

Features









1. Search User (Autocomplete)
2. Search User (Exact Match)
3. Find mutual friends of a user
4. Minimum friends(steps) to make a user friend
5. Friend Suggestion
6. Find Twin Friend of a user
7. Top Hashtags
8. Search based on filters
9. Add Hashtag
10. Sort users
11. Sort friends of a particular user
12. Sort friends of all users
13. Write current data to file

Structure of User



```
public class User {  
    private int id;  
    private String firstName;  
    private String lastName;  
    private List<Integer> friends;  
    private Set<String> hobbies;  
    private Set<String> hashtags;  
    private String location;  
}
```

Create User by Random Data

| | | | |
|--|------------------|---------------|--------|
|  cities | 22-07-2023 13:05 | Text Document | 24 KB |
|  first_names | 20-07-2023 17:16 | Text Document | 32 KB |
|  hash_tags | 25-07-2023 19:54 | Text Document | 5 KB |
|  hobbies | 22-07-2023 11:44 | Text Document | 2 KB |
|  last_names | 20-07-2023 17:17 | Text Document | 35 KB |
|  user_data | 25-07-2023 19:55 | Text Document | 375 KB |

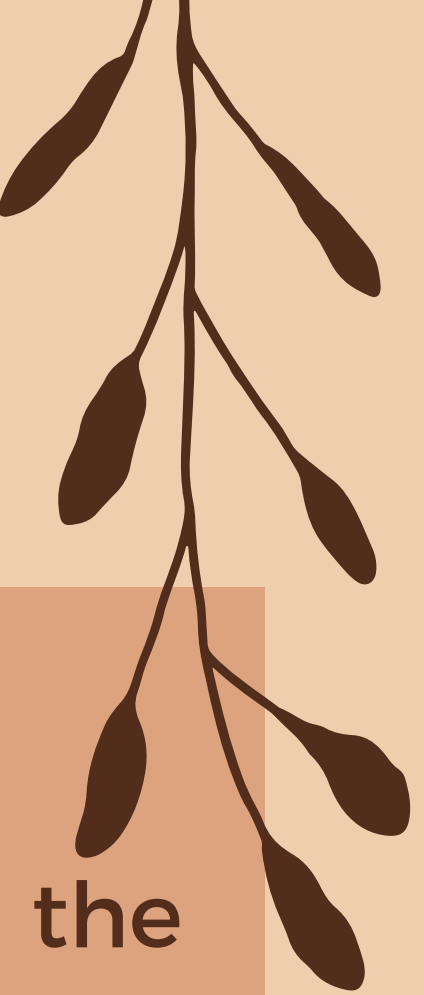
We can customize following properties



```
int maxFriendsLimit = 50;  
int maxHobbiesLimit = 10;  
int maxHashtagsLimit = 10;  
int minCommonHobbyToSuggestFriend = 3;
```

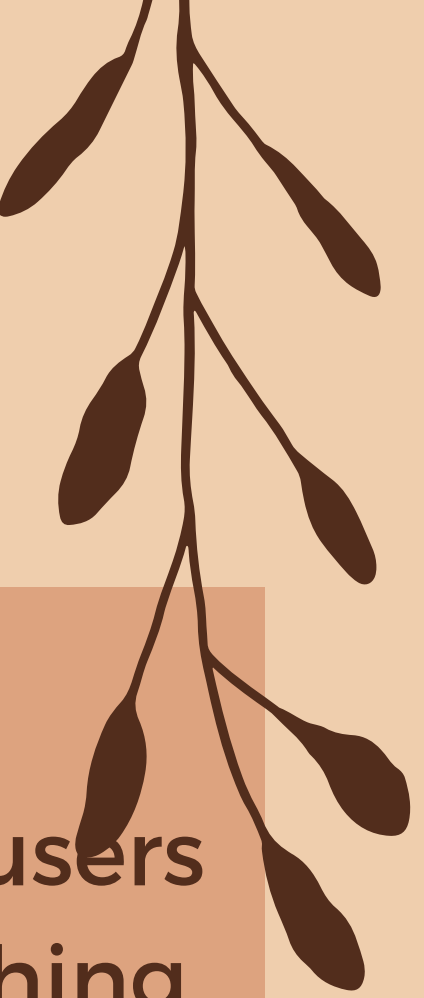
1. Search User (Autocomplete)

Users can conveniently find other users by typing partial names, and the system will automatically suggest relevant matches as they type. This feature is **implemented using a Trie data structure**, which efficiently stores user names and enables quick auto-completion suggestions. As users type characters, the algorithm traverses the Trie to find matching prefixes and provides autocomplete suggestions in real-time.



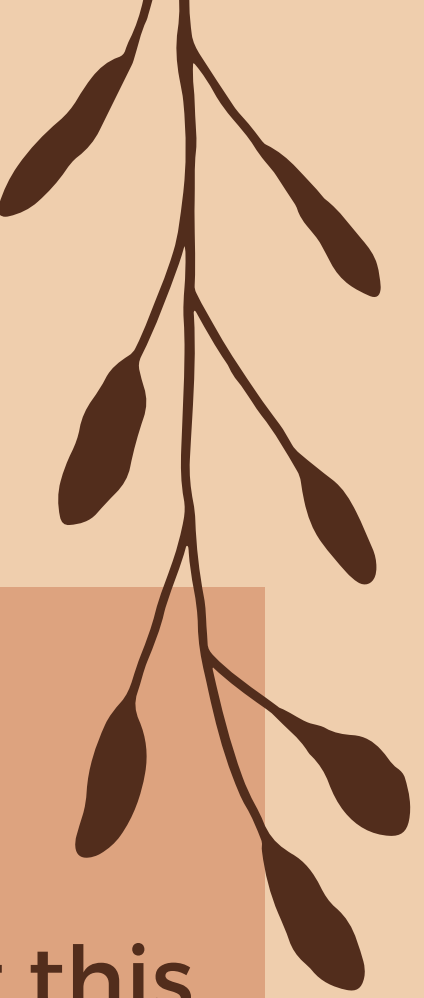
2. Search User (Exact Match)

The "Search User (Exact Match)" feature allows users to find specific users based on their exact names. To achieve efficient and fast exact matching, the feature is **implemented using a Trie data structure**. When a user enters the full name of the person they want to find, the system traverses the Trie to locate all users whose names match the input exactly. With the Trie data structure, the search operation is optimized, making it easier for users to find specific individuals in the social media network.



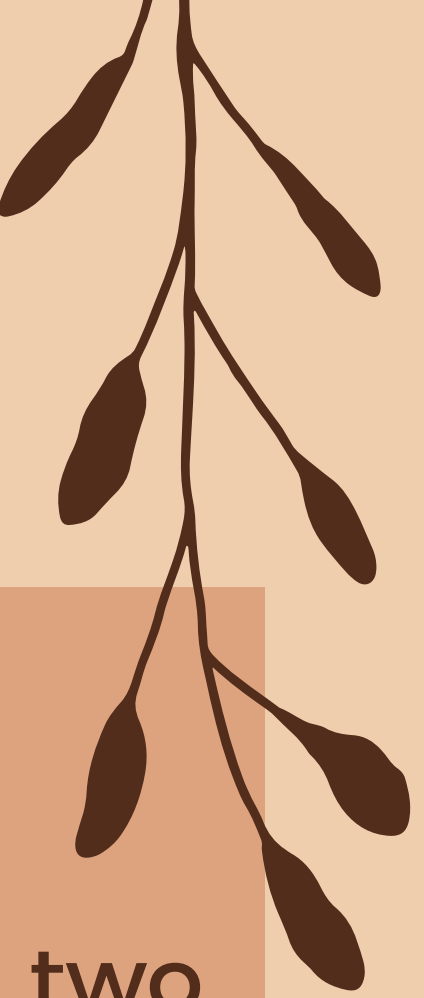
3. Find Mutual Friends of a User

Users can discover mutual friends between two users. To implement this feature, we **represent the user network using a graph** with an adjacency list. By checking the friends of two users and finding common users (via userid) between them, the system identifies mutual friends and displays the results to the user.



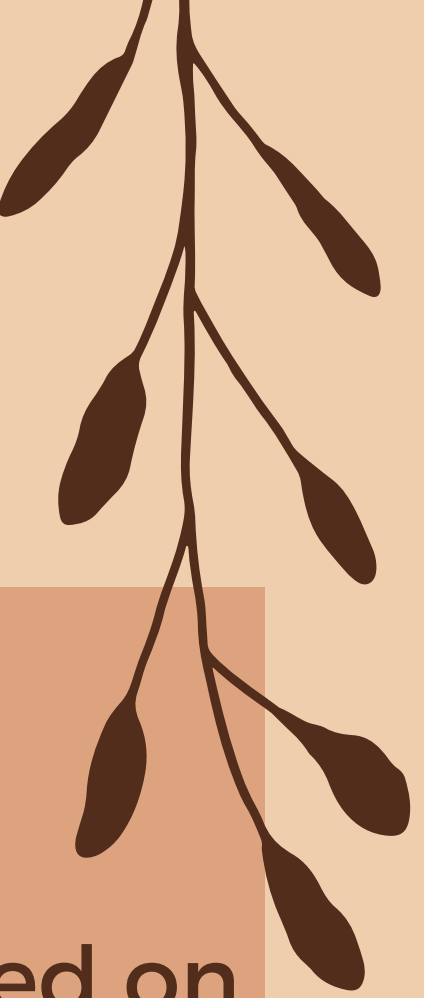
4. Minimum Friends (Steps) to Make a User Friend

This feature determines the minimum connection distance between two users. The system employs **Breadth-First Search (BFS) on the graph representation of the user network**. By exploring the shortest path between the two users, the algorithm calculates the minimum friends needed to make them friends.



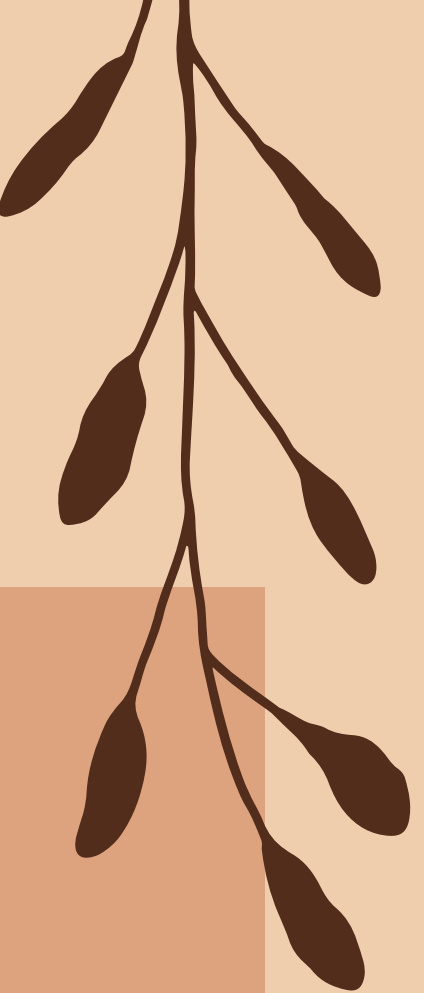
5. Friend Suggestion

Friend suggestion aims to recommend potential friends to a user based on common hobbies with their friends' friends. Using **graph representation and set operations**, the system identifies the hobbies shared by the user's friends and their friends. Users with a significant number of common hobbies are suggested as potential friends.



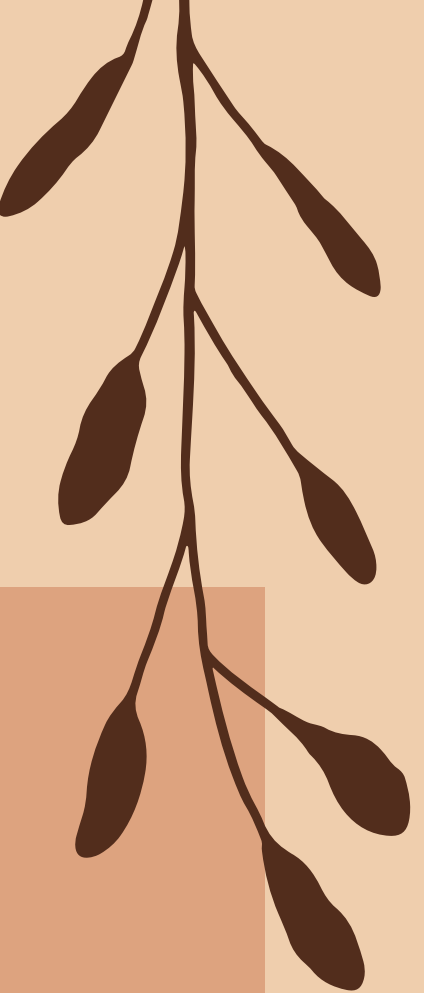
6. Find Twin Friend of a User

This feature helps users find their "twin" friend, i.e., the friend who shares the most common hobbies with them. The system calculates the number of common hobbies between the user and each of their friends. The friend with the highest count of shared hobbies is identified as the twin friend.



7. Top Hashtags

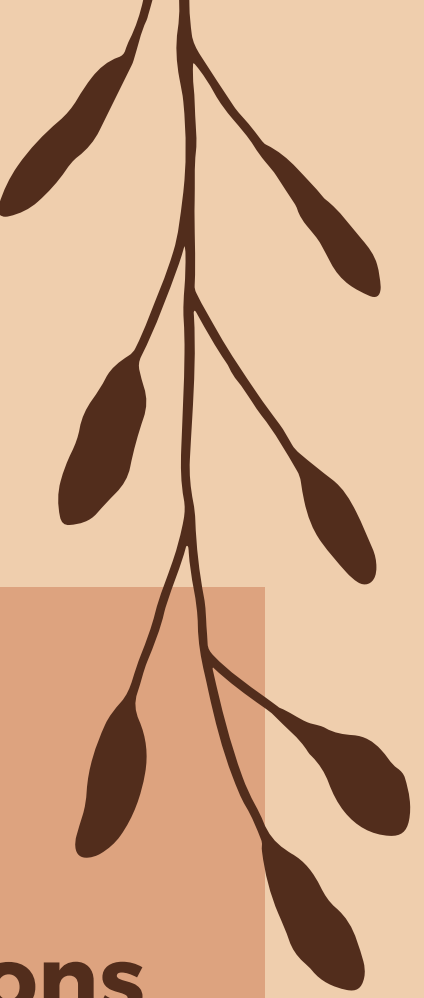
The system showcases the most popular hashtags on the platform. This is achieved using a **Priority Queue (Max-Heap)** with a custom comparator. Hashtags are ranked based on their usage frequency, and the top N hashtags are displayed to the users.



8. Search Based on Filters

Users can filter search results based on **hashtags, hobbies, or locations**.

This **functionality is implemented using HashMaps**, which allow for efficient retrieval of filtered data. Users can input a filter type and the specific filter value, and the system will display relevant search results.

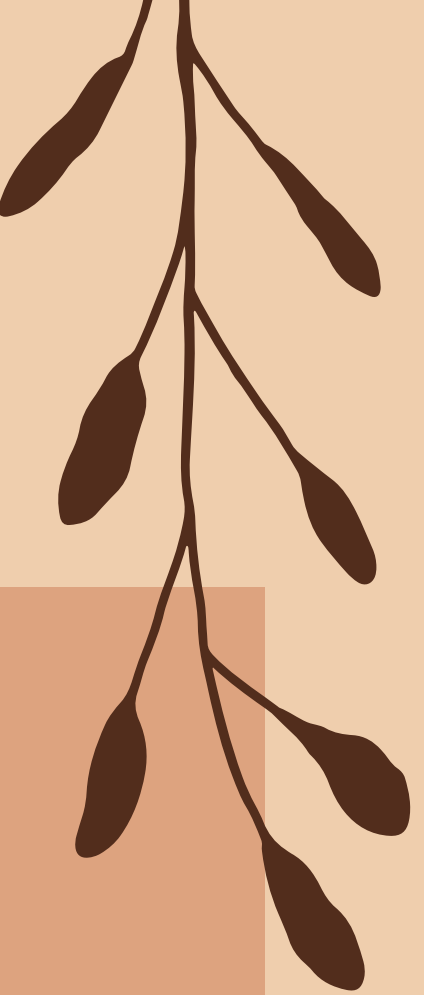


9. Add Hashtag

The "Add Hashtag" feature allows users to add hashtags to their posts and profile. Before adding a hashtag, the system checks if it is in the correct format using **regular expressions (regex)**. The hashtag must start with "#" and consist of only alphanumeric characters, without any spaces or special characters. Valid hashtags are then added to the user's profile, and their usage count is tracked for popularity analysis. certain hashtags may be banned. To prevent the use of **banned hashtags**, the system employs the **Boyer-Moore string search algorithm**. The algorithm checks if the hashtag contains any banned words and prevents their addition if found.

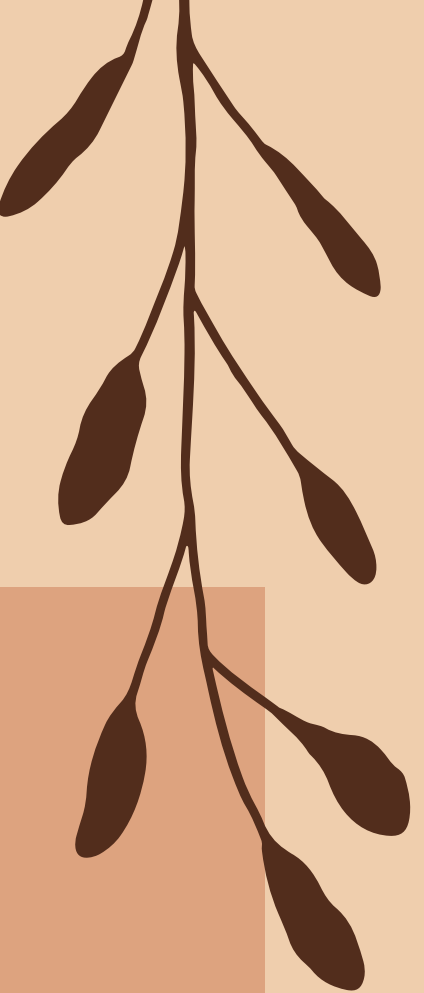
10. Sort Users

Users can sort the user list based on their names in ascending order. The system utilizes the `Collections.sort()` method with a custom comparator to arrange the user list alphabetically by their full names.



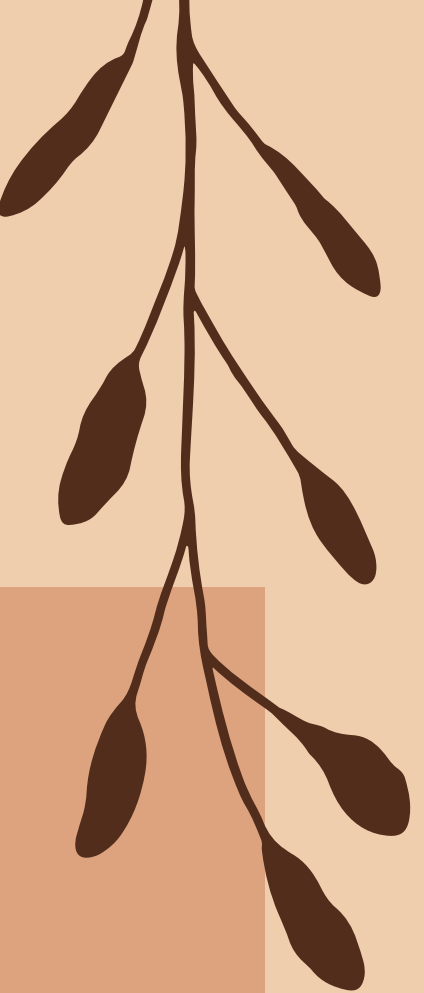
11. Sort Friends of a Particular User

This feature allows users to sort their friend list based on their friends' names in ascending order. The system uses the `Collections.sort()` method with a custom comparator to arrange the list alphabetically by their friends' full names.



12. Sort Friends of All Users

Users can sort the friend lists of all users based on their names in ascending order. The system iterates through each user and applies the sorting process mentioned in the previous feature to organize the friend lists alphabetically.



13. Write Current Data to File

Users have the option to save the current user data to a file for future reference. The system employs file handling with **BufferedWriter** to write user information to the specified file. The data is formatted and saved in the file for later use.

