A blue and white logo

Description automatically generated

# **SOCIAL MEDIA FEED GENERATIION ALGORITHM**

**Submitted To: Sir Tanveer**

**Submitted By:**

**Abdul Muizz (FA23-BCS-117)**

**Muhammad Abdullah (FA23-BCS-108)**

**Muhammad Siraj (FA23-BCS-077)**

**Muhammad Abuzer Zia (FA23-BCS-055)**

**Dated, December 26, 2024**

**CONTENTS:**

[**SOCIAL MEDIA FEED GENERATIION ALGORITHM** 1](#_Toc186060556)

[**ABSTRACT:** 3](#_Toc186060557)

[**INTRODUCTION:** 4](#_Toc186060558)

[**FEATURES:** 5](#_Toc186060559)

[**FUNCTIONALITY:** 7](#_Toc186060560)

[**DATA STRUCTURES USED:** 9](#_Toc186060561)

[**CODE EXPLAINATION:** 12](#_Toc186060562)

[**HOW THE COMPONENTS WORK TOGETHER:** 16](#_Toc186060563)

[**TECH STACK:** 17](#_Toc186060564)

[**UML CLASS DIAGRAM:** 18](#_Toc186060565)

[**SCOPE:** 19](#_Toc186060566)

[**LIMITATIONS:** 20](#_Toc186060567)

[**CONCLUSION:** 21](#_Toc186060568)

## **ABSTRACT:**

The Social Media Feed Generation System is an innovative software solution designed to revolutionize content delivery in social networking platforms. Developed in C++, it employs sophisticated algorithms to curate and deliver personalized content streams to users. The system leverages key data structures including doubly linked list, priority queues for content ranking, graphs for social network analysis, and hash tables for rapid user preference lookups. By integrating user behaviour tracking, content relevance scoring. the system efficiently manages content distribution while optimizing engagement metrics. The architecture supports dynamic updates to user preferences, real-time content prioritization, and scalable performance analysis, providing valuable insights into user engagement patterns and content effectiveness. This system serves as both a practical implementation for social media platforms and an educational tool for understanding complex content distribution algorithms.

## **INTRODUCTION:**

In today's digital landscape, social media users face an overwhelming volume of content that makes finding relevant information increasingly challenging. The Social Media Feed Generation System addresses this challenge by implementing an intelligent content curation mechanism that automatically sorts and prioritizes social media posts based on user preferences and behaviour patterns. By combining advanced algorithms with efficient data structures, the system analyses user interactions, content relevance, and social connections to create personalized feeds that enhance user engagement and satisfaction. This project demonstrates how modern software engineering principles can be applied to solve real-world content distribution challenges while maintaining scalability and performance.

## **FEATURES:**

**1. All-Interest Management**

* A centralized **All Interest Class** to store and manage user interests effectively.
* Dynamic updating of interests as users engages with the platform.
* Supports scalable handling of diverse user interest categories.

**2. User Management**

* **User Class** to store and manage user profiles, preferences
* Provides an interface for users to update their profiles and preferences.
* **User Management Class** to store users in hashmap and allow feature of unique username and efficient lookup.

**3. Connection Management**

* **Connection Manager** to establish and maintain links between users and the feed generator.
* Efficiently handles real-time updates to user connections and preferences.
* Ensures smooth data flow between components to generate accurate, personalized feeds.

**4. Feed Generation**

* **Feed Generator** utilizes advanced algorithms to create a priority-based, personalized feed.
* Generates feeds considering user interests, connection activity, and priority of posts.
* Supports algorithm updates to improve feed accuracy over time.

**5. File Handling**

* **File Handler** to manage data storage and retrieval operations efficiently.
* Handles logs, user profiles, post data, and analytics records.
* Ensures data integrity and quick access for other components.

**6. Post Creation and Management**

* **Post Class** enables creation of diverse content types, including text, images, and videos.
* **Post Management Class** organizes posts, tracks metadata, and ensures relevance for feed generation.
* Facilitates post updates, deletions, and tagging of related interests.

**7. User Analytics**

* **User Analytics Class** provides detailed insights into user behavior and engagement patterns.
* Visualizes metrics such as user activity trends.
* Helps improve personalization algorithms and enhances user experience.

**8. Priority-Based Sorting**

* Advanced sorting mechanisms to rank posts based on user interests, post priority, and engagement metrics.
* Ensures the most relevant content is displayed at the top of user feeds.

This feature set outlines the comprehensive functionality of the Social Media Feed Simulator, showcasing its capability to deliver a robust and personalized user experience.

## **FUNCTIONALITY:**

**1. All Interest Class**

* Manages user interests by showing them interests and asking the user to choose from them.

**2. User Class**

* Manages user profiles, including personal details and preferences.
* Give unique ids to users

**3. User Manager Class**

* Handles user registration, deletion, and retrieval with unique usernames and IDs.
* Manages users using unordered\_map for efficient lookup and updates.

**4. Connection Manager**

* Establishes and manages connections between users.
* Dynamically updates feed generation based on new or removed user connections.
* Ensures efficient data flow to support personalized feed creation.

**5. Feed Generator**

* Generates personalized feeds using priority-based algorithms.
* Incorporates user interests, connections, and activity for relevance.
* Updates feeds in real-time to reflect the latest user interactions.

**6. File Handler**

* Manages storage and retrieval of user profiles, interests, posts, and analytics.
* Handles file operations for backups and logs with error management.
* Ensures data persistence and integrity across the system.

**7. Post Class**

* Enables users to create posts with text, images, and videos.
* Tags posts with relevant interests to ensure accurate feed placement.

**8. Post Management Class**

* Tracks and organizes post metadata, including timestamps and engagement metrics.
* Supports updates, deletions, and relevance adjustments for posts.

**9. User Analytics Class**

* Tracks and analyzes user behavior, engagement trends, and post performance.
* Provides actionable insights and graphical visualizations for users and system optimization.

This class-based functionality ensures modular and efficient operations, making the system adaptable and user-focused.

## **DATA STRUCTURES USED:**

The Social Media Feed Simulator employs a variety of efficient data structures to ensure modularity, scalability, and high performance. Below is an analysis of the data structures used and their specific roles in the system:

**1. Doubly Linked List**

* **Usage**:
  + The **User** class also uses a DoublyLinkedList to store and manage user interests and posts, enabling quick insertion, deletion, and access operations.
  + The **ConnectionManager** uses a DoublyLinkedList to maintain user connections, providing bidirectional traversal for efficient management of connections.
* **Advantages**:
  + Allows efficient navigation in both forward and backward directions.
  + Supports dynamic resizing for changing connections, interests, or posts.

**2. Unordered Map**

* **Usage**:
  + The **UserManager** class uses an unordered map to store and retrieve users based on their unique usernames and IDs.
  + This provides an efficient mechanism for mapping user data and enabling fast lookups.
* **Advantages**:
  + O(1) average-time complexity for insertions and lookups.
  + Ensures quick user identification and management.

**3. Custom Priority Queue**

* **Usage**:
  + A custom **PriorityQueue** is implemented for two key purposes:
    1. In the **FeedGenerator**, it is used to rank posts by relevance score and timestamp to ensure the most relevant and recent posts appear at the top of the feed.
    2. In the **UserAnalytics**, it is tailored to identify and rank the top 5 most influential users based on their engagement and activity metrics.
* **Advantages**:
  + Dynamically maintains elements in sorted order based on custom-defined priorities.
  + Optimized for scenarios where ranking is crucial, such as prioritizing posts or users.

**4. PostLinkedList**

* **Usage**:
  + A custom linked list, **PostLinkedList**, is used to manage posts specifically.
  + It includes methods for adding, removing, and retrieving posts, as well as calculating the number of posts.
* **Advantages**:
  + Tailored for the unique requirements of post management within the system.
  + Facilitates seamless integration with other components like the **FeedGenerator**.

**5. Vector**

* **Usage**:
  + The **Interest Class** uses vectors to store all interest and then let user to choose from them.
  + The **UserAnalytics** class uses vectors to store data about active users and connections.
  + Supports efficient storage and retrieval of analytics data.
* **Advantages**:
  + Dynamic resizing and random access capability.
  + Ideal for managing sequential data with variable sizes.

**6. Node (Component of Priority Queue and Linked Lists)**

* **Usage**:
  + Used in the **Priority Queue** and **Post LinkedList** to store data items along with pointers to adjacent nodes.
* **Advantages**:
  + Provides a flexible way to structure and link data elements dynamically.

## **CODE EXPLAINATION:**

The **Social Media Feed Simulator** system comprises several classes, each with distinct responsibilities and data structures to ensure modularity and efficient functionality. Here's a breakdown of the code components based on the UML:

**1. User Class**

* **Attributes**:
  + int userId: A unique identifier for each user.
  + string userName: The unique username of the user.
  + string email: The user's email address.
  + DoublyLinkedList<string> interests: Stores the user's interests.
  + PostLinkedList postList: Manages the posts created by the user.
  + static int userCounter: A static counter to assign unique IDs to new users.
* **Methods**:
  + Constructor initializes user attributes and increments userCounter.
  + hasInterest, addInterest, updateInterest and removeInterest: Manage the user’s interests.
  + addPost, removePost, displayPosts: Handle the user's posts.
  + getPostCount: Returns the number of posts created by the user.
  + displayUserDetails, displayUserInterest: display user details and user interests.

**2. UserManager Class**

* **Attributes**:
  + static unordered\_map<string, User\*> userByUserName: Maps usernames to user objects. (used for unique username)
  + static unordered\_map<int, User\*> userById: Maps user IDs to user objects.
* **Methods**:
  + addUser: Adds new user and adds it to the hashmap.
  + removeUser: Removes a user from the hashmap.
  + getUserByName and getUserById: Retrieve user objects efficiently.
  + userExistsByUserName: Checks if a user exists in the hashmap by their username.
  + listAllUser: Lists all users present in the hashmap.

**3. ConnectionManager Class**

* **Attributes**:
  + DoublyLinkedList<pair<int, int>> connections: Stores connections between users as pairs of user IDs.
* **Methods**:
  + addConnection: Adds a new connection between two users.
  + removeConnection: Removes an existing connection.
  + getConnectionsForUser: Retrieves all connections for a specific user.
  + displayConnections: Displays all connections for the system.

**4. Post Class**

* **Attributes**:
  + string content: The content of the post.
  + string timestamp: The time when the post was created.
  + int interestRelevanceScore: The post's relevance score.
* **Methods**:
  + Constructor initializes post attributes.
  + setContent, setTimestamp: Allow updating the post's content and timestamp.
  + getRelevanceScore: Retrieves the relevance score for ranking purposes.

**5. PostLinkedList Class (Custom Linked List)**

* **Attributes**:
  + Node\* head: Points to the first node in the list.
  + int size: Tracks the number of posts in the list.
* **Methods**:
  + addPost: Adds a post to the list.
  + removePost: Removes a specific post from the list.
  + getPostAtIndex: Retrieves a post at a specific index.
  + getNumberOfPosts: Returns the total number of posts in the list.

**6. FeedGenerator Class**

* **Attributes**:
  + UserManager userManager: Used to retrieve user data.
  + ConnectionManager connectionManager: Accesses user connections.
  + PriorityQueue<Post> priorityQueue: Ranks posts for feed generation based on relevance and timestamp.
* **Methods**:
  + generateFeed: Fetches posts from connected users and ranks them.
  + addPostToFeed: Adds a new post to the feed.
  + calculateRelevanceScore: Computes the relevance score for each post.

**7. PriorityQueue Class (Custom Priority Queue)**

* **Attributes**:
  + Node\* start: Points to the highest-priority element.
  + int size: Tracks the number of elements in the queue.
* **Methods**:
  + enqueue: Adds a new element based on its priority.
  + dequeue: Removes the element with the highest priority.
  + peek: Retrieves the highest-priority element without removing it.
  + isEmpty: Checks if the queue is empty.

**8. UserAnalytics Class**

* **Attributes**:
  + None directly, but relies on ConnectionManager and UserManager.
* **Methods**:
  + getMostActiveUsers: Identifies users with the highest activity (e.g., most posts or interactions).
  + getTopInfluencers: Uses the **custom priority queue** to rank and return the top 5 influential users based on their connections and engagement metrics.
  + getTotalConnectionsCount: Computes the total number of connections in the system.

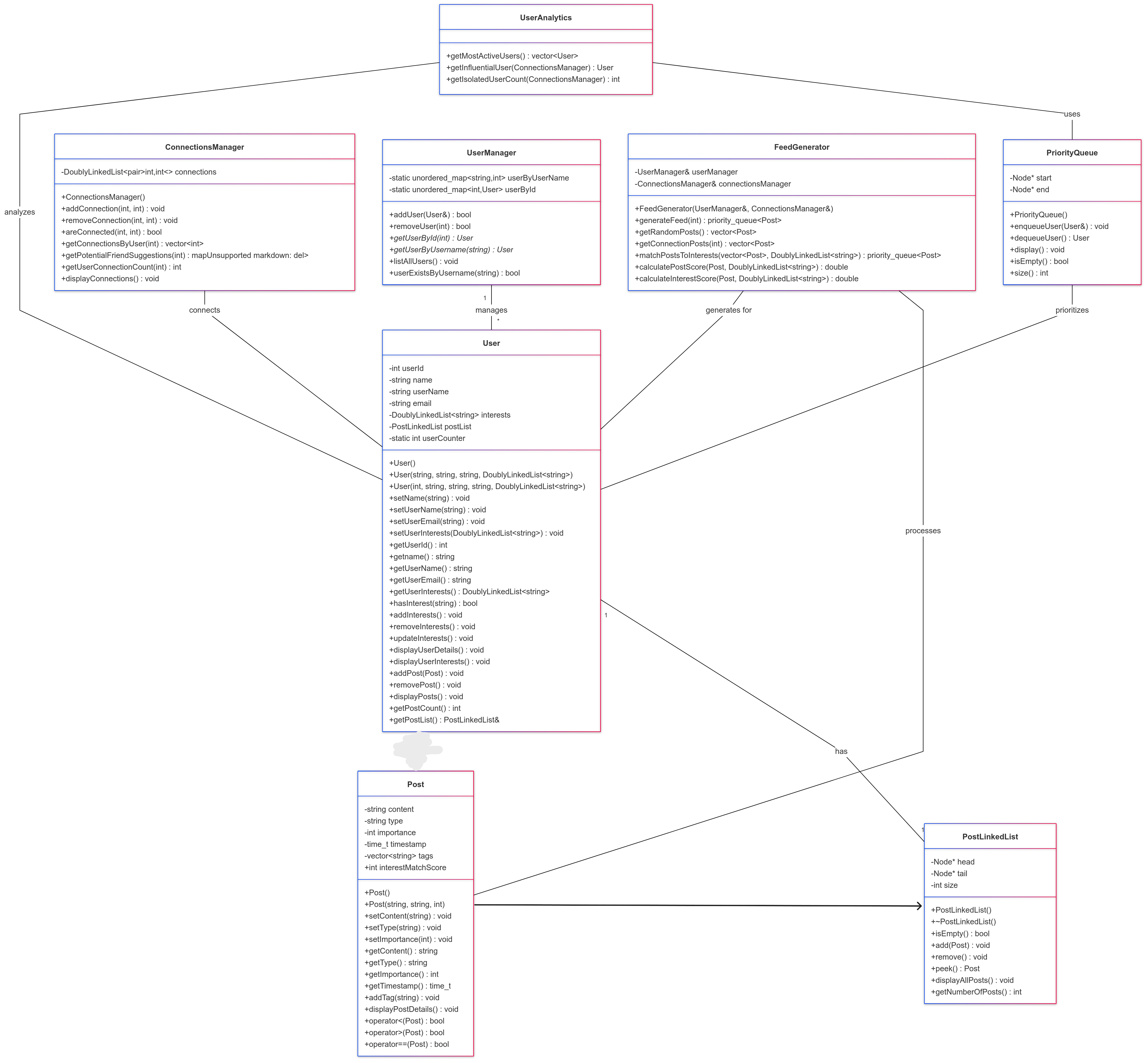
## **HOW THE COMPONENTS WORK TOGETHER:**

1. **User Management**:  
   The UserManager creates and manages users, storing them in an efficient unordered\_map. Each user’s interests and posts are managed through a DoublyLinkedList and PostLinkedList, respectively.
2. **Connection Management**:  
   The ConnectionManager tracks relationships between users using a DoublyLinkedList. Connections influence the feed generation and analytics calculations.
3. **Feed Generation**:  
   The FeedGenerator retrieves posts from connected users and uses the PriorityQueue to rank them based on relevance. It provides personalized feeds that reflect the user’s interests and activity.
4. **Post Management**:  
   Posts are managed through the Post and PostLinkedList classes, ensuring efficient storage, retrieval, and updates for each user’s content.
5. **User Analytics**:  
   The UserAnalytics class processes user activity and connections to generate insights. It leverages the **custom priority queue** to rank the top influencers and active users.
6. **Overall System**:  
   All components integrate seamlessly, using efficient data structures to handle user interactions, post management, and feed generation while ensuring high performance and scalability.

## **TECH STACK:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | **Category** | **Tool/Technology** | **Description** | | --- | --- | --- | | **Programming Language** | C++ | The core language for your project. | | **IDE** | CLion, Visual Studio | IDEs for C++ development with various features and platform support. | | **Build System** | CMake | Cross-platform build system (CMake) and platform-specific systems. | | **Version Control** | Git, GitHub/GitLab/Bitbucket | Version control and hosting services. | |

## **UML CLASS DIAGRAM:**



## **SCOPE:**

This project focuses on developing a modular backend system for a social networking platform, covering user management, connection tracking, post handling, and personalized feed generation. It aims to efficiently manage user interactions, prioritize content, and provide a foundation for further scalability and feature enhancements.

## **LIMITATIONS:**

1. Limited scalability for large datasets due to the use of basic data structures.
2. Absence of real-time updates and distributed system integration.
3. Minimal implementation of security, privacy, and error-handling features.
4. Feed generation relies on basic algorithms, lacking advanced AI-driven personalization.

## **CONCLUSION:**

The project demonstrates a functional and modular approach to managing users, connections, and personalized feeds in a social networking context. While effective within its defined scope, addressing limitations such as scalability, real-time processing, security, and advanced recommendation systems will enhance its utility and applicability to larger, dynamic environments. The project serves as a solid foundation for further development.