Comma: A real-time geolocation-based random chatting app @xiangyi

1. Introduction

1.1 Project Overview

Comma, a real-time chatting app that allows people to start random chats with people nearby, is set out to be a minimalist, comfy, and secure social app that allow people to start chatting without too much in mind. Besides chatting, we are also gonna establish an activity functionality where people can post pictures, videos, or texts which will be automatically pushed to other users within a certain distance.

1.2 Objective

Social platform is one of the most popular and variegated categories of softwares on today's market. We have gigantic companies like Meta that owns billions of users around the globe, dating apps like Tinder that many young people love using, community apps like Discord or Reddit, and social media like Twitter or Weibo. You name it.

Despite the variety and maturity of the current market status, there's a viral internet vernacular in China called social-phobia, where people are too scared or burdened to start some conversation with another person. Thus we aim to create a space where people can have the chance to start random talk with another human-being that's nearby, and really have no burden on their mind, since it's totally anonymous.

1.3 Expected Customers and Market

Anyone at anywhere can access our product. There's nothing too much to say about this, except that there's a big chance users might be unable to find anyone nearby using this app when the app has few users. They can choose to have a short conversation online and abort when they are bored, or they can arrange some meetup since they would be really close. (We'd like to set the initial range to 5km).

As for the market prospect, since we are also implementing some activity or posting feature, and this app is location-based, we plan to provide advertisement service to merchants or organizations who would like to invite people to galleries or shows. Commercials will be embedded into the information stream, and there will be one commercial in approximately 8 to 10 feeds.

1.4 System Features

Main functionalites of our product consist of geolocation positioning, chatting, posting and information feed. Users can start a conversation with random people within 5km around them. They can post text, pictures, and videos anytime they like, and they can receive feeds posted by people 5km within them.

Currently we are not considering allow users to follow or establish friend connections with other users. They can exchange contacts if they like, but for now, they wouldn't be able to re-find the person they were talking to if they terminated their conversation. We leave this to karma for now.

2. Background

In 2013, Snapchat went viral. It's a chatting app that automatically deletes your messages after they have been read by your friends. In retrospective, the tremendous popularity it gained was largely due to the "lack" of functionality. In a world where social apps are already dominated by behemoths like Facebook or Instagram, it managed to acquire a large number of users again by dropping features instead of adding, that is, by the *via negavita* approach.

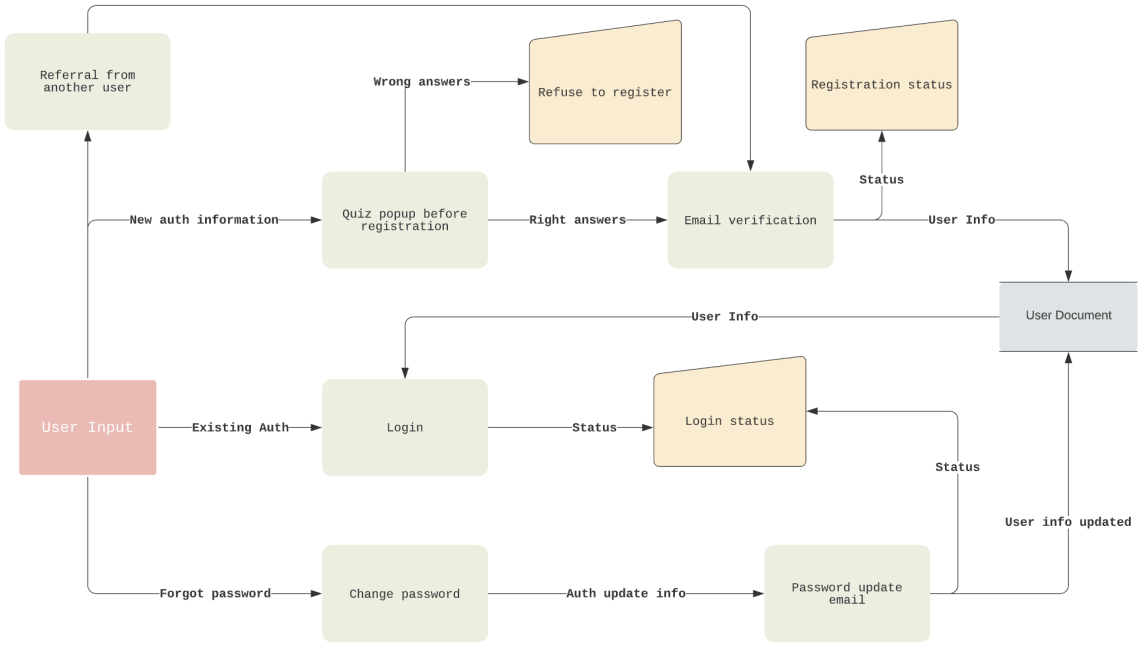
Today, in both China and the western world, social platforms are monoployed by a few tech giants. This kind of monopoly is very hard to break since it's not the product but people's contacts that are retaining them to their current social platforms. We are not in the mind of breaking this monopoly, either. Rather, we are just providing an alternative for people nearby to communicate.

Also, all mainstream hardware suppliers like Apple, and software providers like Chrome and Brave are making the privacy policy more restrictive. Old-fashioned customized user-tracking advertisements are on the decline. Our product will push commercials to local people, which is precise and has no worry on the limitation of the privacy policies.

3. Specification

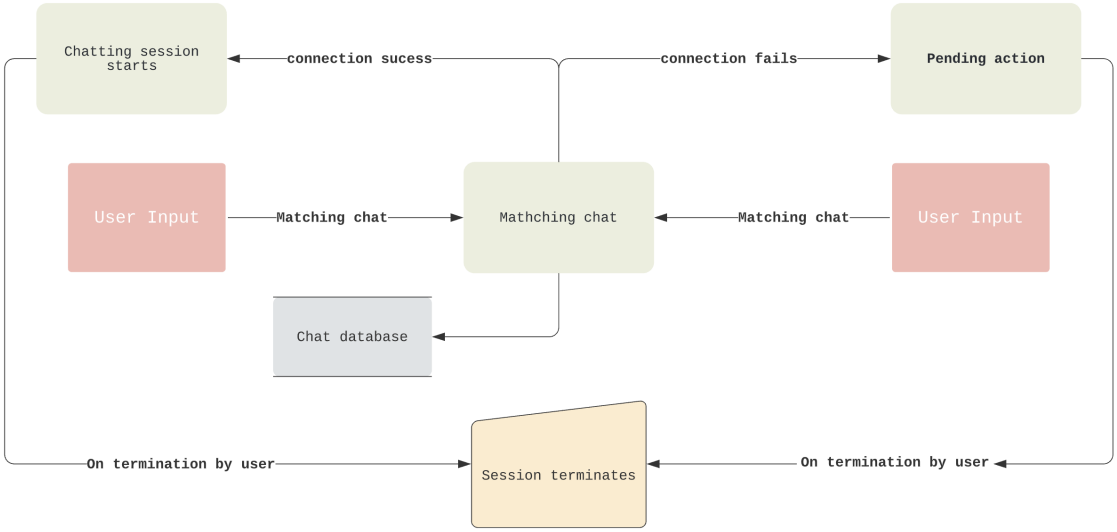
We give the data flow diagrams for: `Login/Registration`, `Chatting`, and `Posting`. Since we have not decided how to setup the backend for commercial services and transaction details, or how to model our data and APIs, we will not display them here.

3.1 User Login and Registration



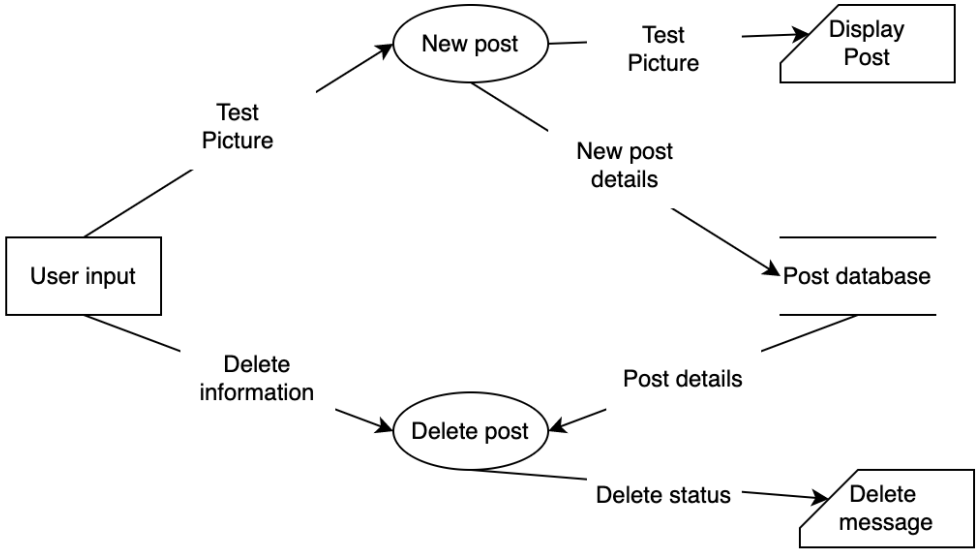
We'd like to walk users through a few quizzes before they create their account and enter the platform. This is mainly out of some vain efforts to prevent empathy-less people from saying something offending. The quizzes, however, will be made gentle enough that people won't be able to recognize its goal.

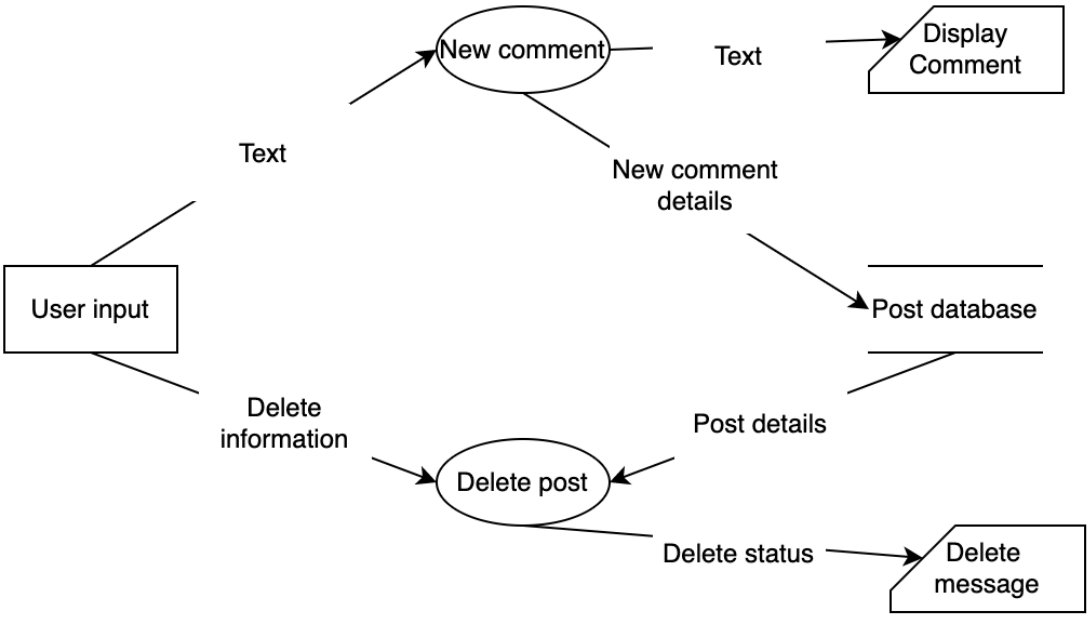
3.2 Chatting



Users are able to terminate any conversation anytime they want. If the connection fails before the session starts, session can also be terminated by the user.

3.3 Feed and comments





Users will only receive feed from people within 5km around them. Users can like and comment on the posts they receive. Details of the feed of posts and display of comments will be implemented afterwards.

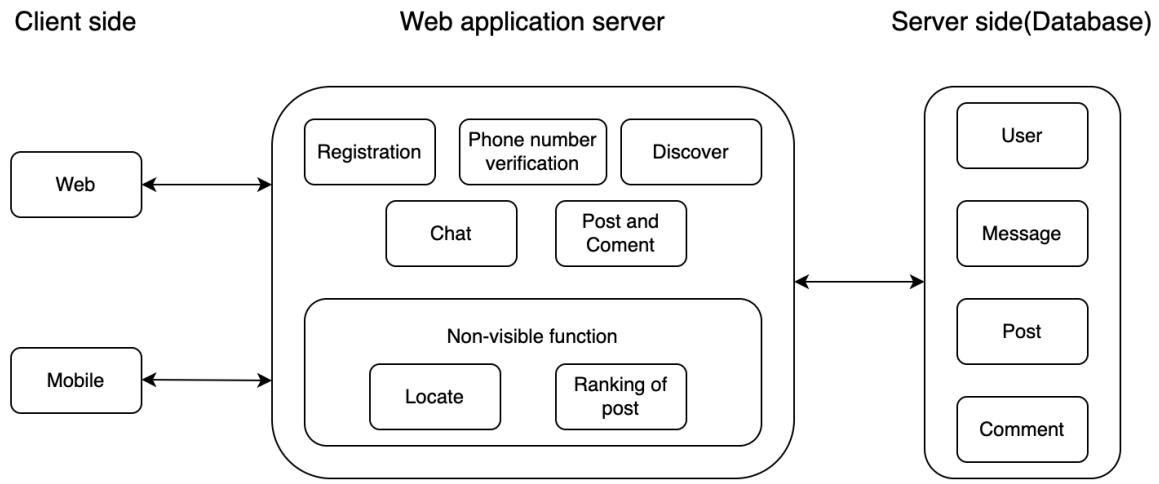
There are many details that can be difficult to be put into diagrams, such as blocking, flagging as action of users, tagging, searching, video playing in the post design, and many other implementation

details. In a word, this diagram serves no more than a draft in this whole project. We will not elaborate too much in these whiteboard diagrams. We will, however, diagramming this sub-processes in detail when implementing the front-end or the API if necessary.

4. System Architecture

We plan to ship our products into three platforms: the web, iOS, and Android. For the client-side we are deciding between Angular and React. If we choose React in the end we will use React Native for mobile deviced. If we choose Angular we will switch to Flutter instead. For the server-side, we are almost certain to dockerize the API code and use Kubernetes for them to run on a cluster of cloud servers. We are pretty sure to use MongoDB because Firebase would prevent Chinese users from accessing the app. If we are using Node.js/Deno.js to write our API we will use socket.io. If we use the Go language we will probably implement the realtime websocket library by ourselves. This will be dependent on the time remaining.

4.1 Architecture Diagram



There’s really nothing much to say about this graph. It’s pretty generic. Anyone with some software development experiences is likely to consider explanation on this graph trivial.

4.2 System Components

4.2.1 Frontend

Sure: Tailwind, TypeScript, Jest

Scenario with React: React, Remix, Storybook

Scenario with Angular: Angular, Flutter.

4.2.2 Functions

- User Login and Registration: we will require email plus phone verification (for Chinese users phone verification, for others email). We will either handle the email verification by ourselves or using a SaaS. We will resort to SaaS for phone verification.
- Geolocation positioning: we will request location access from the user. If it's on the web we will use browser's geolocation API. If it's on mobile devices we will use the native geolocaiton API. We will decide how to use the location data once we figure our how these API work and what kind of data structure it uses.
- Chatting: When the users starts requesting conversation with another user, the server will handle the geolocation data and push the matched person to him. As for the real-time chatting session, either we implement our own websocket logic with go, or we use socket.io with Node.js. People can terminate the conversation if they like to. They can also flag or block other people for malbehavior.
- User Feeds: users can post contents and they will be broadcast to people within a certain range. People who received the feeds can like, comment as they like. User can also block or flag users or content for malbehavior. Commercial use of the information feeds are not thought about clearly so we will not elaborate.

4.2.3 Backend Development

Sure: Docker, Kubernetes, Ansible

Not sure:

Databases: MongoDB, FaunaDB, Cassandra, MySQL, Redis

Language: Go, Node.js, Deno.js

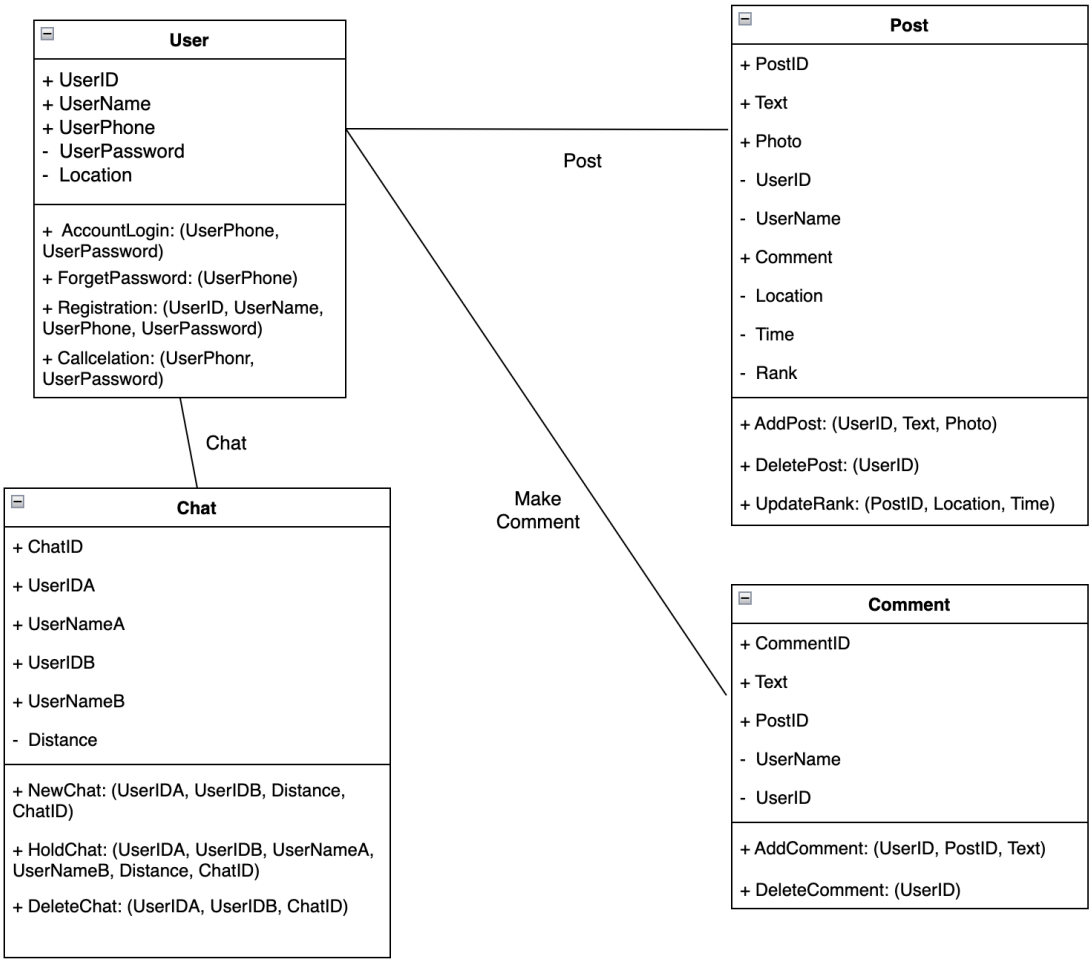
Network: WebSocket, socket.io

Frameworks: Rx.js, DynamoDB, and a few bunch of other common Node.js frameworks that help with the development.

Cloud provider, AWS/GCP/DigitalOcean, Aliyun/TencentCloud

Our product doesn't depend on connections between users much, thus we just did give graph databses any consideration.

4.3 Description of Major System Components by UML.



The UML diagram models the simplified version of the system functionalities mentioned above. Simple as it seems, a chatting app is not so easy to build. We just hide the undecided details from the graph.

5. References:

https://getstream.io/blog/build-chat-messaging-app/

https://thenewstack.io/rust-vs-go-why-theyre-better-together/

https://btholt.github.io/complete-intro-to-realtime/

https://theprimeagen.github.io/dev-productivity/

https://www.manning.com/liveprojectseries/websocket-based-chat-backend-in-go-ser

https://fireship.io/courses/firestore-data-modeling/

https://static.frontendmasters.com/resources/2019-09-24-golang/golang.pdf

https://www.reddit.com/r/node/comments/nx9qqr/deno_vs_nodejs_a_comparison_you_need_to_know/

https://socket.io/docs/v4/server-installation/

https://keenethics.com/blog/deno-vs-node-js-all-you-need-to-know

https://fireship.io/lessons/fauna-db-quickstart/

https://docs.fauna.com/fauna/current/

https://www.algoexpert.io/systems/fundamentals

https://docs.ansible.com/core.html

https://azure.microsoft.com/en-us/topic/kubernetes-vs-docker/#:~:text=Kubernetes is open-source orchestration,containers%2C deployed across multiple servers.

These are far from encompassing our experiences and research in for building this project. Besides contents on the public domains, we consulted many seasoned engineers both in person and online for technical decisions. And these are very helpful for us to progress with our research.