FriendQuest: Social Networking using User Matching Algorithms and Leaflet JS.

Gautami Gupta
Department Of Computer Engineering
Atharva College Of Engineering
Mumbai, India
gautamigupt0620@gmail.com

Dipti Jalgaonkar
Department Of Computer Engineering
Atharva College Of Engineering
Mumbai, India
diptijalgaonkar03@gmail.com

Omkar Patil
Department Of Computer Engineering
Atharva College Of Engineering
Mumbai, India
omidevo93@gmail.com

Nilesh Phadtare
Department Of Computer Engineering
Atharva College Of Engineering
Mumbai, India
nphadtare88@gmail.com

Prof. Bhavna Arora
Department Of Computer Engineering
Atharva College Of Engineering
Mumbai, India
bhavnaarora@atharvacoe.ac.in

Abstract

In an era where digitalization has transformed social interactions, many individuals struggle to find companionship for everyday activities, leading to increased feelings of isolation. FriendQuest is a web-based platform designed to bridge this gap by connecting people based on shared interests and availability for activities such as shopping, social gatherings, and movie outings. The platform utilizes an intelligent matching algorithm to recommend suitable companions while ensuring a secure and user-friendly experience through robust safety mechanisms and detailed user profiles. Leaflet JS is integrated for interactive location-based services, enhancing the platform's efficiency in identifying potential activity partners within a user's proximity. Additional features such as real-time chat using Socket.io, GPS location tracking and interactive maps with nearby user recommendations further enhance the user experience. By fostering meaningful social connections, FriendQuest aims to enhance emotional well-being and encourage active participation in social activities.

Introduction

The rapid advancement of digital technology has significantly influenced the way individuals interact and build relationships. Despite the ease of online communication, many people still face challenges in finding suitable companions for everyday social activities. Engaging in routine tasks such as shopping, attending events, or watching movies alone can lead to social isolation, reducing overall well-being. The lack of a structured platform to connect individuals with shared interests exacerbates this issue.

FriendQuest is a social networking platform designed to facilitate companionship for such activities. It leverages user matching algorithms to recommend compatible partners based on common interests, availability, and location. By integrating Leaflet JS, the platform provides an interactive mapping system that enables users to find activity partners nearby. Additionally, the platform incorporates security features such as user verification, encrypted communication using JWT, real-time chat functionality using Socket.io, and GPS-based location tracking to ensure a safe networking experience.

This research explores the development and impact of FriendQuest, emphasizing the role of user matching algorithms, real-time communication, and location-based services in fostering real-world social interactions.

Results and Discussion

FriendQuest Flow

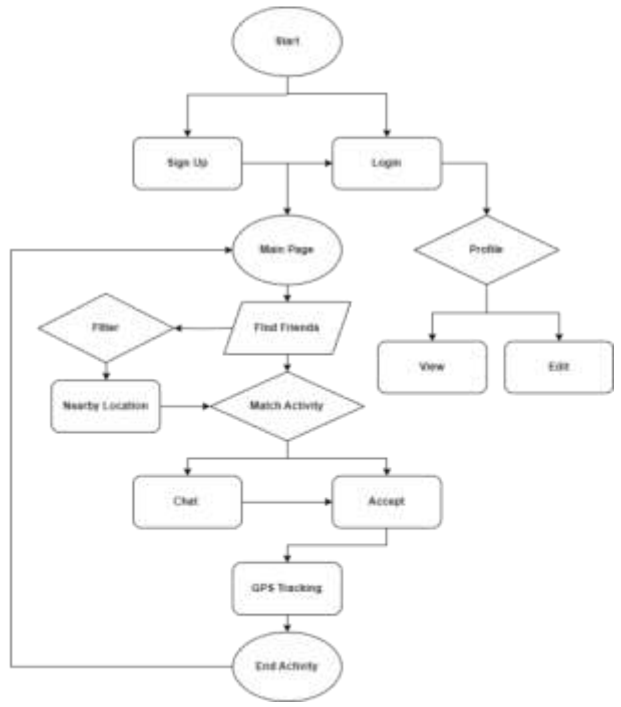

Fig.1. FriendQuest Flowchart

1. Start
The user opens the FriendQuest application on their device. The home screen displays essential options, including login and sign-up prompts for new users.

2. User Authentication

i. Sign Up: New users register by providing their email, phone number, and setting up a secure password. A
verification process via email or OTP ensures security.

ii. Login: Existing users securely log in using their credentials, utilizing encrypted authentication (JWT) for enhanced protection.

If the user forgets their password, an option for password recovery is available.

3. Main Page Navigation

i. Profile Management: Users can access their profile, update personal information, add profile pictures, and adjust privacy settings.

ii. View Profile: Displays user details, activity preferences, and past interactions.

iii. Edit Profile: Allows modifications to interests, bio, and availability.

iv. Find Friends: The user searches for potential activity partners based on matching preferences and loction.

4. Finding a Companion

i. Apply Filters: Users refine search results using filters such as:
Common interests (e.g., sports, dining, movies, travel)
Availability for activities (time slots)
Proximity to their current location

ii. Search Nearby Locations:
Uses Leaflet JS integration to visualize potential matches on an interactive map.
Allows manual location input to find companions in a specific area.

iii. Match Activity:
An AI-based recommendation system suggests compatible users based on shared interests and past interactions.
Users receive match suggestions and can review profiles before initiating communication.

5. Interaction & Communication

i. Chat with Matched User:
Real-time chat functionality powered by Socket.io enables seamless communication.
Users can exchange messages, share images, and discuss potential activities.
Integrated safety features like message reporting and blocking ensure a secure chat environment.

ii. Accept Invitation:
Once users agree on an activity, they can send and accept invitations within the chat interface.
A confirmation notification is sent to both participants.

6. Activity Execution

i. GPS Tracking Enabled:
Users can opt-in for live location sharing for safety purposes.
The GPS tracking feature ensures users reach the designated meeting point efficiently.

ii. Activity in Progress:
Users engage in the planned social activity.
The app allows participants to check in, update status, and confirm attendance.

iii. End Activity:

Once the activity concludes, users mark it as completed.

The system updates user history and logs successful interactions.

Feedback and rating options are provided to help improve future matches and refine recommendations.

This structured flow represents the technical workflow of FriendQuest, ensuring seamless user interaction, safety through real-time communication, and efficient activity coordination using location-based services

User Matching Algorithm Formula

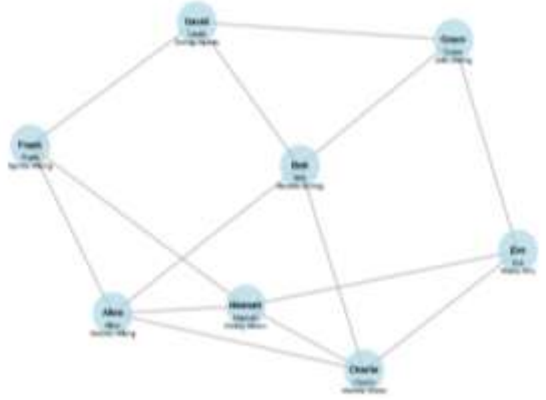

Fig.2. FriendQuest User Network

(Matching users based on similar activities)

To match users based on **location** and **shared interests**, we can use a **similarity score** that combines both factors.

The formula can be structured as:

$$Score(U_1, U_2) = w_1 \times S_{interest}(U_1, U_2) + w_2 \times S_{location}(U_1, U_2)$$

Where:

- U_1, U_2 are two users being compared.
- $S_{interest}(U_1, U_2)$ is the **Interest Similarity Score** (based on common interests).
- $S_{location}(U_1, U_2)$ is the **Geographical Proximity Score** (based on distance).
- w_1 and w_2 are weights to balance the importance of interest vs. location (e.g., $w_1 + w_2 = 1$).

Step 1: Interest Similarity Score (Jaccard Similarity)

We use **Jaccard Similarity** to measure the overlap between the users' interests:

$$S_{interest}(U_1, U_2) = \frac{|\, I_{U1} \cap I_{U2} \,|}{|\, I_{U1} \cup I_{U2} \,|}$$

Where:

- I_{U1} and I_{U2} are the sets of interests for user 1 and user 2.
- $|\, I_{U1} \cap I_{U2} \,|$ is the number of common interests.
- $|\, I_{U1} \cup I_{U2} \,|$ is the total number of unique interests.
- The result ranges from **0 (no shared interests) to 1 (perfect match)**.

Step 2: Location Similarity Score (Gaussian Distance Function)

We calculate the location similarity based on the geographical distance between users. A common method is using a Gaussian function:

$$S_{location}(U_1, U_2) = e^{-\frac{d(U_1, U_2)^2}{2\sigma^2}}$$

Where:

- $d(U_1, U_2)$ is the geographical distance between the two users (Haversine formula can be used).
- σ is a scaling parameter to control how distance impacts the similarity score.
- The score is **higher for closer users and decreases exponentially as distance increases**.

Step 3: Final Matching Score

After computing both similarity scores, the final score is:

$$Score(U_1, U_2) = w_1 \times S_{interest}(U_1, U_2) + w_2 \times S_{location}(U_1, U_2)$$

- If **interest matching is more important**, set $w_1 > w_2$ (e.g., 0.7 and 0.3).
- If **proximity is more important**, set $w_2 > w_1$.

Working of Leaflet

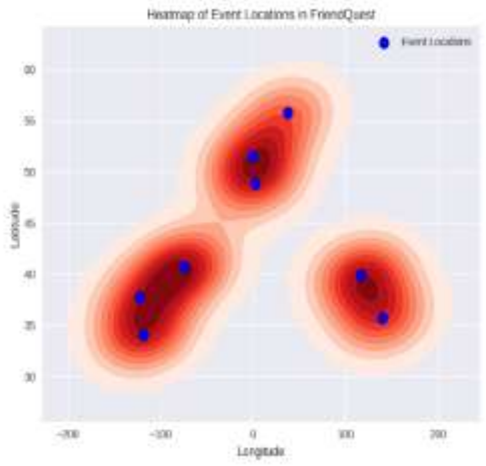

Fig.3.Heatmap of Event Locations in FriendQuest

Leaflet is a lightweight JavaScript library for creating interactive maps in web applications. It allows adding tile layers, markers, popups, and handling user interactions like clicks and zoom. It is widely used for location tracking, data visualization, and geospatial applications.

1. Haversine Formula (Great-Circle Distance)

The **Haversine formula** calculates the shortest distance between two points on the Earth's surface using their latitude and longitude. This formula is essential for determining the proximity between geographical locations, such as finding the distance between users on a map.

Formula:

$$a = \sin^2\left(\frac{\Delta\phi}{2}\right) + \cos(\phi_1) \cdot \cos(\phi_2) \cdot \sin^2\left(\frac{\Delta\lambda}{2}\right)$$

$$c = 2 \cdot \text{atan2}\left(\sqrt{a}, \sqrt{1-a}\right)$$

$$d = R \cdot c$$

where:

- R = 6371 km (Earth's radius)
- $\Delta\phi = (lat_2 - lat_1)$ in **radians**
- $\Delta\lambda = (lon_2 - lon_1)$ in **radians**
- ϕ_1, ϕ_2 are latitudes converted to **radians**
- d is the distance between two points in **kilometers**

Steps:

1. Convert latitude and longitude to **radians**.
2. Calculate differences $\Delta\phi$ and $\Delta\lambda$.
3. Compute a using sine and cosine.
4. Determine c using the atan2 function.
5. Multiply by R to get the distance in kilometers.

2. Nearest Neighbor Formula

The **Nearest Neighbor formula** identifies the closest point (person) to a specific location by comparing distances. It is widely used in mapping applications to find the nearest user, service, or facility.

Formula:

$$\text{Nearest} = \min(d_1, d_2, d_3, \ldots, d_n)$$

where:

- d_i represents the Haversine distance from the user's location to each other person.

Steps:

1. Calculate the **Haversine distance** for each person relative to the user.
2. Identify the person with the **smallest** distance as the nearest neighbor.

Working and Mechanism of Socket.io in a Real-Time Chat Application

Socket.io enables real-time communication using **WebSockets** and falls back to HTTP polling if necessary. It follows an **event-driven architecture** where clients (browsers) and servers exchange messages through predefined events.

1. Core Mechanism of Socket.io

Socket.io relies on **WebSocket technology** for full-duplex communication. If WebSockets are not supported, it automatically switches to **long polling**.

Formula for Message Transmission

The core idea of a chat system is to ensure **instant delivery** and **synchronization** of messages across multiple users. The delay (latency) in a message delivery system can be approximated as:

$$T_{delivery} = T_{emit} + T_{transmit} + T_{receive} + T_{process}$$

Where:

- T_{emit} = Time taken for the client to send the message event to the server.
- $T_{transmit}$ = Network transmission time.
- $T_{receive}$ = Time taken for the server to receive the message.
- $T_{process}$ = Time taken for the server to process and broadcast the message to other clients.

Since WebSockets maintain an **open connection**, $T_{transmit}$ is minimized compared to traditional HTTP polling.

2. Steps in Socket.io Real-Time Chat

Step 1: Establishing a Connection

- A client (browser) initiates a connection to the server using **WebSockets**.
- The **server acknowledges** the connection and assigns a **unique socket ID** to the client.

$$Connection_{socket} = Handshake(Client, Server)$$

Step 2: Sending Messages

- A user sends a message using an event like sendMessage.
- The client emits the message event along with the message data.

$$Emit(Client, Server, Data)$$

Step 3: Message Broadcast

- The server listens for sendMessage events.
- Once received, the server **broadcasts** the message to all connected users.

$$Broadcast(Server, All\_Sockets, Data)$$

Step 4: Receiving Messages

- Other users receive the message event and update their UI in real time.

$$Receive(Client, Data) \rightarrow UpdateUI$$

Step 5: Disconnection Handling

- When a user leaves, the disconnect event is triggered.
- The server updates the user's status and informs other users.

$$Disconnect(Client) \Rightarrow Inform(Server, Peers)$$

3. How WebSockets Reduce Latency

Compared to HTTP polling, where the client continuously requests updates, WebSockets maintain an open connection, reducing **network overhead**. The efficiency is represented as:

$$Efficiency = \frac{1}{Latency_{Polling} - Latency_{WebSocket}}$$

Where:

- **Polling Latency** = Delay due to repeated HTTP requests.
- **WebSocket Latency** = Minimal delay as the connection remains open.

This ensures **real-time updates** with minimal bandwidth usage.

4. Additional Features Using Socket.io

1. **Rooms & Namespaces**
   - ○ Users can join specific **chat rooms** to separate conversations.
   - ○ The formula for room assignment:

$$Room_{id} = Hash(User\_ID, Chat\_ID)$$

2. **Typing Indicators**
   a. A typing event is emitted when a user types, alerting other users.
3. **Message Storage & Database Synchronization**
   - ○ Messages can be stored in **MongoDB** for persistence.

OUTPUT:

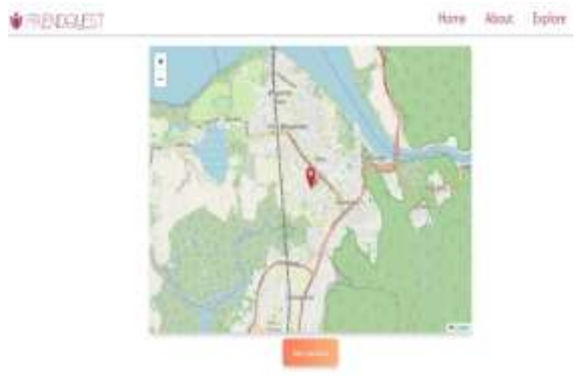

Fig.4.Map implementation using Leaflet

Conclusions

FriendQuest is an innovative solution that leverages user matching algorithms, Leaflet JS, real-time chat, and GPS tracking to facilitate meaningful social interactions. By addressing the growing challenge of social isolation, the platform provides users with an efficient way to find companions for shared activities. The integration of intelligent matchmaking, live messaging, and location-based recommendations enhances usability and accessibility. Future developments may include AI-driven behavioral analysis and enhanced

security protocols to further optimize user experience. FriendQuest demonstrates the potential of technology in fostering real-world social engagement beyond traditional social networking paradigms

References

1] H Sun, N Wang, J Jia, J Huang, H Xiong, L He, X Liu, S Zhang, S Qiao, J Zhao “Platform-Oriented Event Time Allocation” 09-12 May 2022.

2] Stephen Ricken, Louise Barkhuus, Quentin Jones “Going Online to Meet Offline: Organizational Practices of Social Activities through Meetup” June 2017.

3] Vaughn, Danielle, MeetUp and Social Capital: Building Community in the Digital Age June 2015.

4] Mehmet Taş ,Alper Kiraz “A Model for the Acceptance and Use of Online Meeting Tools” October 2023.

5] Xing Wan , Ashish Kumar Jha , Nikolai Kazantsev , Wai Fong Boh “Online-to-Offline Platforms: Examining the Effects of Demand-Side Usage on Supply-Side Decisions” March 2023.

6] Kanika Tyagi, Deepali Singh, Bhumi Sharma “De-Link Chat Application using MERN stack and Socket.io” Vol (5), Issue (5), May 2024.

7] Jing Ren, Feng Xia, Xiangtai Chen, Jiaying Liu “Matching Algorithms: Fundamentals, Applications and Challenges” March 2021.

8] Samuel Ngugi “Leaflet in Practice: Create webmaps using the JavaScript Leaflet library” July 2023.

9] Gang Zheng, Yilu Liu, Ghadir Radman “Wide area frequency based generation trip event location estimation” 22-26 July 2012.

10] Chunlai Ma, Chao Chang, Tao Ma, Jun Huang, Zhao Niu “User Identity Matching for Multisource Location Data” 13-16 October 2021.

11] S. K. Goyal, S. Bhattacharya, and A. Kumar - “A Framework for Event Recommendation in Online Social Networks” (IEEE Transactions on Computational Social Systems, 2019).

12] M. D. Ekstrand, J. T. Riedl, and J. A. Konstan – “Social Recommendation Systems for Online Communities” (IEEE Internet Computing, 2017).

13] J. Li, Y. Zhang, and J. Tang – “Personalized Event Recommendation in Social Networks: A Survey” (IEEE Transactions on Knowledge and Data Engineering, 2020).

14] H. Liu, J. Zhang, and X. Li - “A Social Event Recommendation System Using Collaborative Filtering” (IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2017).

15] Y. Zhang, J. Tang, and J. Li - “Social Network Analysis for Event Recommendation” (IEEE Transactions on Knowledge and Data Engineering, 2018).

16] V. Agarwal and K. K. Bharadwaj, "A Collaborative Filtering Framework for Friends Recommendation in Social Networks Based on Interaction Intensity and Adaptive User Similarity," *Social Network Analysis and Mining*, vol. 3, pp. 359–379, 2013.

17]G. Liao, X. Huang, N. N. Xiong, and C. Wan, "An Intelligent Group Event Recommendation System in Social Networks," *arXiv preprint arXiv:2006.08893*, 2020.

18]M. Wang, X. Zheng, Y. Yang, and K. Zhang, "Collaborative Filtering with Social Exposure: A Modular Approach to Social Recommendation," *arXiv preprint arXiv:1711.11458*, 2017.