entered, and collects all the tags as they go through the books. The flexibility plays a part in the role of the structure because the user can input an infinite amount of books. Secondly, with the use of list we are able to combine multiple lists together in order to create a resultant pseudo hashtable. For example, if we want to see how many times a certain tag was found; we could have one list for the tags generated from the data of each book, then a separate list for the count of the occurence of each tag. After the data from the user ceases to grow any further, we can simply combine the two lists into one list that has (Tag,Count) as their elements. This makes it much easier for us to

visualize the data and go through it. The main idea of the approach was to emphasize on flexibility due to the large amounts of data we have, and how vulnerable these data is to change.

User Interface Design and Implementation

From the mock-up, it was originally planned to display the books with their images on the homepage of the site. However, on the home page the only features that will appear on there are just the search bar and a horizontal menu which consists of a genre button and a favorites button. The genre button gives a dropdown of all possible genres for books and the favorites gives a dropdown of the user's favorite books. The horizontal menu will be positioned at the top and the search bar is centered in the middle of the home page.

Design of Tests

Test Case Description

a. The test cases for our program are fairly simple because we just have to input in various books ranging from genres that are related like Harry Potter and Lord of the Rings, to books that are different like Romeo and Juliet and the Hunger Games. We will also input incorrect spelling of titles, null values, and books that do not exist. (enter some example test cases if you can)
b. The main idea of the list data structure is in order break down the describing characteristics of our user input, and put all the information of the inputted tags all in one place (a different list). So the testing for this data structure was fairly simple as we only needed to make sure that the number of times certain tags came up in our inputted list was recorded. So being as that we already have developed a work-in-progress database, we used some of the books from that database in order to test different quantities of inputs from a user, and made sure that the list accurately batched up all of the data from those inputs accurately.

Test Coverage

a. The test coverage of our tests is mainly for the front end because by the time the inputs arrive to the backend to be used in the algorithm they should already be preprocessed to make sure the spelling is correct, the books exist, and the correct numbers of inputs have typed. The front end is in charge of telling the user to keep on entering the correct inputs until the desired format of inputs is correct. At that point we can assume, the program will return the recommended books without any problem. Like any recommendation programs, depending on the inputs sometimes you may get back books that are actually similar to the books you inputted, but if you input a bunch of drastically different books the program will have a hard time returning back similar books for you to read.

b. Testing for the backend will require more effort. Using a sample set of books and estimating return values, we can adjust our algorithm multipliers (for different tag types) in order to return books that are most like the books we've inputted. With repetition we should be able to enter any list of books and receive a list of books which has strong correlation to those entered.

Integration Testing Strategy

a. Our integration testing is going to be focused on multiple book input and ensuring that having more than one changing variable will not drastically affect the outcome of our product.
b. We are going to show that our system can take infinite book inputs and generate the recommended output almost instantly. For this, the length of book inputs is until the last input added in the search bar (of our front end site) until the Search Icon is clicked.
c. We are going to show that the system is able to detect spelling errors and predict an appropriate book name based on the list we have in our database. For this, we will intentionally make some spelling mistakes for some book of our choice and show that the suggestion matches the input we are looking for.

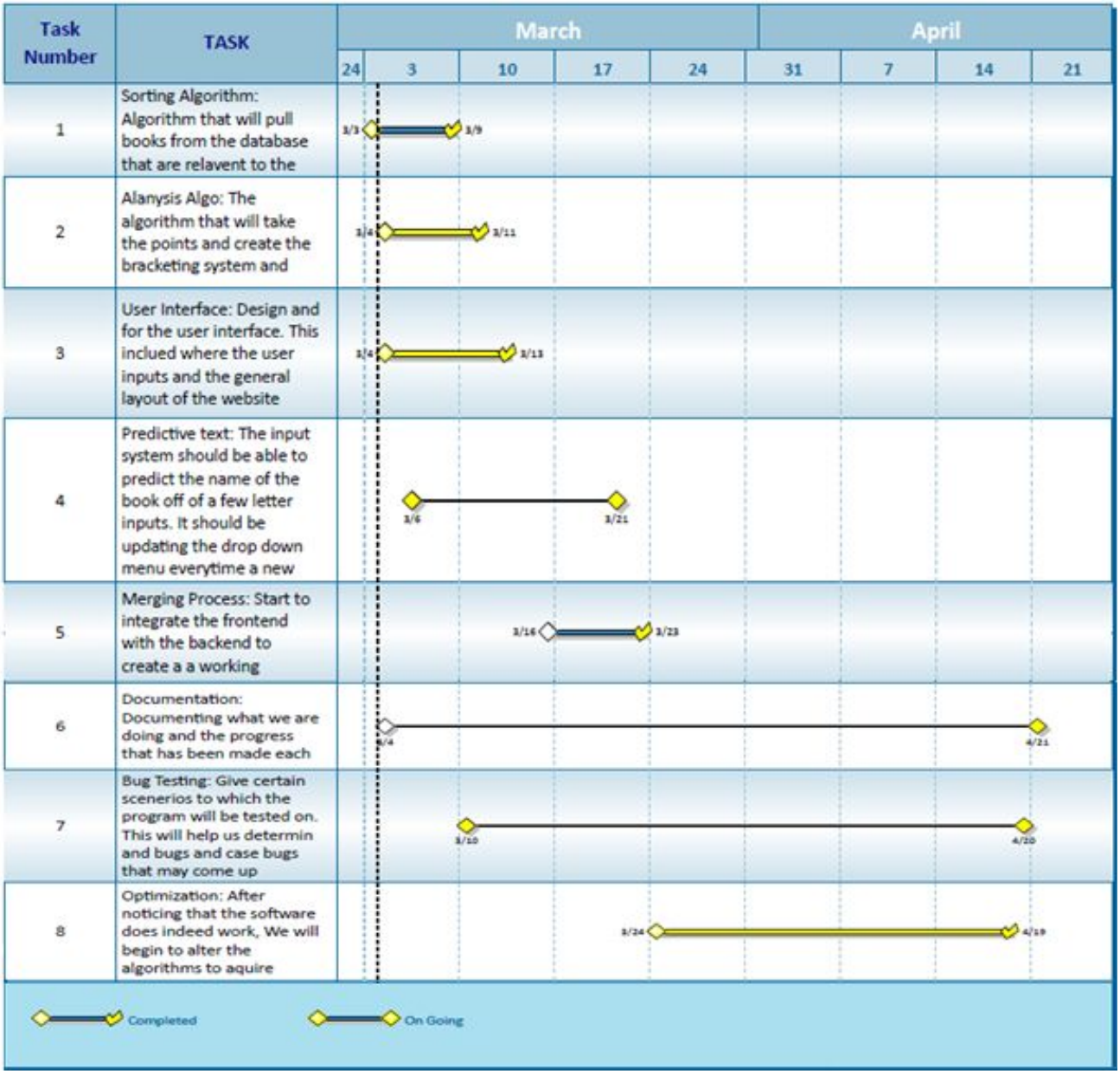
Project Management and Plan of Work

Starting from backend programming, it was a challenge for us to figure what source library to refer to that could possibly have a vast variety of books in its database. Eventually, after referring to some book service sites, we came across Goodreads and referred to its book libraries and added them to our database. For frontend, we were able to outline a design for how it will look to user that visits our site. At every search, the system is suppose to take in all the book inputs and store the output of recommended books in the database and clear it during the next search. However, a challenge that exists whether there will be erroneous data produced if the site is accessed from multiple devices, generating wrong outputs for users. Therefore, we were researching ways to restrict the system functioning to unique devices so that the one person's inputs do not generate outputs on other people's devices.

Project Coordination and Progress Report

For the frontend, the bootstrap has been initialized in that the html file contains a search bar for the user's input and a search button that can be clicked on. When clicking on it, the animation shows that it can be clicked on. None of the use cases have been implemented yet because the algorithms are still being worked on.

Plan of Work



Breakdown of Responsibilities

Backend: (Algorithm, Book, Tag): Vedanta, Shazidul, Alan
Frontend: Anthony, Akshat, Joel
Recommendation (Recommendation): Joel, Shazidul, Vedanta
Product/Stress Testing: Kutay, Avani
Documentation: Avani, Kutay

Integration Coordination:

- Akshat, Anthony and Joel will coordinate the integration of the backend algorithms with the front-end site functioning

Performance and integration testing:

- Kutay and Avani will be testing out implementation and outputs of use cases and multiple checkpoints, along with debugging sub functions and sub classes

For the breakdown:

Our team first divided into 3 subgroups in order to focus on each subdivision of the project. We divided the project into frontend, backend, and bug testing/ documentation. As Akshat, Anthony and Joel worked on the frontend part of the project, including the UI development and appearance of the website, along with the tier development system, Avani recorded all the information and helped communicate it to all the backend people working on the algorithm. The backend developers including Vedanta, Shazidul, and Alan all worked to get the classes developed as well as set up the database for the project. Avani and Kutay sat with both groups and debugged the issues that had come up while the algorithms were in the early stages. They went through each of the 10,000 lines in the excel database and checked and fixed errors where many characters either were not readable words or were just complete gibberish. Kutay relayed information from the backend group to the frontend group about progress updates and what the future plans were going to be before the integration aspects would occur.

As of right now, the contributions made by every team member are pretty equal in terms of how much work was done. Those who were not as proficient in coding took up other tasks while proceeding to learn the coding language as best as they could in order to help with the bug testing aspect of the project. Avani and Kutay were the main documentation people and took some weight off the other members who are focused on doing the frontend and backend processes. While having meetings every week kept the group focused with the task at hand, smaller groups also met to work on their parts of the project aside from the meetings. Vedanta and Shaz had done individual work at home and brought what they had finished to the meetings which was the bulk of the code for the searching algorithm. They updated the github repository as soon as they finished so that everyone would be able to see what was being done. As a whole, everyone in the group had equal participation and no one had been singled out for being a burden to the group. Everyone helped in every way that they could have and we all are on schedule because of that.

- Coding - Shazidul and Vedanta will code the database and backend communication. Alan will structure the database and dynamically code it for new data entries

- Development -Akshat will develop the Predictive Text search bar and Anthony will develop other frontend site functioning with respect to the backend development
- Testing - Kutay and Avani will test input for UI functioning and debugging

Name	Did	Currently	Will Do
Shazidul (Analysis Algorithm)	Learned SQL and Python for compatibility Documentation	Working on Analysis Algo and sorting methods Documentation	Merge algo with database and site Documentation
Vedanta (Analysis / Sorting Algorithm)	Created Github repository. Documentation	Working on backend algorithm. Documentation.	Complete algorithm (backend/analysis) and merge with database
Avani (Presentation and UI design)	Documentation and Group Meeting Coordinator	Documentation and Group Meeting Coordinator.	Documentation and Group Meeting Coordinator Will add input for UI
Anthony (Predictive Text)	Contributed concepts for the functionality.	Working with front end for site development.	Will merge algo with site (he is the bridge between both)
Alan (Database Entries)	Database creator (local csv, static).	Maintain and organizing database	Will maintain and update database on new entries
Akshat	Learned SQL and Python, gained understanding of Bootstrap and AWS	Currently working on front end site.	Have predictive text up and merge with algo
Joel	Research for point/tier system. Documentation	Working on sorting and selecting algo.	Help merge the algo with the analysis algo and the site
Kutay (Presentation and UI design)	Documentation	Documentation. Learning programming languages to help with bug testing. Cleaning up csv.	Documentation and bug tester