Algorithms:

**Leaderboard**

User Ranking:

*showUserLeaderboard(List users[])*
    *LET userList = New PriorityList*
    *FOR user in users*
        *IF user.numPollsVoted > minimumVotes //minimum Votes is a constant*
            *currScore = calculateUserScore(user)*
            *userList.push(user, currScore)*
    *displayUserRanking(userList)*

*//Private helper method added to calculate score. Located in User class.*
*calculateUserScore(User user)*
    *Return user.numCorrect / user.numPollsVoted * 100*

*//Private helper method added to display the ranking of a user. Located in Leaderboard class.*
*displayUserRanking(PriorityList list[])*
    *FOR user in list:*
        *display(Rank {i+1}: User {user.getId}, Score: {user.getScore})*


Referee Ranking:

*showRefLeaderboard(List refs[])*
    *LET refList[] = New PriorityList*
    *FOR ref in refs*
        *currScore = calculateRefScore(ref)*
        *refList.push(ref, currScore)*
    *displayRefRanking(userList)*

*//Private helper method added to calculate score. Located in Referee class.*
*calculateRefScore(Ref ref)*
    *LET accuracy = ref.correctCalls / (ref.getCorrectCalls + ref.getMissedCalls)*
    *LET rating = ref.getRating*
    *LET score = accuracy * rating*
    *Return score*

*//Private helper method added to display the ranking of a user. Located in Leaderboard class.*
*displayRefRanking(PriorityList list[])*
    *FOR ref in list:*
        *Display (Rank {i}: Ref {ref.getName}, Score: {ref.getScore}, YearsExperience {ref.getYearsExpereience})*

**User**

Verification:

*login()*
> *LET String username = user input*
> *LET String password = user input*
> *IF !validateUser(username, password)*
>> *throw error*


*//private helper method added to validate user. Located in User class.*
*validateUser(String username, String password)*
> *Search database for username*
> *IF username exists*
>> *Return validatePassword(password, userID)*
> *ELSE*
>> *Return false*

*//private helper method added to validate password of user. Located in User class.*
*validatePassword(String password, float userID)*
> *LET String password2 = getPassword(userID)*
> *IF password == password2*
>> *Return true*
> *ELSE*
>> *Return false*


**Poll:**
*processVotes():*
> *LET List polls[] = new List*
> *WHILE(gameStreaming)*
>> *IF(callMade):*
>>> *winner, stats = initiatePoll(call)*
>>> *waitFor(winner, stats)*
>>> *call.outcome = winner*
>>> *return stats*

*// Private helper method to perform countdown of poll. Located in Poll class.*
*initiatePoll(Call call)*
> *startCountdown(time)*
> *LET List votes[] = new List*
> *WHILE(countingDown)*

```
            IF (voteSubmitted):
                    votes.push(voteSubmitted.answer)
    LET sideOneCnt = 0
    LET sideTwoCnt = 0
    FOR vote in votes:
            IF vote == sideOne:
                    sideOneCnt++
            ELSE
                    sideTwoCnt++
    LET String winner
    LET stats
    IF sideOneCnt > sideTwoCnt:
            winner = sideOne
            stats = sideOneCnt/(sideOneCnt+SideTwoCnt)
    ELSE
            winner = sideTwo
            stats = sideTwoCnt/(sideOneCnt+SideTwoCnt)
    Return [winner, stats]
```

**User:**

Voting:

```
vote(bool answer)
        waitForPollToComplete()
        IF answer == correctAnswer
                Increment User's score and numVotes
        ELSE
                Increment numVotes
```

**Game Chat:**

```
LET List allChats[]
```

```
// public method added to GameChat class to configure the chat, allow users to input messages,
and check for derogatory information
configureChat()
   LET chatLog[] = New List
   LET usersWatching = getUsersWatching(stream.id)
   allowUserSubmission(usersWatching)
   WHILE(streamOngoing)
```

```
    IF(len(allChats) > 0):
        LET submission = priorityQueue.pop(allChats)
        IF containsDerogatory(submission)
            IF user.numDerStatements >= 3
                banUser(User)
            ELSE
                Display Warning
                user.numDerStatements++
        ELSE
            Display ({user.name}: {submission})


submitMessage(User user, String chat)
    IF canSubmit(user):
        priorityQueue.push(user: chat, priority: determinePriority(user, chat))
    ELSE
        Display Error ("You are sending messages too quickly.")


canSubmit(User user)
    // Throttling logic
    IF currentTime - user.lastMessageTime < MESSAGE_THRESHOLD
        return False
    ELSE
        user.lastMessageTime = currentTime
        return True


determinePriority(User user, String chat)
    // Logic to determine the priority of the message
    // This could be based on user reputation, type of message, etc.
    return priority
```

**Direct Message Chat:**

```
LET int numUsers
LET User[] users
LET String[] chat
LET long chatId


submitChat(User user, String message)
        int left = 0
        int right = length of users - 1

        WHILE left <= right:
```

*int mid = left + (right - left) / 2*

*IF user name equals users[mid] name:*
    *FOR int j from 0 to the length of users[mid] messages:*
        *IF chatId equals messages[j] chatId:*
            *messages[j].push(message)*
            *return chatId*
        *ENDIF*
    *END*

    *// If chatId is not found, add new message at the end*
    *int newMessageIndex = length of users[mid] messages*
    *messages[newMessageIndex].chatId = chatId*
    *messages[newMessageIndex].push(message)*
    *return chatId*
*ENDIF*
*// Adjust search range for binary search*
*IF user name is less than users[mid] name:*
    *right = mid - 1*
*ELSE:*
    *left = mid + 1*
*ENDIF*
*END*

*return CONTACT_NOT_FOUND*

*deleteChat(chatId chat)*
    *FOR int i from 0 to the length of users:*
        *IF user name equals user[i] name:*
            *FOR int j from 0 to the length of user[i] messages*
                *IF chatId equals messages[j] chatId*
                    *delete chat*
                *ENDIF*
            *END*
        *ENDIF*
    *END*

## Profile

*LET String name*
*LET String email*
*LET int age*

```
LET Profile[] friends

changePrivacy()
        IF name privacy set to private
                user.namePublic = false
        ENDIF
        IF email privacy set to private
                user.emailPublic = false
        ENDIF
        IF age privacy set to public && age >= 18
                user.agePublic = true
        ENDIF
        ELSE
                user.agePublic = false
        ENDELSE
        IF friends privacy set to private
                user.friendsPublic = false
        ENDIF

addFriend(Profile friend)
        FOR int i from 0 to the size of friends:
                IF friend.name less than friends[i] name:
                        IF size of friends > length of friends
                                length of friends *= 2
                        ENDIF
                        FOR int j from size of friends+1 to i
                                friends[j] = friends[j-1]
                        END
                        friends[i] = friend
                ENDIF
        END

removeFriend(Profile oldFriend)
        FOR int i from 0 to the size of friends:
                IF friend.name equals friends[i] name:
                        FOR int j from i to size of friends
                                friends[j] = friends[j+1]
                        END
                        friends.size--
                ENDIF
        END
```

UI:

https://www.figma.com/file/JPY2REeACxWjkEOt5V0nfn/Design-II?type=design&mode=design&t=S3rTa2rfxOwIAp0W-1