Juntos

Name : Shubham Jagdhane
Roll No : 18116

Problem Statement:

We have to design architecture for Chat Application.

My Chat Application name is **Juntos**. It's a spanish word that means together. It's a Mobile First App. (Planning for Web App)

I'm providing following services for **Juntos**:
Services:

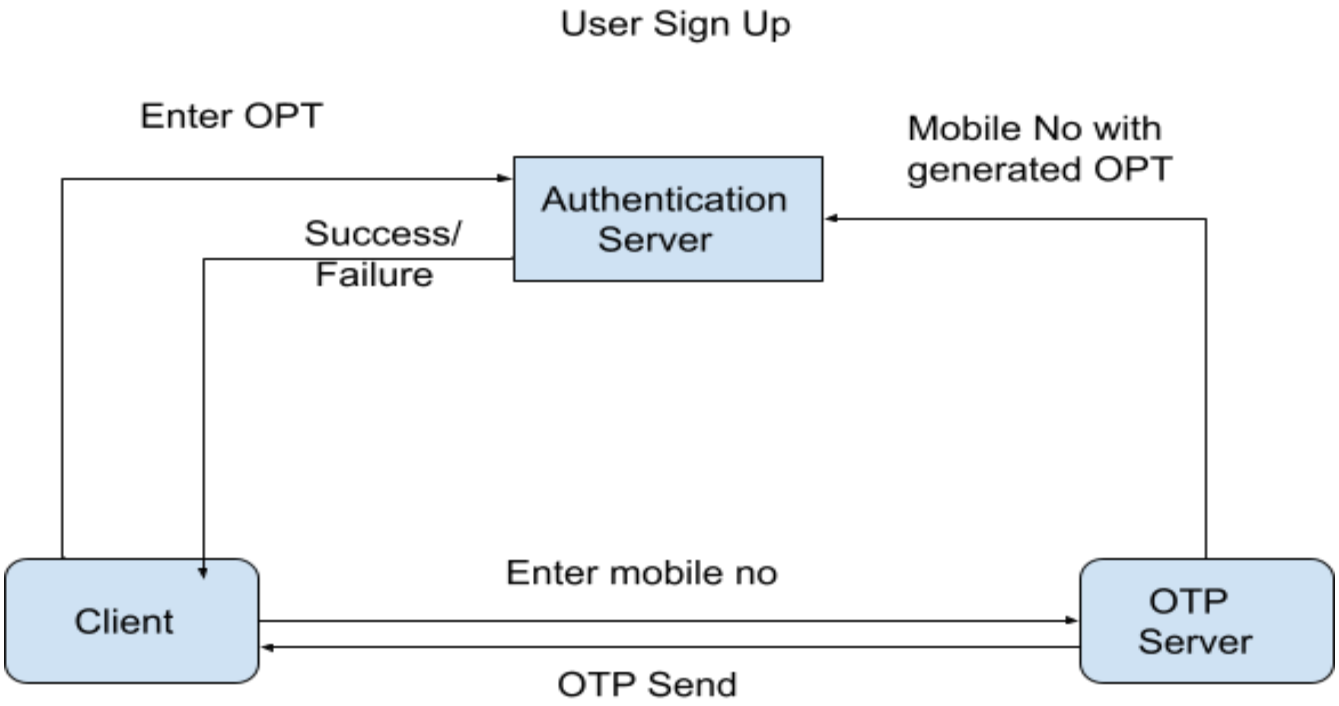
- One to one messages which include plain text, images and videos. (Encrypted messages)
- Group messages which include plain text, images and videos.
- Messages are permanently stored, but there is an option which deletes messages from last day, week or month automatically.
- Users are allowed to update their profile as an image file with specified dimensions.
- Users can set maximum 100 statuses which could be images, videos and those last for 24hrs. Once a user reaches 100 statuses in 24hrs, alert him/her with a limit exceeding.
- Users can see other users last seen, online.
- Users can reply to previous messages.
- Users can forward sent messages
- Users must have mobile number to login into application
- OTP will be sent to the user's mentioned mobile number for authentication.
- Users must have to give device accessibility for camera, contacts(mandatory) and notification.

- Users are allowed to see only those users' information which are in their contact list.
- Status privacy has my contact, only selected contact option.
- Users can delete his/her account.
- The backup of the data is going to store at the client side on his/her device.

Please take a look at the following diagrams

---------- Intentionally left blank ----------

Sign Up/ Login

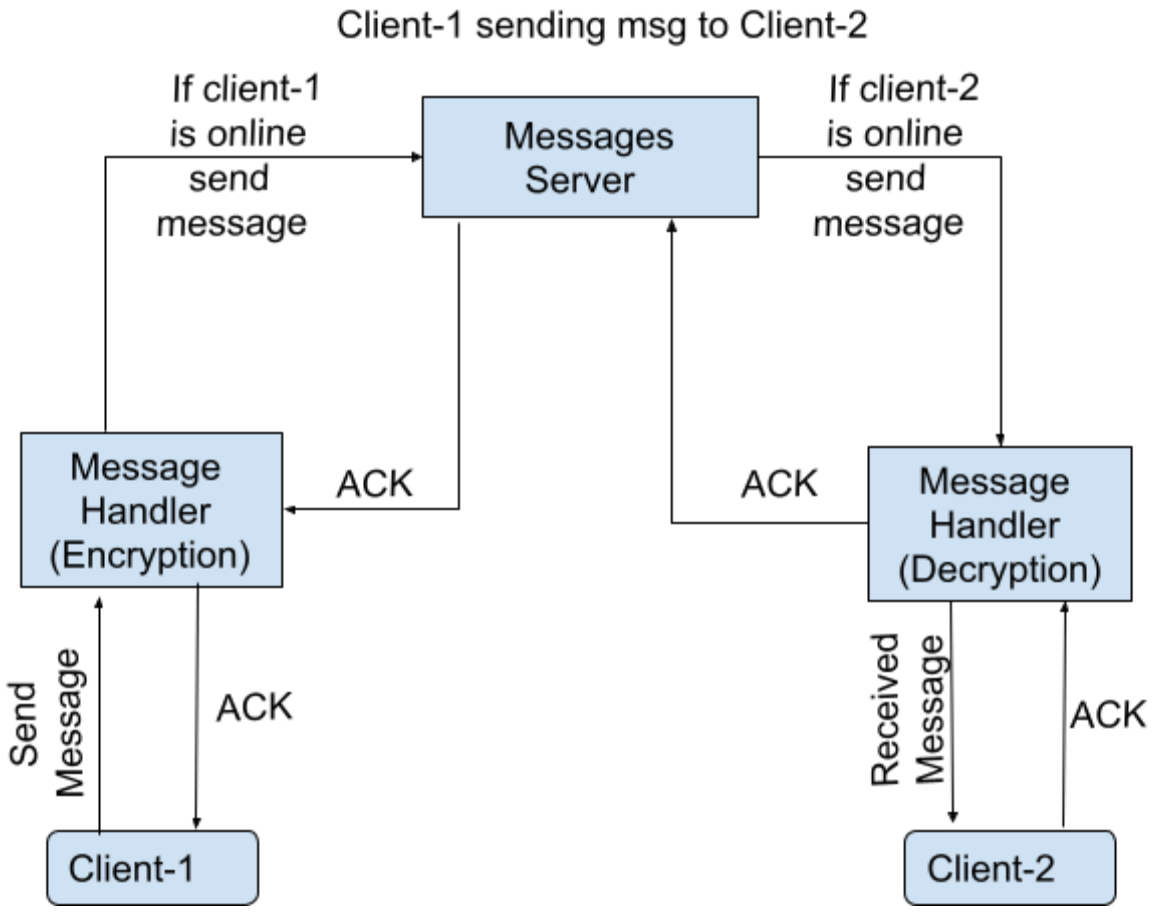


1) Clients are allowed to login using Mobile Number only.
2) For new users they must have a Junto App.
3) Clients must have to enter his/her mobile number.
4) He/She shall get an OTP from our OTP server, the respective mobile number with the generated OTP shall send to our Login Server.
5) Once a client enters valid and correct OTP he/she shall be able to logged in to our app else he/she shall get an authentication error.

Reasons to maintain OTP server and Login Server Different:

- There might but some day on which many users try to login into our App. So as we do have OTP servers which have to generate only OTP's it could have more ideal CPU and there are very less chances to fail that OTP service.
- We do have a backup server if an OTP/Login server fails.
- We are maintaining the server logging which helps us to track what might go wrong.

Message from Client-1 to Client-2



Our app can have the ability to send multiple messages to multiple clients. The above diagram gives you an idea about how our messaging server works.

- Working:

1) It is the most important part of our messaging server which can take messages from clients (whether they online/offline) and encrypt or decrypt. Once the sender becomes online then the handler sends a response to the server with the message.

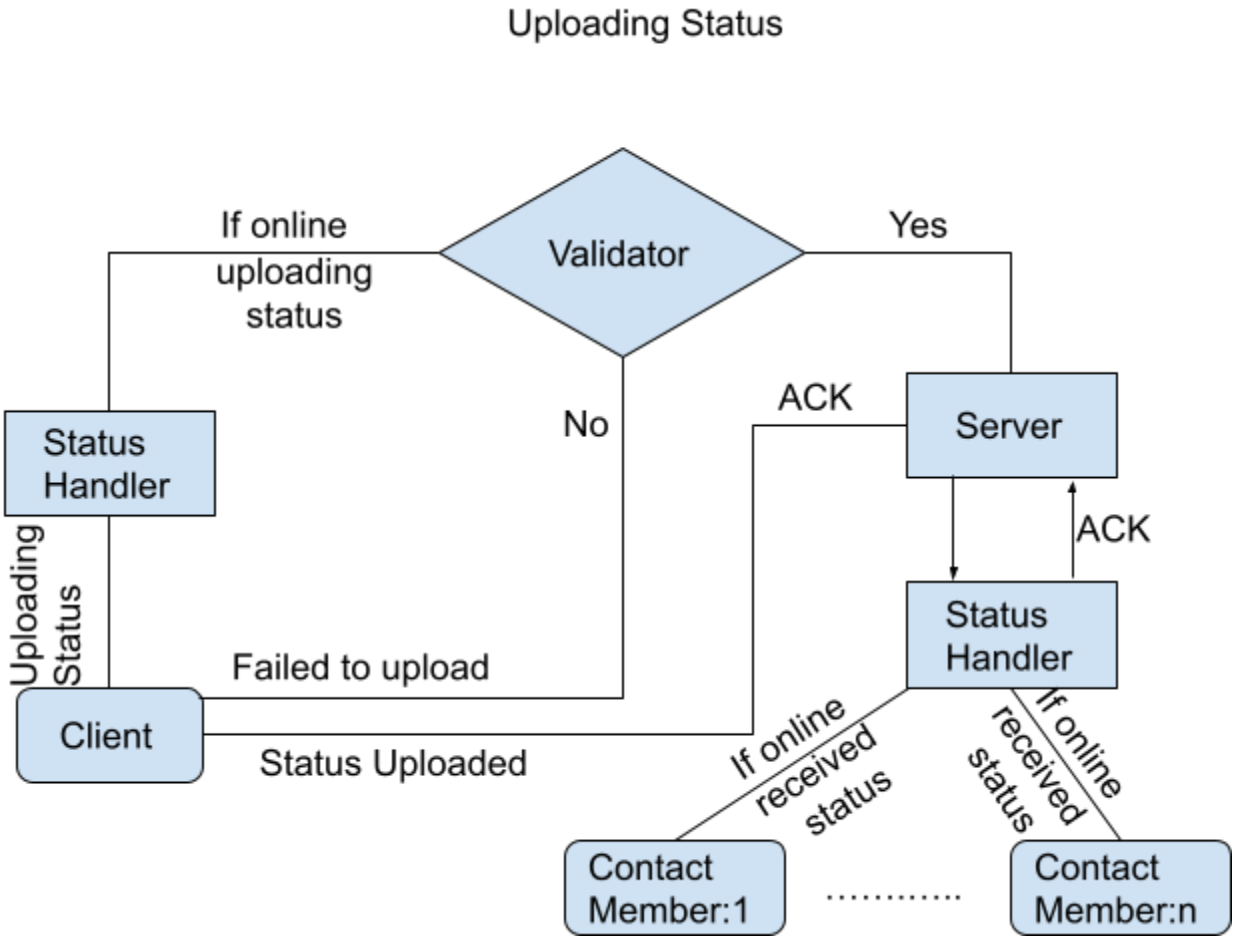
2) Messages server gives response to the sender client that his/her message is sent from his/her device and it sends that message to the message handler.

3) The message handler checks whether the receiver client is online, if he/she is then it decrypts the message and sends it to the receiver client.

4) Receiver client sends acknowledgement to message handler and in backward ways Messages Server sends acknowledgement to message handler from message handler to sender client.

---------- Intentionally left blank ----------

Upload status on Junto App



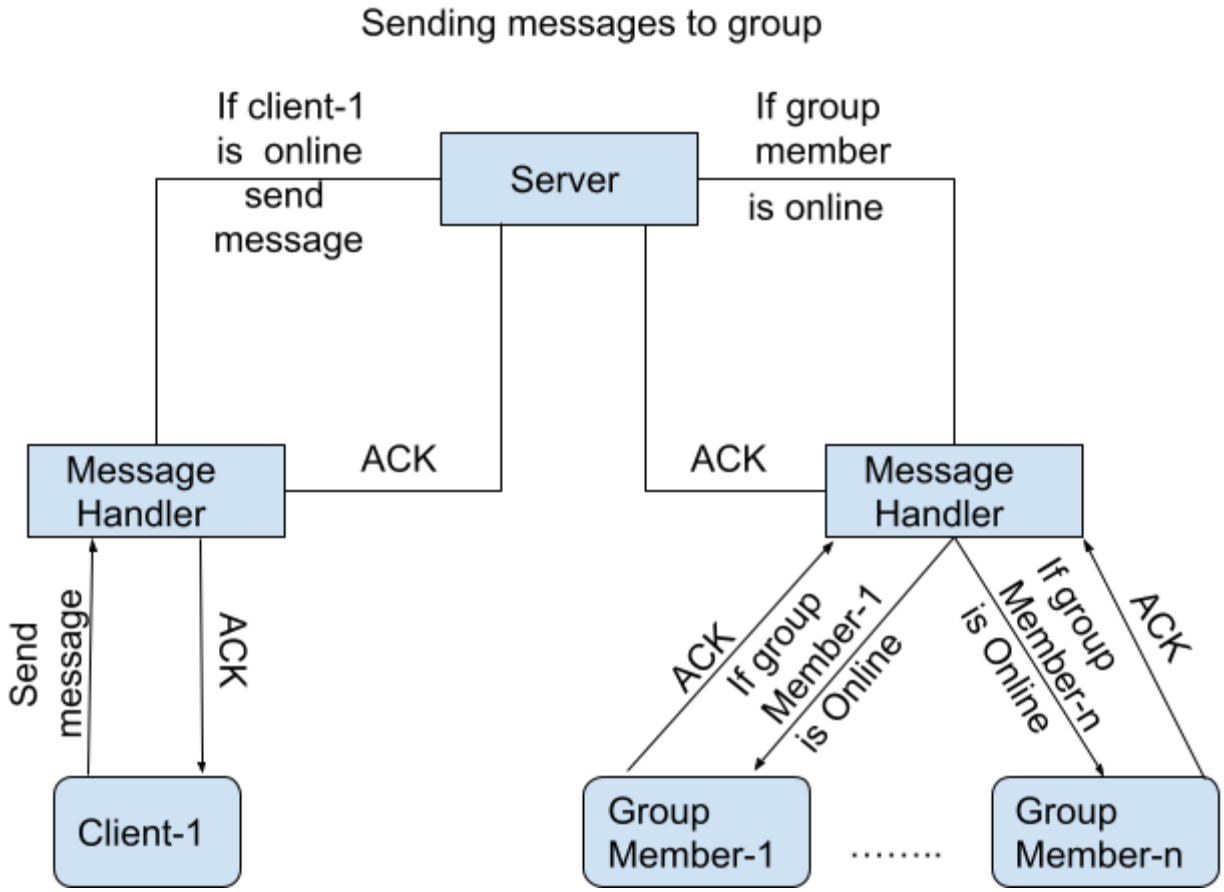
Uploading status is one of the favourite services we are providing.

- Working:

1) Client can set any status he/she wants but there are some limitations on maximum status count per day i.e 24hrs.

2) Once a client has done his/her part from the UI part, it first goes to the Status Handler which can hold that status. Once the client becomes online the status passes to the validator which validates the size of status, type of status. If everything is validation failed it gives the client alert otherwise Validator passes the status to the Server with contact information of that client and then passes status to Status Handler which then upload status to their contact list.

Send message to a Group

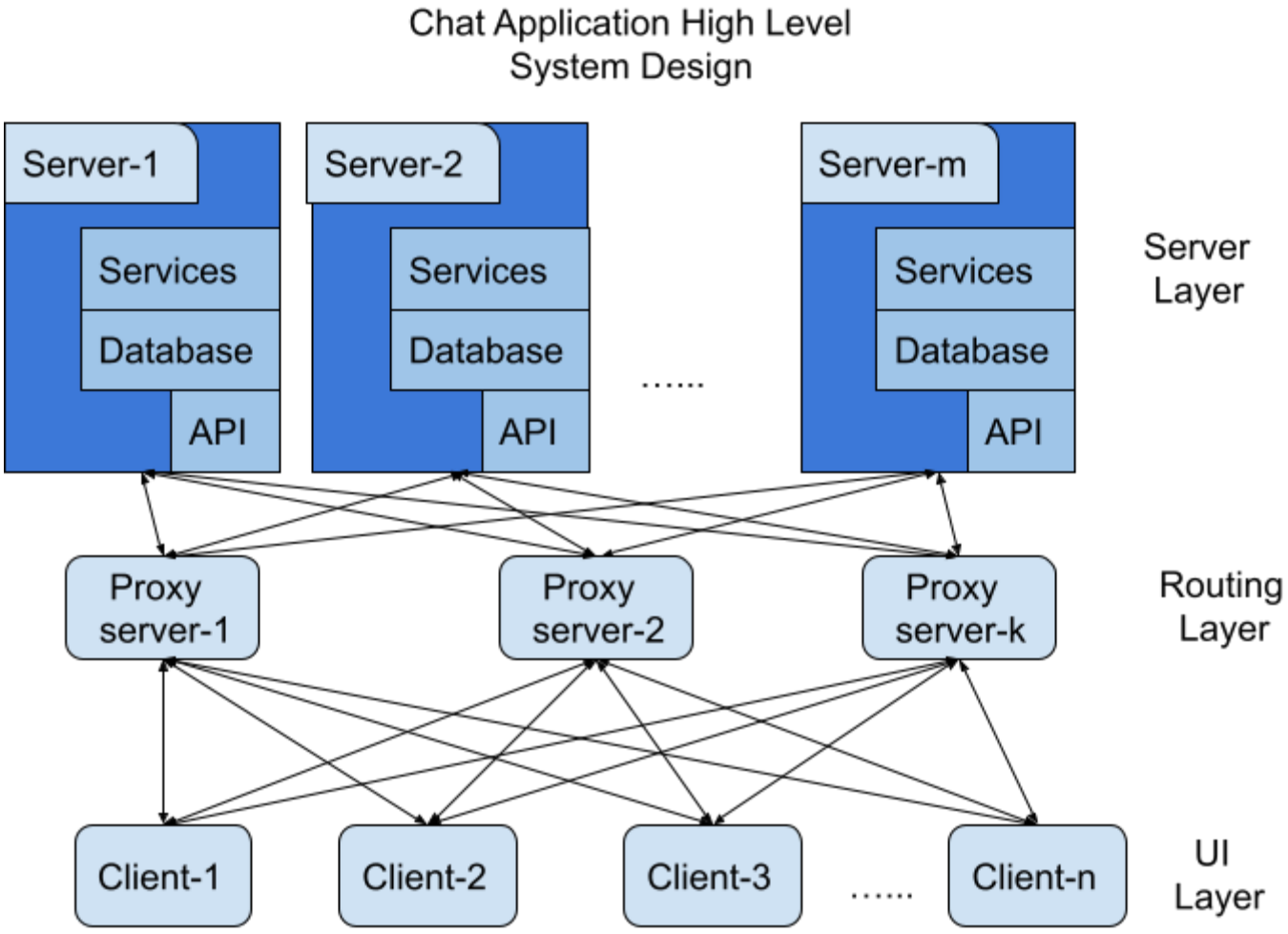


Only group members are allowed to send messages in the group.

- Working:
    1) It is the same as the client-1 sends message to client-2 only difference is at receiver side, message handler checks whether any group member is online and sends the message.
- Note:

    In group messaging the respect to messages is not getting encrypted or decrypted.

Junto High Level Design



As per the high level design of our diagram, our Junto App divided into Three layer architecture.

Layer-1: UI Layer

All user interaction parts of the App are handled at this layer. The UI part includes typing messages, select photos, videos, view statuses, see profile pictures, see contact list, online/last seen.

Layer-2: Routing/Proxy Layer

This is a very important layer of our App. This routes the clients request in an efficient manner so that everytime client can get high availability on our platform.

This layer sends a request of the client to the nearest the located server and if there is something wrong in that server, then it also handles the request in an intelligent way and passes the request to the closest server to the client.

It also manages the hit ratio of the server and balances the request with respect to total number of servers.

This layer also includes our security of application. It checks whether is there any network attack happening or not, any unknown or unauthorize request is interfering or not.

If for some reason it fails, then rest of the proxy servers balance the load and gives clients high availability on our platform.

Layer-3: Server Layer

This is the brain of our App, all massage, images, videos, compression/decompression of image/video.

As there might be millions/billions of user can use App, to give then high availability with low latency this servers helps our application to achieve the same.

These server can communicate to each other to share or to get data which is stored in frame format. As we are storing an images/videos in streams byte of data to achieve the low latency and high availability of the platform.

These server includes database, API which are running as services and we can increase those services when we hit with high traffic.

Our application is deployed on Cloud to achieve high scalability, the architecture of Application is written in the way that we can achieve high scalability by increasing the number of servers in server layer.

This layer also maintains the backup of our clients data. We are doing data replication for data recovery.

Technology:

Today we do have HTTP2, which is very fast and very powerful. I'm going to use HTTP2 for my HTTP request.

I’m going to use gRPC for my application because there might be cases that we are using different languages for different purposes and we have them to communicate and in future if we wanted to add more functionality it should be easy to add functionality without hampering the existing functionality.

Backend:

1) Erlang
2) Golang

Frontend:

I’m using hybrid frameworks with gRPC which can benefit us for the development process and make us faster, no worries about different functionality in different languages and gRPC can communicate with different languages.

1) Javascript
2) HTML, CSS
3) React.js

Database:

1) Sqlite3: To store data on client devices.
2) Redis: To store hot data for short period of time
3) MongoDB: To store streams of images/videos (one time insertion for each images/videos) and to store cold data.

Log maintaining:

As we have written application in such format we can see any movement happen for every request, those logs are stored in the databases which respect the events.

Incident Management:

As we are replicating data from our application and storing it into different servers so it can help us for disaster recovery.

High and easy scalability:

As per discussed in the high level design we can achieve high scalability by increasing the number of server which is very easy in Cloud.