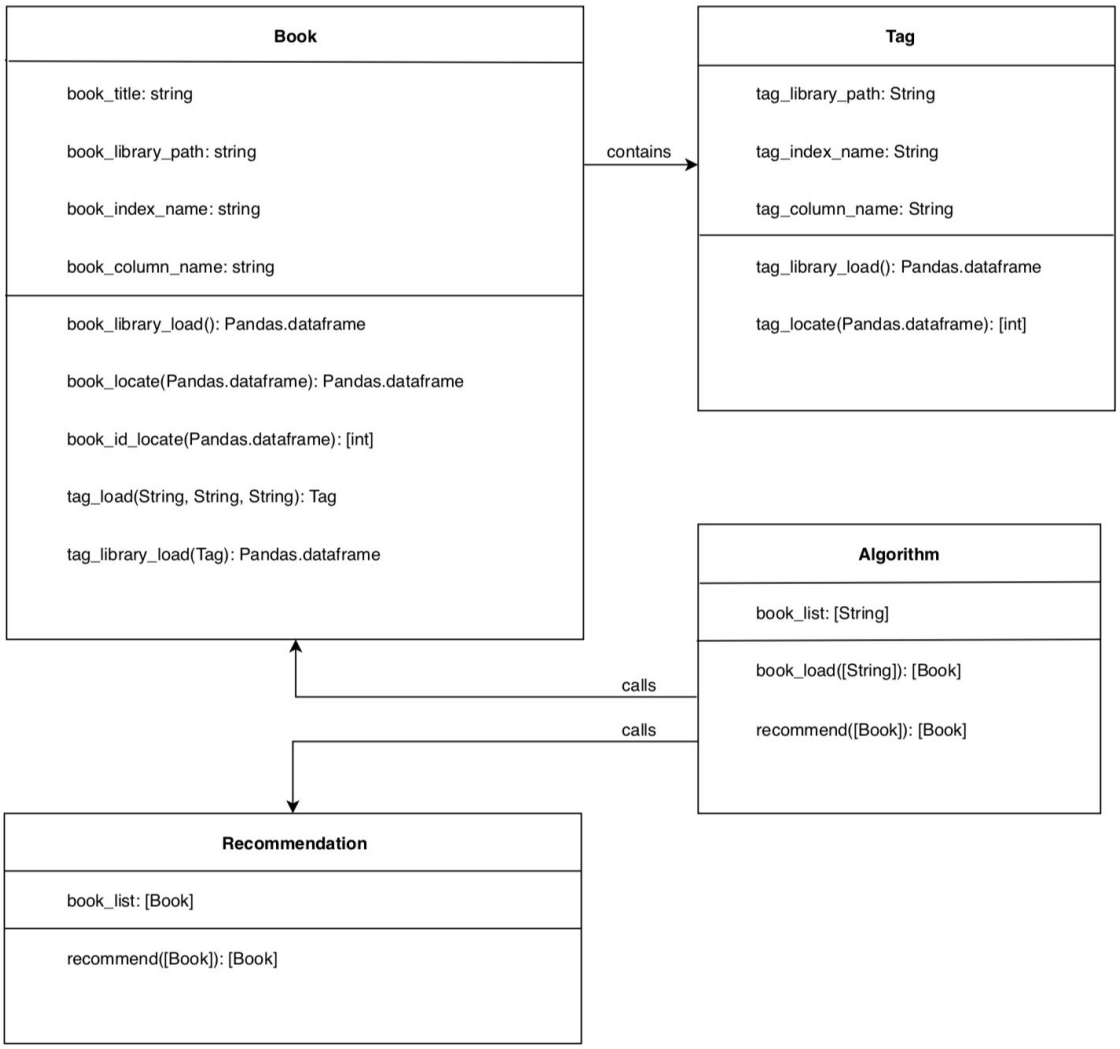
Group Members:

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

Project Title: Recommending Alpha
Group Number: 16
Date: March 10, 2019
Report 2 Part 2

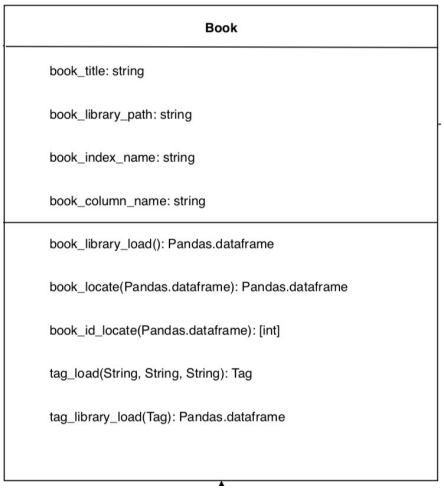
Class Diagram and Interface Specification

Class Diagram

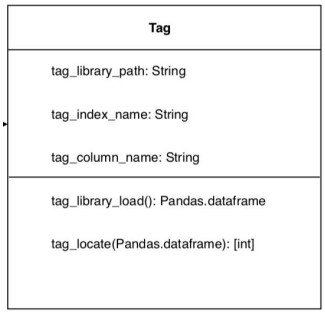


Data Types and Operation Signatures

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

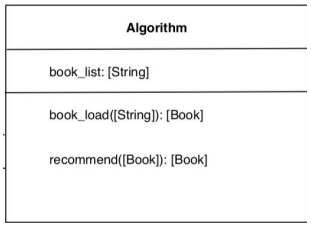


  - book_library_load(): creates a dataframe of our csv file containing the book data.
  - book_locate(Pandas.dataframe): creates a dataframe containing only the searched books.
  - book_id_locate(Pandas.dataframe): creates a list of book_ids pertaining to each book searched.
  - tag_library_load(): creates 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.

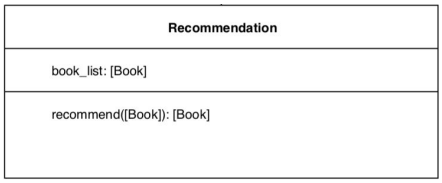


  - tag_library_load(): creates a dataframe of our csv file containing the tag data.
  - tag_locate(): creates a dataframe containing only the tags pertaining to the book in which this method is called.

- 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

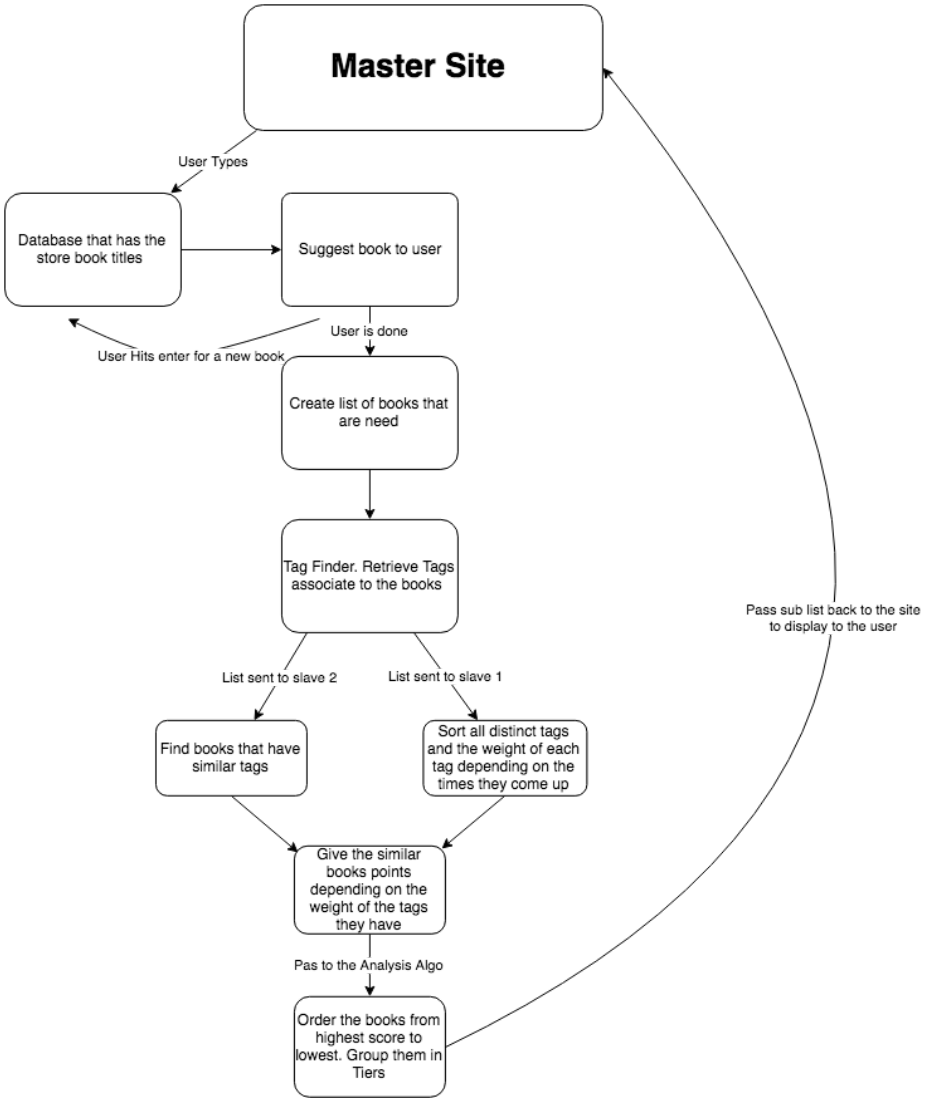
Class	Method	Identifier	P.W.	Definition
Recommendation	recommend([Book])	**REQ-1**	**2**	Use a breadth for search for optimized time
Tag	tag_locate(int)	**REQ-2**	**5**	The tags will be organized in a link list type of fashion that allows for easier more organized traversing.
Recommendation	recommend([Book])	**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.

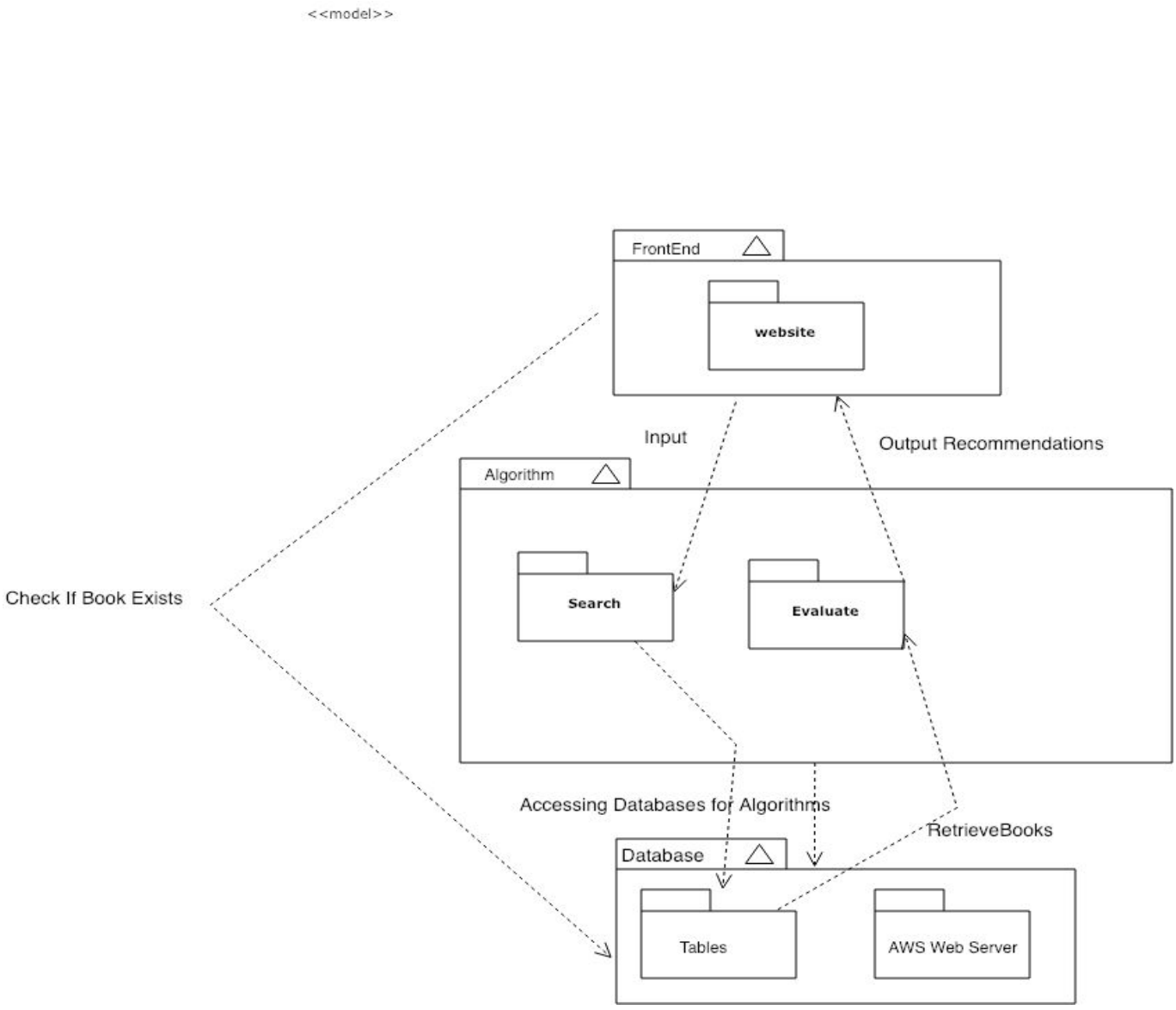
System Architecture and System Design

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.



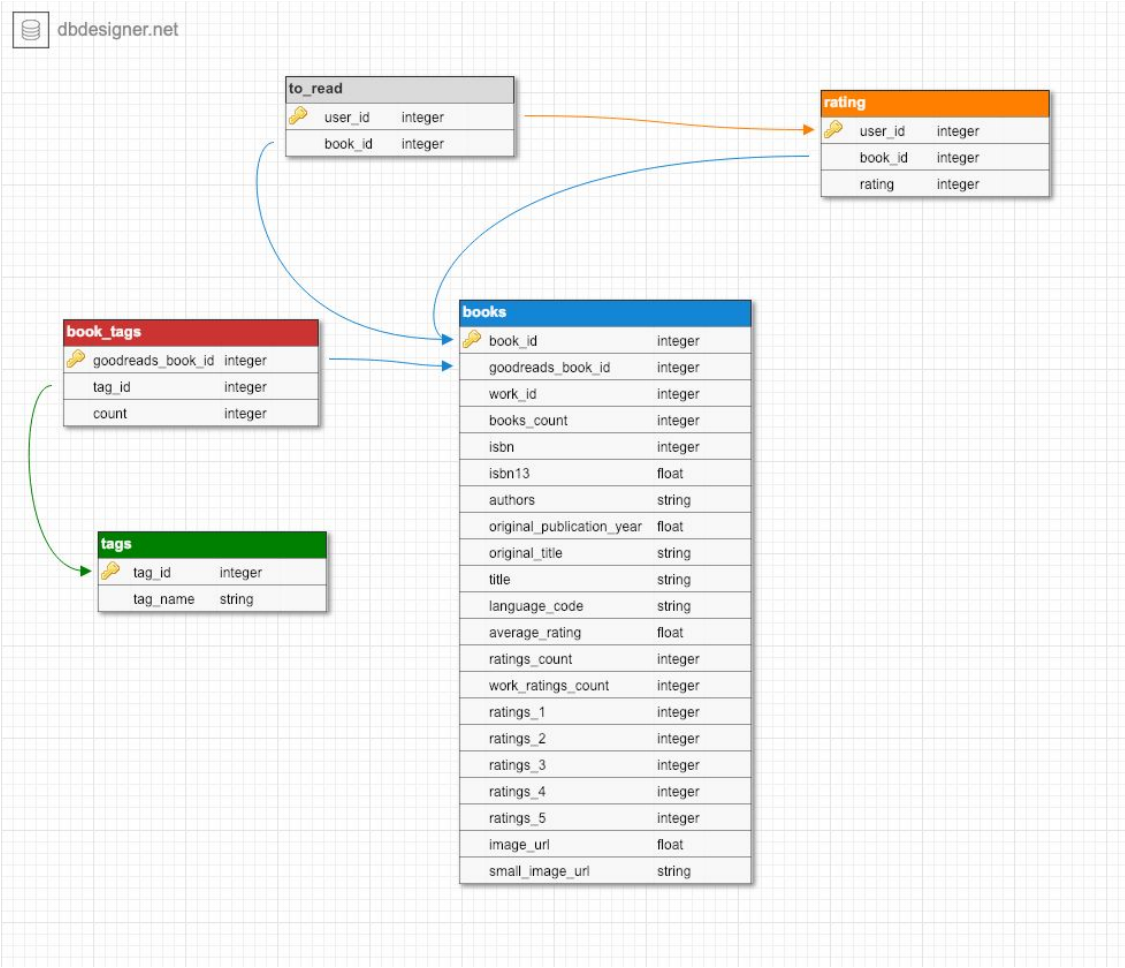
Identifying Subsystems



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).

Project Management and Plan of Work

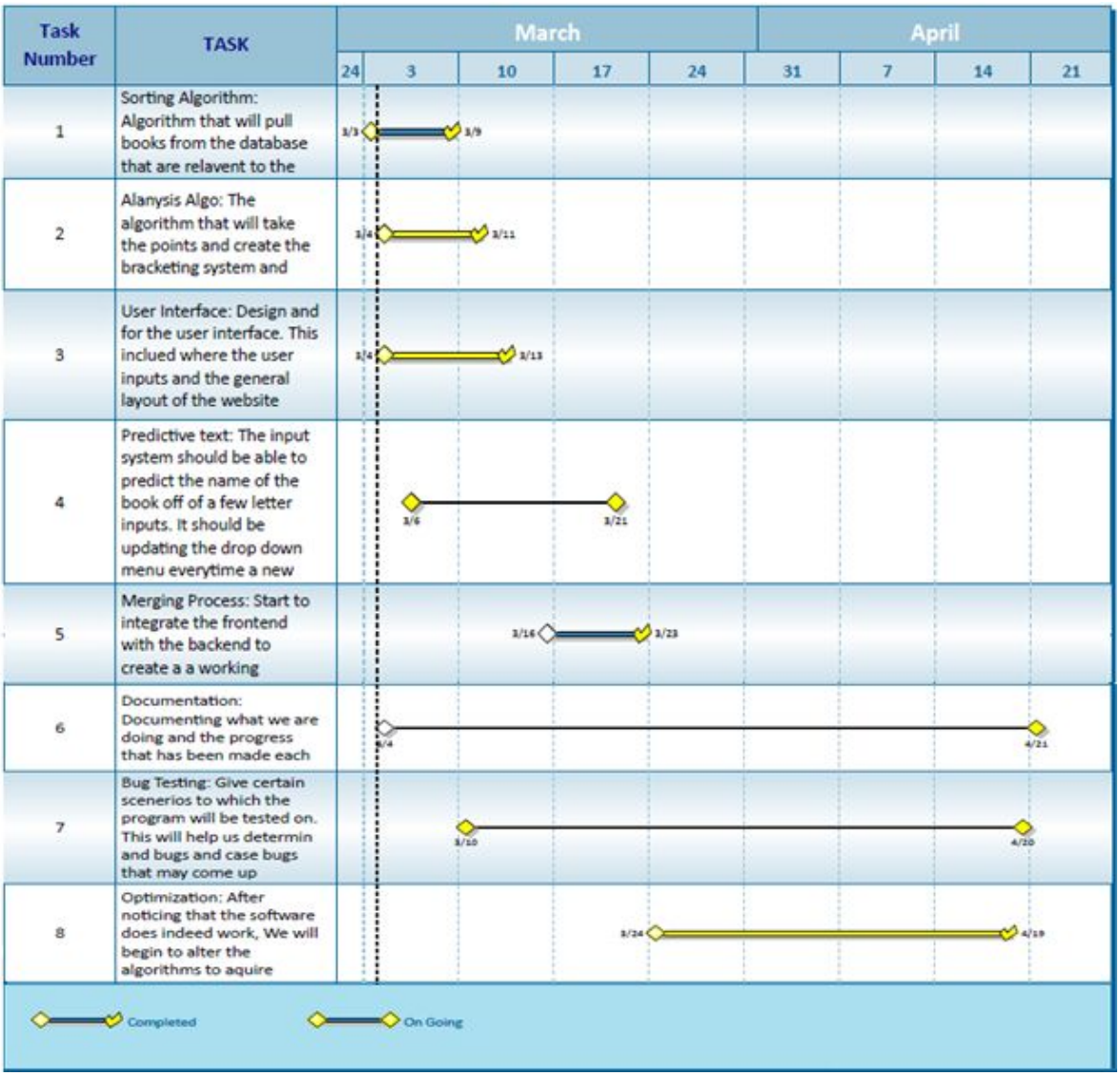
Starting from backend programming, it was a challenge for us to figure what source library to refer to that could possibly have a vast variety of books in its database. Eventually, after referring to some book service sites, we came across Goodreads and referred to its book libraries and added them to 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

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