Recommending Alpha

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, documentation
Akshat:	C++, Java, Python, SQL, design, documentation

b. **Team Leader:** Vedanta Dhobley

Table Of Contents

Statement of Work and Requirements

1.1 Problem Statement

A current issue at hand is that most “Recommendation” programs for books simply search with only author or genre tags when looking for books at libraries or bookstores. This leads to titles being recommended that are not similar to the person’s liking or all recommendations which have one very odd tag match but don’t have any other similarities to the first.

a. Example: If someone were to say they read a disney book, what can happen is that the recommendation algorithm will search for similar genre or author. However a better search would be Disney tag. Disney would return more suitable recommendations for the person that read something that was Disney related.
b. Another Example: Many who put that they have read Harry Potter, seem to get back a recommendation of Twilight due to it falling under the fantasy genre. However a series such as Percy Jackson would be a better match due to its use of magic along with fantasy, which is a more precious and intuitive approach.
c. Another bad recommendation would be when differentiating between what categorizes as ‘historical fiction’ and a simple history documentary. For example, a student who is trying to write a history paper on the roaring twenties prior to the Great Depression, they would most likely want to find articles and books with facts about this time period, instead of receiving a search result of ‘The Great Gatsby.’ Although the Great Gatsby was written in this time period, the novel is not focused on the general historical value. It was written as a fictional novel during that time. A student doing a report on American history would probably not want to waste time reading ‘The Great Gatsby’ because of the fictional aspect. The implemented algorithm would be able to differentiate between ‘history’ and historical fiction’ in order to give the user a better search result with books of the appropriate tags.

The target for this program would be to entice avid readers that have issues with finding books similar to their taste, or those that simply do, can’t decide that their taste is. With this algorithm, these readers can narrow their searches and find material that is much more relatable to the genre of books they have interest in. There is no age target since everyone at any age can enjoy reading!

1.2 Proposed Solution:

For the program, there will be two algorithms, which are ‘Recommendation,’ ‘Analyze,’ and a string search being implemented. In order to implement the first algorithm, this search must pass correctly. The string search does not assume that the user will type the book name in correctly. The data, which are the books, will be kept in a database. This search ensures that if the book is in the database, there will be a dropdown menu allowing the user to choose from the options that are in the database. If the book does not exist in the specific database, there will be no search results available. For the tags, the way the search function will work is by using a breadth first search. For this to work properly and most efficient, it needs to be made so that the tags of every book is in the same order For example, Age → Genre → Author → Length → etc. Creating them in a link list formation makes it easier to actually go through the heap and find everything needed. As the developer runs through the tags, it starts tallying the total number of points and then record that next to the book. The breadth first search will be used for optimized time. The tags will be organized in a link list type of fashion that allows for easier more organized traversing. Account for all distinct tags that associate with all the books entered

This also works well for people who are consistent at misspelling words. If words are misspelled, they will not be able to find the book that is actually available in the database. Therefore, having a dropdown menu that starts giving the user recommendations as they start typing the book will make it more convenient for the user to find the book rather than be unhappy when a search is unavailable due to a spelling error. Essentially, there will also be an assumption that the user is searching for books that are already in the database, for the books that are selected from the dropdown menu. For example. if someone were to try to find a book in a different language, this search would return no result because that specific book would not be able to be located in the database.

The program will allow multiple users. To allow this, the program will create a new spreadsheet for a new user. That spreadsheet will be used to store the information about the books entered as wells as the books that we pulled from the database that has the other books. This will help with sorting later down the line. As the user is inputting the names of the book, the system will being to predict what the use is trying to input. This will save the user time and thus increase satisfaction. Not only this but because the data is case sensitive, this will minimize errors and discrepancies.

For the second part, there will be a ‘Recommendation’ algorithm. This algorithm will work with the tags we put on the books. Each tag will have a set multiplier (1x, 5x, 10x, etc) added to the books. For example, the ‘age’ tag will have a x50 or so multiplier to ensure that the recommendations that are coming back are in the same age limit of the user/reader. ‘The Lord of the Rings’ would definitely not be a good recommendation for a 5- year old with a very limited vocabulary. Another example is a tag for books written during certain time periods. A modern millennial may find a book with a few similar tags, but when he or she sees the tag that determines what time period it was written in, he or she may not want to read a book that was written such a long time ago because he or she may not be familiar with the vernacular of the time. This is also a useful tool for parents who do not want their children to read books with a much higher maturity level than they can comprehend. A big concern for parents these days are whether or not a book is fit for a child, especially if they have no knowledge about what the book is about. A kid may want a book on monsters, but would probably not be to happy reading a book that would give them nightmares because of a book that was very explicit and too gruesome for even the most avid young readers. This recommendation algorithm will recommend books to users without typing in specific keywords, which is essentially the whole point of the algorithm.

Once the search is done, the results will be displayed in tiers. These tiers will be based on a statistical analysis of each book's tags. The more similar the tag, the more closely related the books will be. This would be considered as the second algorithm being implemented in the program. This 'Analyzing' algorithm uses the points allocated to different books based on our Search Function, using the data for statistical analysis of the results. Using this analysis, we can group our results into tiers (S, A, B, C from highest to lowest) based on standard deviations. This allows the user to see not only which books are recommended to them based on their inputs, but also how how closely related each result is, giving a more organized and focused list of results. The S tier books will contain books in the top 0.5% , A tier books will contain books in the next 2%, and so on and so forth. Using a standard deviation to split the tiers, we will decide a baseline score required for each book to earn a spot on the list of recommended books. This is the most optimal way for organizing the books for the user because it is easy to implement on a user interface and will be convenient for the user to see the tiers and maybe just have to decide between 5 or 10 books instead of 50+ books.

The most important feature in this program would be the recommendation algorithm. This algorithm needs to identify the tags on each book in the database in order to return the tiers. It will group the books that have the most tags in common, and list them after the user selects the book from the dropdown menu. The tags can include things such as genre, age level, "Disney", number of pages, difficulty of vocabulary, etc. The database is required to have these books with these tags in order for the search to go through. The database that will be used will be an open source library with about at least 50,000 books that will be searched through for their tags.

For the user interface, there will be an easily accessible website to use. So every input, whether it be from a input device or a smartphone screen, will be handled through features of the user interface and will essentially allow the program to perform its job through user demands. The list of books that have already been read will be displayed horizontally. Below this list of books that are already read, there will be a section called "Top Picks for You." This section will be based off all the recommended books that the user has displayed interest in previously. There will be a trending section which lists all the new books that are based off popularity and reviews. When scrolling through each book, there should be a brief summary of the books, the ratings, and the genres. Any additional lists will have other genres for easy access to a book that is **not** part of the user's preferred genre. For example, these lists may include: comedy, horror, romance, drama, etc. However, only a few genres should have their own list at this point because if not, users would have to keep scrolling to find their preferences which may end up ruining their experience on the platform. As previously mentioned in the search string implementation, the drop down menu can also be used to have users explore further. In any list, the image of each book will be its cover so that the user can easily identify the book without any difficulty. every input, whether it be from a input device or a smartphone screen, will be handled through our features of the user interface and will essentially allow the program to perform its job through user demands. There will need to be a way to check the algorithm consistently after it is implemented. The database will constantly need to be refreshed as new books are added to the system. As the database is updated, the tier list will change for every input that is added in. If more books are added, more tags will also be added. This algorithm should be able to take this into account when the search is being done by the user.

The on-screen appearance aspect of this project is where most of the front-end design and development will be hosted. Along with the front-end work, this is also where the user interface aspects of this project will reside. The front-end design and development will drive how the visuals and website aesthetics will not only look, but also operate accordingly to the functional demands of the backend partition. All of the visuals of the website and user input will be held and driven through the on-screen requirements. The on-screen appearance aspect of the project will also be where the user-interface will strive. So every input, whether it be from a input device or a smartphone screen, will be handled through the features of the user interface and will essentially allow the program to perform its job through user demands.

There are a few main goals that need to be accomplished by this project. First, there needs to be a well made interactive website for the user to connect with the database. This database will need to be managed by a developer who updates the database every two weeks or so to make sure that the latest trending novels are included in the database. The website should be user friendly, and look the part. It should be easy to navigate through and easy to read. Next, there should be random checks to make sure that the tier list that was created is up to date with the database. There should not be books left out when more are added into the system, and each books' tags should be registered into the system as well. As more books are added, the searching string needs to make sure that it takes into account the added books. Adding more books into the database should not cause the algorithm to run slower. It should be optimized at the same speed that it was running on prior to the adding of new books into the database. The biggest challenge will be optimizing the runtime for this algorithm to run efficiently and quickly. Since there will be more than 50,000 entries, it might take a while to get a recommendation from the database which requires optimization. Getting the runtime to be efficient will require a lot of testing on a public database which will give good feedback on how efficient the algorithm is. Even after the program is launched, it should be checked on occasionally to make sure that the runtime is not affected badly at any point in time.

1.3 Glossary of Terms

Algorithm: A process or set of rules to be followed in calculations or other problem-solving operations.

Analyze: The algorithm that is used to separate the returned books into a specific tiers.

Books: The entries in the database.

Database: Hosted on the website, used to store the entries (books) and their tags.

Entries: Every single book in the database is an entry itself.

Feedback: Information about reactions to a product, a person's performance of a task which is used as a basis for improvement. For this project, the feedback would come from how well it works for users.

Tag: Words and short phrases that are used to describe a book.

Tier: Subsections which identify how close in relationship two books are. It includes a list of books that are related to the searched book.

Search: The algorithm that is used to find the user input from the database.

Recommendation: The algorithm that is used to recommend a book based on the tags that is associated with it.

User Interface: How the user and the computer system interact with each other. In this case, it will be a website that the user uses to search for their recommendations.

String Search:

Platform: The basic hardware (computer) and software (operating system) on which the applications are run.

Optimization: Bettering the algorithm so its runtime is faster and more efficient.

Multiplier: Associated with the tags. Each tag has a multiplier. They vary based on the importance of the tag.

Vernacular: The language or dialect that is spoken by people in a specific region or country

2. Functional Requirement & User Interface

2.1 Stakeholders

- User
- Administrator
- Bookstore Owners

2.2 Actors and Goals

- User
    - User will be an initiating actor.
    - The user's goal will be to retrieve recommended books from the database which match the search that the user has made.
    - Includes: Parents, Children, Avid Readers, School teachers looking for books for their classrooms, People finding books to buy for their friends.
- Administrator
    - Administrator will be an initiating actor.
    - The administrator's goal will be to ensure that people are satisfied with their selections. He or she will make sure that the database is refreshed to a more up-to-date version of the database so that their are more options for users to choose from. He or she will also make sure that the tags on each book are still placed appropriately for the new incoming books.
- Bookstore Owner
    - Bookstore Owner is a participating actor.
    - The bookstore owner's goal is just to allow the system to be implemented within his or her store. He or she will just take a look around to make sure customers have easy access to the website on the computers in the store.
- Librarian
    - Librarian is a participating actor.
    - The librarian's goal is just to allow the system to be implemented within his or her library. He or she will just take a look around to make sure customers have easy access to the website on the computers in the library.
- Website Developer
    - Website Developer is an initiating actor.
    - The website developer's goal is to keep the website up to date and refreshed with the latest books. He or she will check the frontend and backend parts of the system to ensure that the appearance of the UI looks the way that it should.

- Database Manager
    - Database Manager is an initiating actor.
    - The database manager’s goal is to keep updating the list of books in the database every two weeks or so. He or she will make sure that the books are equipped with the appropriate tags. He or she will notify the updated database to the website developer, the store owner, librarian, etc.

2.3 Requirements

2.3.1 Enumerated Functional Requirements

Identifier	P.W.	Definition
REQ-1	**2**	Use a breadth for search for optimized time
REQ-2	**5**	The tags will be organized in a link list type of fashion that allows for easier more organized traversing.
REQ-3	**4**	Account for all distinct tags that associate with all the books entered
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)
REQ-5	**5**	Spreadsheet will hold a list of books that have tags that are matching the distinct tags list
REQ-6	**5**	The list will include all books that have at least Age and Genre matching.
REQ-7	**1**	(Optional) Delete the last sheet to save memory allocation
REQ-8	**4**	System will begin predicting user input
REQ-9	**2**	Update prediction with ever letter input
REQ-10	**4**	The spreadsheet list will be organized via point system to rank them
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
REQ-12	**2**	System will have a short description about the tier in the box located next to the letter
REQ-13	**4**	List should hold tags about the books

2.3.2 Enumerated NonFunctional Requirements

Identifier	P.W.	Description
REQ-1	**3**	Database should be updated once week to account for new books and other updates
REQ-2	**5**	System must be able to handle infinite amount of inputs and display finite amount of outputs for accurate representations. This should not take longer than 5 minutes
REQ-3	**4**	System must display suggestions for spelling mistakes or if the inputs are not written exactly as the actual book name (even if few words are written, or should read 50 as the word “Fifty” in the book and vice-versa)
REQ-4	**5**	System should display an error message for invalid inputs, even if there are few valid list of inputs followed by an invalid on (“Try again” button that refreshes the system.)
REQ-5	**5**	System should prompt an option to start anew after displaying output
REQ-6	**4**	System must be able output the recommended book using the algorithm in a quick fashion (Again limit is 5 min)
REQ-8	**3**	Program should be able to handle multiple clients accessing the program
REQ-9	**5**	Application must be straightforward to use
REQ-10	**5**	System code must be broken down into manageable and maintainable parts
REQ-11	**5**	Must be able to run on web browsers without error
REQ-12	**5**	Output should be displayed only after all desired inputs are submitted

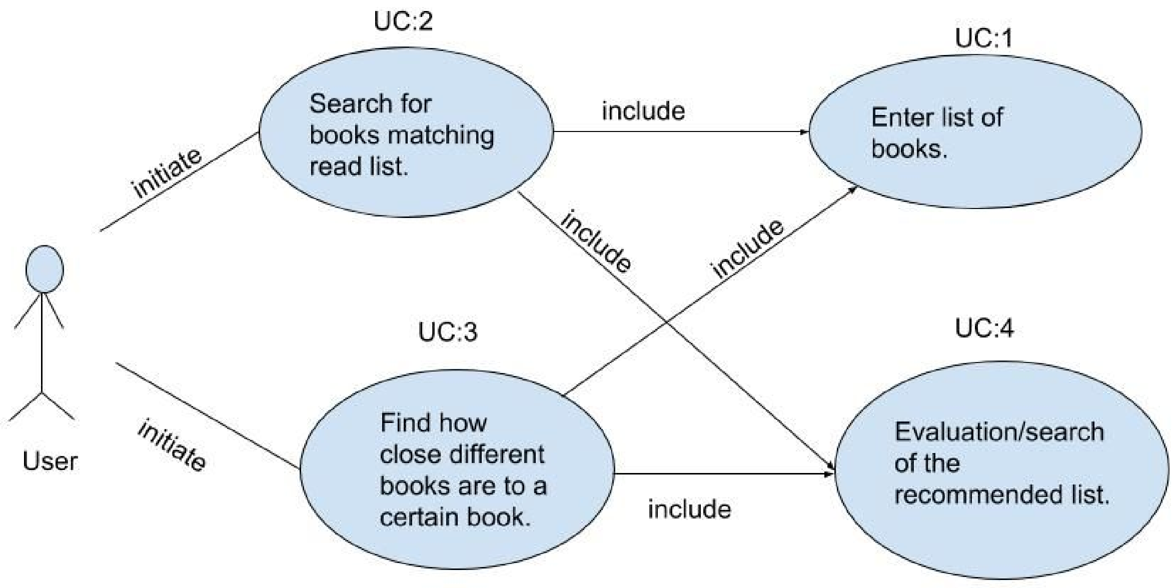
2.3.3 On-Screen Appearance Requirements

Identifier	**P.W.**	**Definition**
REQ-1	**1**	A list of books already read should be displayed horizontally.
REQ-2	**5**	Below the list of books already read, a “Top Picks For You” section should be dedicated to all the recommended books based on the user’s interests
REQ-3	**2**	A trending section must list all books based on popularity and reviews
REQ-4	**5**	When scrolling through each book, there needs to be a brief summary of the book, the ratings, and it’s genre.
REQ-5	**4**	Additional lists should have other genres for easy access to a book that’s not part of the user’s preferred genre. For example, such lists are comedy, horror, romance, drama etc. However, only a few genres should have their own list at this point. Otherwise, the user would have to scroll further down to see all genres which can ruin the user experience
REQ-6	**2**	A browse drop down menu should list all possible genres for users to further explore.
REQ-7	**4**	In any list, the image of each book will be its cover so that the user can easily identify it.

2.4 Use Cases with Casual Descriptions

1. **Case 1 (Entering the List of Books)**
    a. User→ Begins typing the book
    b. System← starts using predictive text, begins to suggest book name
    c. User→ select book which they are looking for.
2. **Case 2 (Find books according to their list of read books)**
    a. User→ Enter the name of book A into the search bar
    b. System← begins to find the name of the book in the database for predictive text
    c. User→ Selects the book from the predictive text
    d. System← Creates table that stores all the books that the user input
    e. System← Starts running the similar match algorithm, and then stores them in the same file with the books that were entered
    f. System← Use Analysis Algorithm to rank the books form S→ C
3. **Case 3 (Decide which book is a better match for a single book)**
    a. User→ Enter the name of book A into the search bar
    b. System← begins to find the name of the book in the database for predictive text
    c. User→ Selects the book from the predictive text
    d. System← Creates table that stores all the books that the user input
    e. System← Starts running the similar match algorithm, and then stores them in the same file with the books that were entered
    f. System← Use Analysis Algorithm to rank the books form S→ C
    g. User→ Use search function to find book B and C and see which book is more closely related to book A
4. **Case 4 (Evaluate Recommendations)**
    a. System← Returns a list of books that are recommended based on what the user inputed
    b. User→ Search/find books that they believe are good reads in their taste by scrolling through the list

2.4.1 Use Case Diagram


UC:2
Search for books matching read list.
UC:1
Enter list of books.
UC:3
Find how close different books are to a certain book.
UC:4
Evaluation/search of the recommended list.
User
initiate
initiate
include
include
include
include

2.4.2 Traceability Matrix

For Enumerated Functional Requirements

Requirements	Priority Weight (of Requirement)	UC-1	UC-2	UC-3	UC-4
REQ-1	2	✓	✓		✓
REQ-2	5		✓		
REQ-3	4		✓		✓
REQ-4	4	✓	✓		
REQ-5	5		✓	✓	✓
REQ-6	5		✓	✓	✓
REQ-7	1		✓	✓	
REQ-8	4	✓			
REQ-9	2	✓			
REQ-10	4			✓	✓
REQ-11	4			✓	✓
REQ-12	2				✓
REQ-13	4		✓	✓	✓

Max Priority Weight (of Use Case)	3	4	5	5

Use cases 3 and 4 are most definitely what makes our software unique compared to similar software available today, hence why they are given the highest weights.The specific detail of the use case 3 so essential to our idea is that it involves ranking the <u>found</u> books into tiers, and then comparing all the tiers to all the books who were entered. This specific tool is what our software does, that no other current software considers, hence why it is one of the crucial and innovative pieces of our project.

Use case 4 is also one of the most important pieces of the software because it carries out the data from use case 3 to the user, and essentially displays to the user what our algorithms and findings would most recommend. Although use case 4 is typical with recommendation programs, it will function very specific and closely to how user case 3 will 'tell' it operate and display.

Use case 1 is the same type of implementation that can be found in practically all web searches, hence why it is important but not as specific to our design; and case 2 does what most recommendation software does when a user inputs <u>single</u> book to find recommendations for. Use cases 3 and 4 is what really sets our idea apart from what's currently available.

The two most important use cases, as shown from out traceability matrix, are use cases 3 and 4. In a nutshell, use case three takes all the data from use cases 1 and 2, then essentially creates the basis for which what our program will conclusively decides what will be recommended to the user. Use case 4 plays the role of communicating the results to the user and also giving a brief plot description of each of the options while also indicating which of the tiers each book falls under.

2.4.3 Fully Dressed Description & Sequence Diagram

Use Case 3: Search Prediction

Primary Actor: User

Goal: To narrow down the search results to the same type of book

Participating Actors: A computer/mobile device

Preconditions: Software has all popular search data and the necessary algorithm

Postconditions: The user has received a list of books with similar titles

Flow of Events:

1. User→ Enters Harry Potter into the search bar.
2. System← begins to find Harry Potter in the database for predictive text
3. User→ Selects Harry Potter from the predictive text
4. System← Creates table that stores all the books that the user input
5. System← All types of Harry Potter books regardless of the specific title are displayed on the user interface
6. System← Use Analysis Algorithm to rank the books form S→ C
7. User→ Use search function to find book B and C and see which book is more closely related to book A

Use Case 4: Evaluate Recommendations

Primary Actor: User

Goal: To find the desired book from the recommendations

Participating Actors: A computer/mobile device

Preconditions: A list of books with the similar titles are retrieved from the database and ready to be used

Postconditions: The user has found the desired book from the predicted searches

Flow of Events:

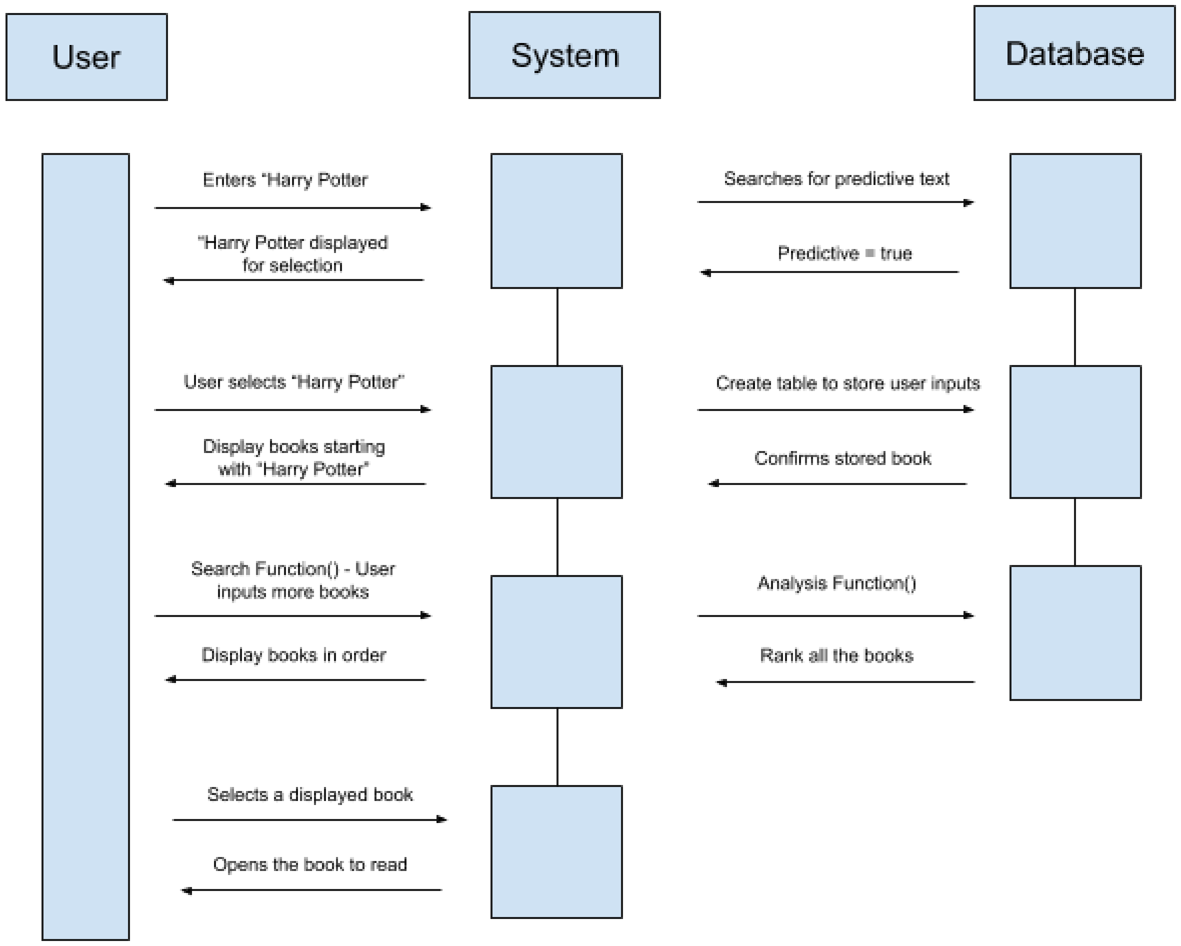
1. User→Wants *Harry Potter and the Sorcerer's Stone*
2. System← Performs algorithm and displays *Harry Potter and the Chamber of Secrets, Harry Potter and the Sorcerer's Stone, Harry Potter and the*

Goblet of Fire. The system takes the key word Harry Potter and finds any book that starts with those keywords. Everything that appears after those keywords are irrelevant

3. User→ Selects *Harry Potter and the Sorcerer's Stone* since it is listed.
4. System← Opens the book for the user to read.

2.4.4 System Sequence Diagram

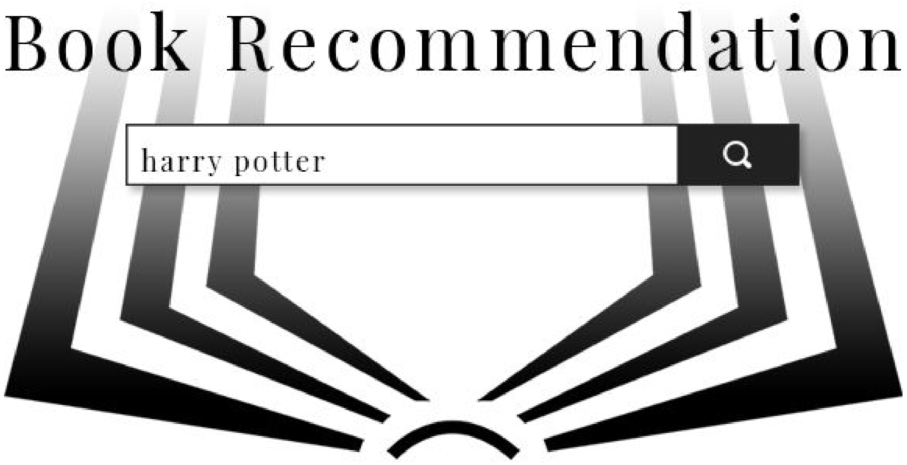
This is the system sequence diagram for User Cases 3 and 4:


User
System
Database
Enters "Harry Potter
"Harry Potter displayed for selection
Searches for predictive text
Predictive = true
User selects "Harry Potter"
Display books starting with "Harry Potter"
Create table to store user inputs
Confirms stored book
Search Function() - User inputs more books
Display books in order
Analysis Function()
Rank all the books
Selects a displayed book
Opens the book to read

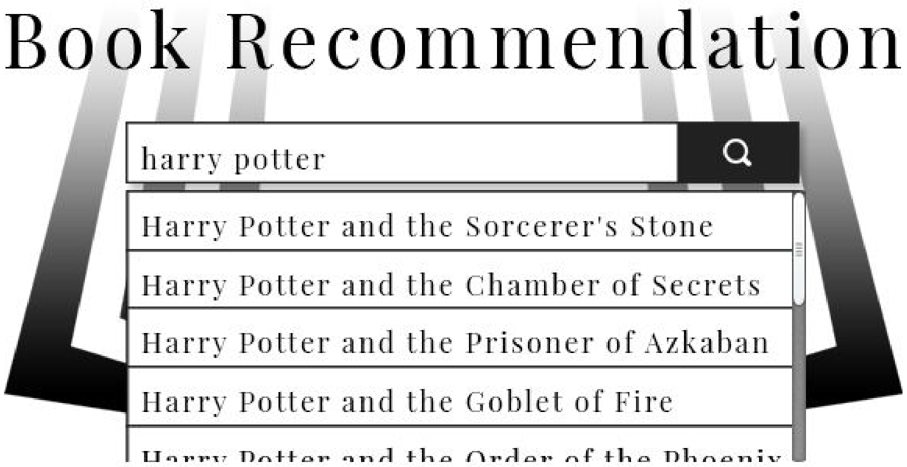
3. User Interface Specification

3.1 Preliminary Design

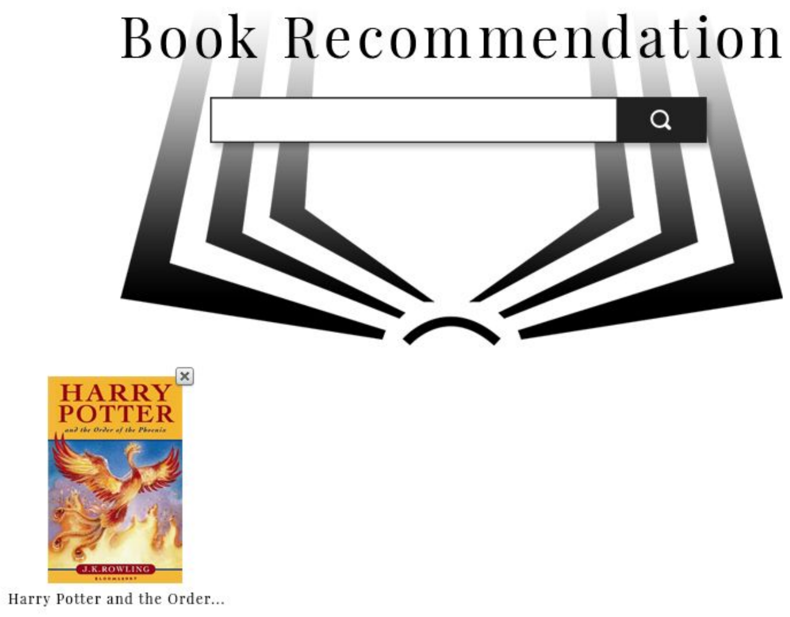
First, the user enters the name of a book into the search bar.



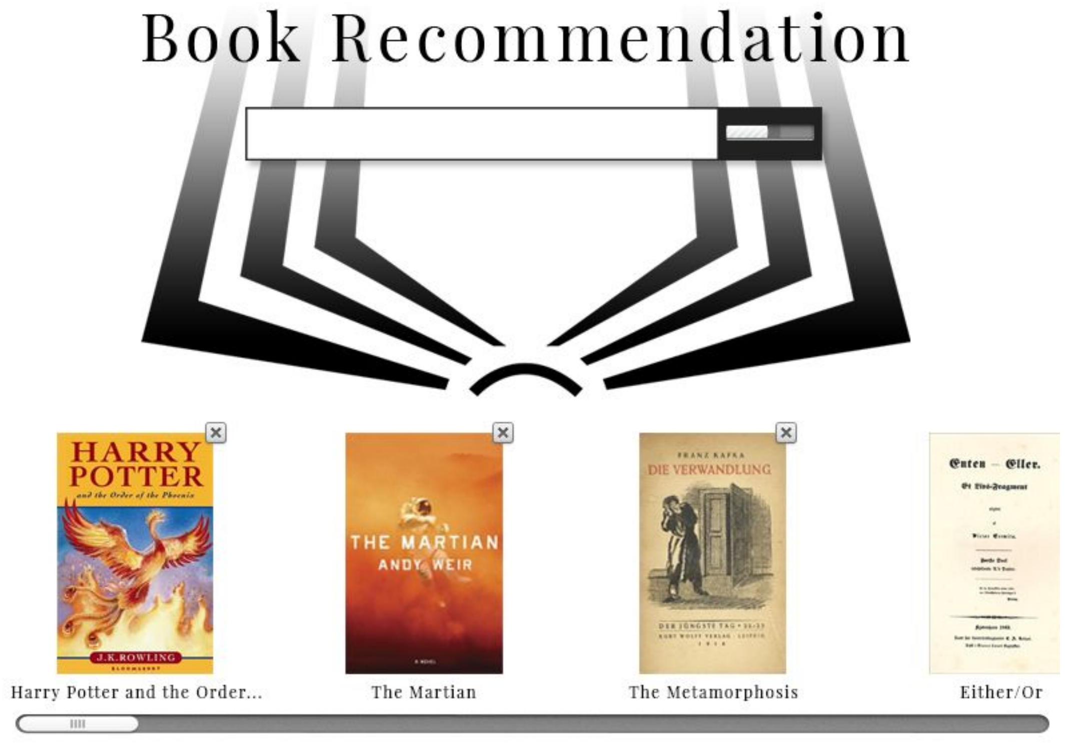
The query completion dropdown displays a list of available books.


Book Recommendation
harry potter
Harry Potter and the Sorcerer's Stone
Harry Potter and the Chamber of Secrets
Harry Potter and the Prisoner of Azkaban
Harry Potter and the Goblet of Fire

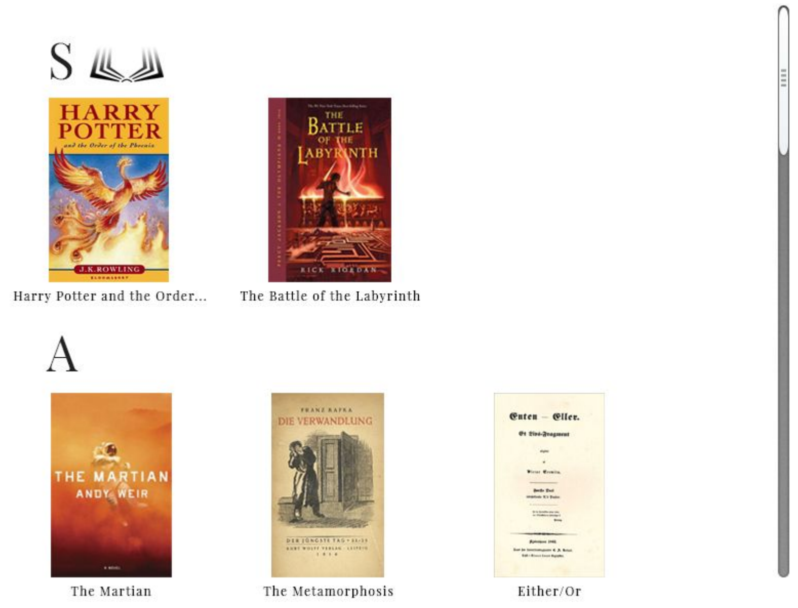
The user then selects a book from the dropdown which is then displayed on the page.



After selecting all the books they wish to use, the user clicks the search button for results.



Based on the user's entries, a page of recommendations is displayed, ranked from S to C.



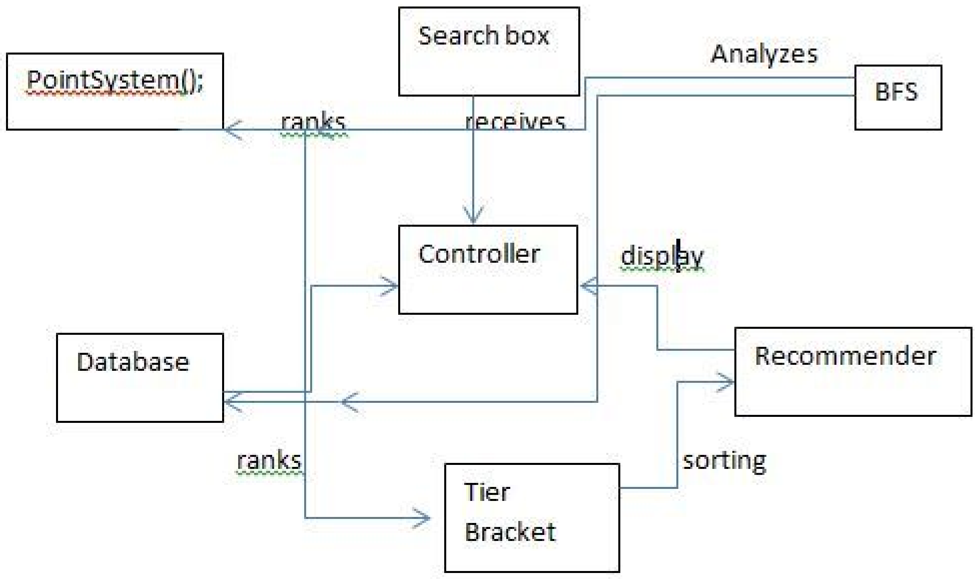
3.2 User Effort Estimation

The scenario with the most usage is likely someone looking for books that explore the same themes and share the same qualities (author, genre, age, etc) as books they have enjoyed reading previously. In order for this scenario to be present, the user would have to enter the name of each book. Because we don't want them to enter the name of a book which we do not have in our database, the application will have a query completion dropdown menu from which they will have to select the book they wish to enter. This will decrease the number of keystrokes required to enter each book's title, and although it increases the number of mouse clicks (1 per book) it will decrease the number of inputs required overall.

After inputting all the books, one mouse click is required to execute the recommendation algorithm on the selected books. Therefore, between 5-10 keystrokes (estimate) and a mouse click may be required per book, and another mouse click to generate results. With 5 books being used to search for results, a total of 25-50 keystrokes and 6 mouse clicks are required. In the results page, after the search is completed, 1 mouse click is required to navigate to each book for information that the user wishes to view (not on our website). Of all of these inputs, none are required for user-interface navigation as the website has a search page and results page; rather all of the inputs are clerical data entry that is required for the algorithm to function.

4. Domain Model Analysis

4.1 Domain Model



4.2 Domain Model Derivation

In simplest terms, the domain model ‘s purpose is to take all the characteristics and operations of the requirements and use cases, then proceed to describe and illustrate how they interact with each other to make the program operate. The domain model allows for an in depth look at these interactions and gives a clearer look as to what is happening in the “backend” portions of the program.

For our particular program, the main components of the program exist in the user interface, database, the manipulation of the data in the database, and then finally the end result outputted to the user. In essence, besides the input partition of the program, every piece of this project interacts with each other and depends on each output from the previous step.

The user interface allows the user to input the books they’d like to search, and our particular search box will then interact with our current database of existing books in order to try and predict to the user the entire title of the book.

The database in itself will be the host of where all the inputted books will reside (provided to the database from the user interface), where all the current books in the database resides, and where the inputted books along with the books for recommendations. So the database interacts with the user interface (search box), and also essentially plays a role with the data that will conclusively be communicated to the user.

The manipulation of the data in the database will use the information gathered in the database, and begin to breakdown the significance of certain books in order to build possible recommendations to the user. So it will interact with the database's given outputs, and proceed to filter out what will/will not be communicated to the user.

The output will interact with the most updated version of the database (what's left after all the manipulation), and output it chronologically to the user. So this will have heavy interactions will practically all aspects of the program except for the initial search box.

4.3.1 Concept Definitions

Controller: Coordinate actions of concepts associated with this use case and delegate the work to other concepts

Searchbox: takes the users input as strings and displays a dropdown of search predictions for the user to potentially click on

Database: prepare database query that best matches the user's search criteria and retrieve the records and retrieve the records from the database

Recommender: goes through tags with the highest multiplier

BreadthFirstSearch(); looks for tags with the highest point value and traverses to other tags with the lowest points

PointSystem(); gives tags points depending on how well they match the user's reading preferences and search patterns

TierBracket(); ranks books according to their points and puts them into their appropriate tiers for easy and fast recommendations

4.3.2 Association Definitions

Controller & Database: Controller passes the search requests to the Database

Controller & PointSystem(); Controller retrieves the data of points ready to be used.

BFS & TierBracket(); BFS looks for books with specific point values for the TierBracket() to sort them

TierBracket() & Recommender: Recommender suggests books to the user after looking up books that are associated with tags of the highest points

4.3.3 Attribute definitions

Searchbox

Attributes: Text Input, Predetermine Title/Author

- Text Input: takes in inputs from the user in the form of text from a hardware keyboard (and potentially other forms of input depending on what we decide will be our select operating systems of compatibility)

- Predetermine Title/Author: Takes whatever text is in the search box and tries to predetermine what the possible title of the book might be, along with the books author. It will update as more and more characters are inputted into the search box

Database

Attributes: book database, input storage

-Book Database: This is where all of the books in the entire database will be stored (should already include the books that the user inputted in)

- Input Storage: This is separate database, where all of the inputs will be stored, and will be update and convert into a database containing the inputs and the books similar to the inputs

Recommender

Attributes: tag multiplier

-Tag Multiplier: will go through each of the similar books to the inputs (from the input storage) and quantify the tags of each of those books, with a multiplier, in order to associate each of the books with an overall number

PointSystem

Attributes: assigns points

-Assigns Points: takes each book from the input storage, and takes the information from the recommender and assigns a point quantity to each book based off of the accumulated tag multipiers

TierBracket

Attributes: Hierarchy

-Hierarchy: will takes the points for each book and rank each book from most points to least amount of points. It will then create cutoffs for each book, based off of the amount of points given to them, and break off the books into their respective tier

4.3.4 Traceability matrix

Traceability Matrix of the Domain Concepts Along With The Use Cases

Domain Concepts	UC-1	UC-2	UC-3	UC-4
Controller	✓	✓	✓	✓
Search Box	✓	✓		
Database		✓	✓	
Recommender		✓	✓	✓
BreadthFirstSearch		✓	✓	
Point System		✓	✓	✓
Tier Bracket		✓	✓	✓
	Priority Weight Of UC-1: 3/5	Priority Weight Of UC-2: 4/5	Priority Weight Of UC-3: 5/5	Priority Weight Of UC-4: 5/5

4.5 System Operation Contracts

UC-3 Search Prediction

Preconditions:

- User had entered the books that they have read.
- Database is updated with the latest books
- Search algorithm is in working condition

Postconditions:

- User’s books’ suggestion is cleared from the search box for next input
- User has received a list of books that are recommended

UC-4 Evaluate Recommendations

Preconditions:

- The list of books recommended from UC-3 Search Prediction has been retrieved

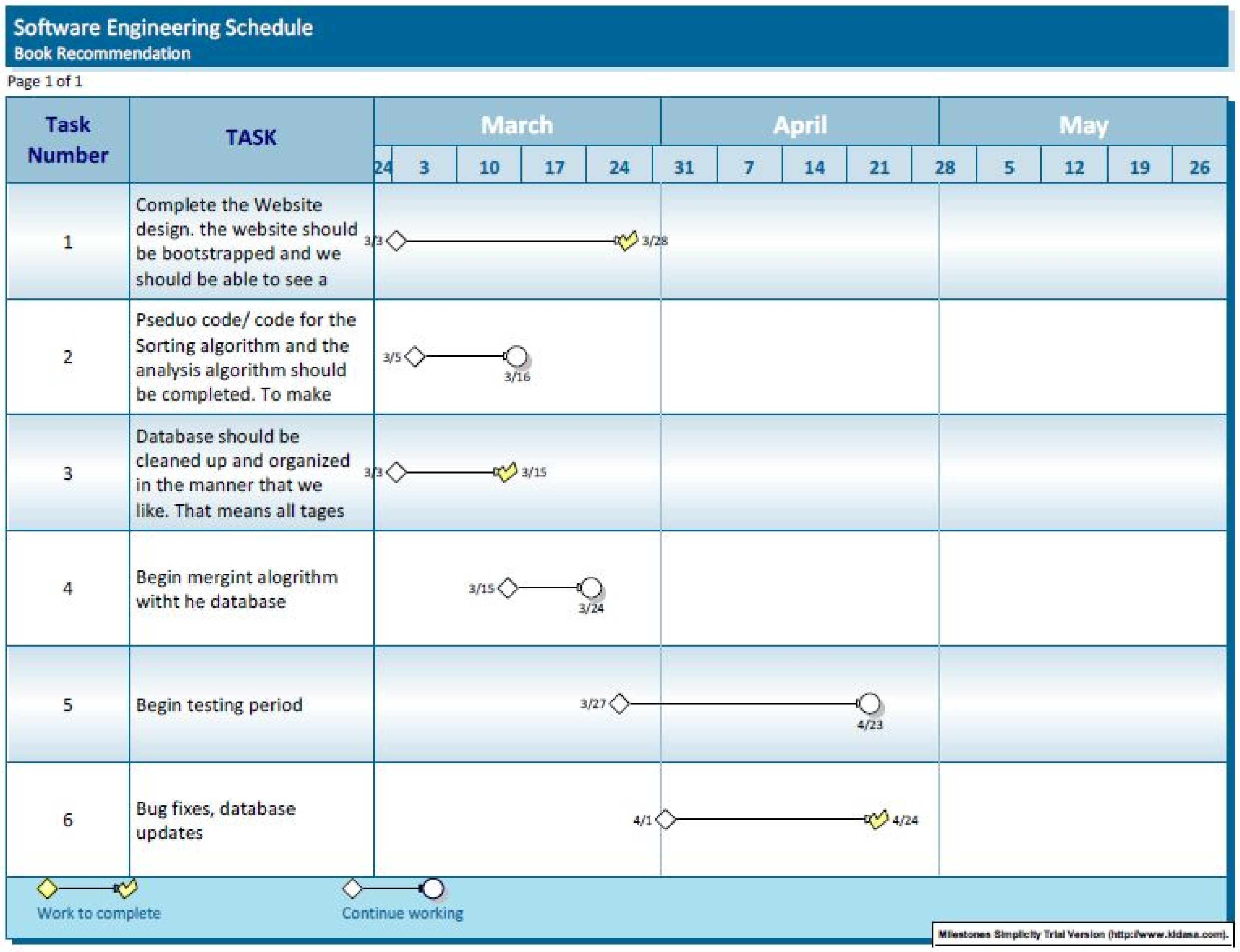
Postconditions:

- Books displayed can be sourced to read or buy online
- Books are sorted into the correct brackets based on the mathematical model

4.6 Mathematical Models

Our mathematical modeling is when we use percentile brackets to break the recommendation into tiers. The way we do this is basically use the Bell Curve model. The reason we use this is because we want to make sure that there are only a few books at the S tier ranking. If we have only a selected few in the S rank, that makes the legitimacy of that tier better. That will be about the top 5% of books. The way that we are getting the point break up is by saving the scores of each book in out list. We take the one book that has the most points, set that as our max and then break that into each group. So for example let's say that the top recommendation had a total point of 2500 points. S rank would be 2975-2500, A rank 2250-2974, B rank 2000-2249, C rank 1750-1999. That is how we are decided which books get placed in which tier

4.7 Plan of Work



Task Number	TASK	
1	Complete the Website design. the website should be bootstrapped and we should be able to see a	3/3 – 3/28
2	Pseduo code/ code for the Sorting algorithm and the analysis algorithm should be completed. To make	3/5 – 3/16
3	Database should be cleaned up and organized in the manner that we like. That means all tages	3/3 – 3/15
4	Begin mergint alogrithm witht he database	3/15 – 3/24
5	Begin testing period	3/27 – 4/23
6	Bug fixes, database updates	4/1 – 4/24

4.7.1 Product Ownership Description

Functional Features:

- Ability to input an infinite number of book entries
- Using BFS to optimize run time
  - Start with looking at the tag that has the highest point value and then from there start looking deeper into the lower point tags to get a more accurate match.
- Use point system to rank recommended outcomes
  - The point system will help give a more accurate recommendation and be a deciding factor between two books that were close but one say had a better fit for the user
- Take the highest score and use that to create percentage brackets to categorize recommendations in S, A, B, C tiers

Responsibilities

Name	**Did**	**Currently**	**Will Do**
Shazidul (Analysis Algo)	Learned SQL and Python for compatibility	Working on Analysis Algo and sorting methods	Merge algo with database and site
Vedanta (Sorting Algo)	Created Github repository	Begin working on the algorithm	Finish algorithm and merge with database
Avani (Presentation and UI design)	Documentation and Group Meeting Coordinator	Documentation and Group Meeting Coordinator	Documentation and Group Meeting Coordinator
Anthony (Predictive Text)	Contributed concepts for the functionality	Working with front end for site development	Will merg algo with site (he is the bridge between both)
Alan (Database Entries)	Database creator	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
Seymour	Research for point/tier system.	Working on sorting and selecting algo	Help merg the algo with the analysis algo and the site
Kutay (Presentation and UI design)	Documentation	Documentation	Documentation and bug tester

a. **Functionality:**
    - Tag Comparisons with all the other books that are in the database already.
    - The storage of the results that were pulled.
    - Categorizing them into tiers via percentile.
    - By doing multiple tries using the algorithm, we can detect how the results are compared to doing the search without the algorithm.
b. **Qualitative Property:**
    - We are going to be creating a website that the customer will be using, displaying a use of intuitive UI (User Interface). As the customer searches his or her book, the predictive text element of our interface will allow the customer to select the book much quicker.
    - We are going to be using a Raspberry Pi to have a server running at all time
    - When listing the books, the tier will give a quick description about what is similar to the books that user input and why it would be a good read.

4.7.2 Project Size Estimation based on Use Case Points.

UC-1:
External Inputs: User entering books, as well as selecting books to use for algorithm.
External Output: None.
External Inquiries: Dropdown with book names (on file) for user to select as inputs.
Internal Logical Files: List of books available to use for algorithm.
External Interface Files: None.

UC-2:
External Inputs: User entering books, as well as selecting books to use for algorithm.
External Output: Results of algorithm with books ranked based on similarity using tags.
External Inquiries: Dropdown with book names (on file) for user to select as inputs.
Internal Logical Files: List of books available to use for algorithm.
External Interface Files: None.

UC-3:
External Inputs: User entering books, as well as selecting books to use for algorithm.
External Output: Results of algorithm with books ranked based on similarity using tags.
External Inquiries: Dropdown with book names (on file) for user to select as inputs.
Internal Logical Files: List of books available to use for algorithm.
External Interface Files: None.

UC-4:
External Inputs: None.
External Output: Results of algorithm with books ranked based on similarity.
External Inquiries: None.
Internal Logical Files: None.
External Interface Files: None.

Overview:
Although each use case (except the last one) requires inputs and results in an output, every use case overlaps every other use case, reducing the size of the overall project substantially. The final requirements will be: the algorithm (points allocation), analysis (statistical representation of results), user interface, and database (local) all of which can be applied to each of the use cases. Once we are able to receive inputs and display the accurate results, each use case should be available for testing, and the focus becomes optimization and increasing accuracy, as well as adding extra features given excess time.

5. References

https://blog.statsbot.co/recommendation-system-algorithms-ba67f39ac9a3

https://www.toptal.com/algorithms/predicting-likes-inside-a-simple-recommendation-engine

https://www.ibm.com/developerworks/library/os-recommender1/index.html