ChatBox: Distributed Messaging Platform

Ramita Maharjan

University of Memphis, Tennessee, rmhrjan1@memphis.edu

Abstract. The way we communicate about something greatly reveals our understanding about it. Working together with several other people as a team is an everyday requirement for most of the people. In addition to being able to receive the messages instantly in real time, it is very important to have those messages saved for future reference and retrieval. ChatBox is a distributed platform based on data-centered architecture that allows multiple users to communicate over a network by sending text messages to each other and the chat may be one to one, many to many or all to all. This exchange of messages is independent of other users and depends on the publish and delivery of data to and from a shared repository. The messages are broadcast in the form of events to only the authenticated private communication channels of recipients. ChatBox, thus helps people to achieve their goals in their workplace and learning environments by providing a distributed messaging platform.

Keywords: ChatBox, Distributed System, Data-centered architecture, Socket, Session, Broadcast

1 Introduction

There are numerous messaging applications that help a team to collaborate in classroom, workplace or for general purpose activities. WhatsApp, Viber, Facebook Messenger, Hangouts, Slack, WeChat etc. are some of the widely used instant messaging applications today for personal messaging and work-related communication. In this unprecedented situation caused by the global pandemic of Corona virus, when people are required to maintain social distancing with each other and work or study from home, the importance of such collaboration tools has been realized now more than ever. As a result, the average time a person spends on sending messages to each other has surged sharply.

ChatBox is a distributed platform for multiple systems/users to share text messages between each other with the purpose of exchanging ideas, knowledge and increase work productivity. In order to guarantee the persistence of the exchanged messages, they need to be stored somewhere from where users may access them whenever required. A shared data space is required to store the messages so that even if a user is not online at the time some other users send him/her messages, those sent messages can be retrieved from the data store easily once the user gets online.

Other practical applications of this project can be in the teaching-learning environments. It is often observed that students are more likely to interact with the teachers and ask their doubts if they can do so anonymously. This is perhaps because they feel embarrassed in front of the whole classroom to ask simple concepts or doubts. In this context, the students may join the chat box anonymously and ask the questions easily. A teacher can join the chat and then answer the doubts without knowing who asked the question.

The followings are the brief overview of background technical terminologies and their relation to this project.

1.1 Distributed System

A distributed system is a set of independent computers linked by a network which communicate and co-ordinate with each other by passing messages in order to achieve common goals. In this ChatBox platform, multiple users are independent systems which are linked by a network and they coordinate with each other by sending messages. Here, the common goal is to collaborate and share ideas by conducting proper communication among different people, be it the team members in office or the students of a class.

1.2 Data Centered Architecture

Distributed systems are very complex, and so they need to be organized properly. Software architecture is the logical organization of how software components are organized and how they interact. Data centered architecture is a software architecture in which components in the system communicate via a common repository which can be active or passive. In this project, the primary components are the generators of chat and the consumers of chat. The producer (a user sending a text message) produces a message and sends it to the central repository from where the consumers pull the message from by joining the network and sending requests to server. The persistence of data is maintained by implementing the data centered architecture and it guarantees the delivery of data.

1.3 Socket

In the standard internet protocols, a socket address is the combination of an IP address and a port number. If a program binds a socket to a source address, the socket can be used to receive data sent to that address [2]. In this scenario, the socket is used as endpoint of a two-way communication link between the server and client systems distributed over the network.

1.4 Session

In computer science, a session is a reference to a certain time frame for communication between two devices, two systems or two parts of a system [5]. A session is a temporary information interchange meaning that a session is established and ended at certain points of time. In this scenario, maintaining login session is relevant to keep track of active users. A login session is the time period between login and logout in the application.

1.5 Broadcasting

Broadcasting is a method of transferring a message to all the recipients simultaneously. In this scenario, depending upon whether a sender wants to send the message to all of the users, a group of users or an individual user, the text message is broadcast to only the intended users by transmitting the message to only the authentication communication channels. A communication channel is an instance of a communication between components within a messaging network.

2 Related Works

Although it is difficult to pinpoint the very first chat system, the earliest of online chat facilities is believed to be Talkomatic (1973) on the PLATO system at the University of Illinois. This allowed 5 people to chat simultaneously on a plasma display (5 lines of text + 1 status line per person) and each person could watch the fellow chatters type their messages, letter-by-letter [3]. Over time, the messaging platforms are developed so rapidly that today people can send not only the text messages through chat but also images, video, web links, documents and many more.

Numerous works can be found related to chat application and most of them focuses on client server model of distributed system. Similar work has been done to develop a LAN chat messenger by Ibrahim and his colleagues in their paper [1]. The paper explains the implementation of two-tier client server architecture using sockets to collect message in a local area network.

Another relatable work to this project was done by El-Seoud and Taj-Eddin in developing an android mobile Bluetooth chat messenger [4]. In [4], they have described how the chat software was developed as an interactive and collaborative learning aid specially for the students. The messenger works by creating a server and then waiting for another client to connect to it or ask another device to chat with it.

This project is based on data centered architecture to preserve the persistence of messages in the common shared repository. As for system architecture, a client-server architecture is implemented whereby a server receives text messages from multiple clients, stores them to common repository and then distribute the messages from central repository to the intended recipient clients.

3 Design

3.1 Architectural Style

For developing this distributed messaging platform, following tools and technologies are used:

i. WampServer 3.2.3
ii. Laravel 8.0
iii. Laravel Echo Server 1.6.2
iv. Redis 3.2.100
v. MySQL 5.7.31
vi. PHP 7.3.21
vii. jQuery 3.2.1

In order to develop and test the ChatBox application using a single computer, WampServer is installed that allows web development and testing locally. Data-centered architectural style is followed by implementing a relational database, MySQL, to store all the necessary data received from multiple clients in a common repository. The application uses Socket.IO which enables real-time, bidirectional and event-based communication. When any message is inserted in database, an event is raised which implements “ShouldBroadcast” interface of Laravel framework. This event dispatches a job which will be executed and then broadcast by Queue Worker using Redis as a broadcast driver. For Socket.IO server, Laravel Echo server is used which is a Node.js server that can connect to Redis and establish web socket connections with clients. It takes the broadcast events from Redis and send it to the connected clients. Redis queue driver is also configured to allow multiple Laravel Redis jobs to be queued and processed. On the client side, socket.io-client (which will connect to Socket.IO server) and Laravel-echo (a helper JavaScript library that makes it easy to listen to the events) are used. In this way, the features of event-based architecture are incorporated with that of data-centric architecture.

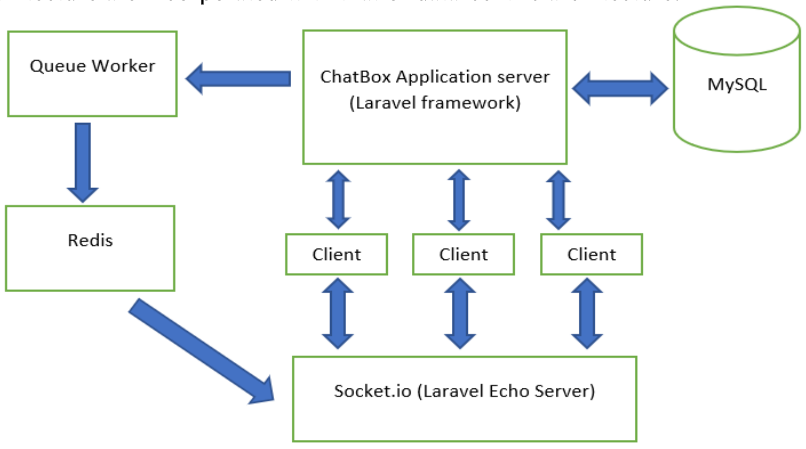


Fig. 1. Overall architecture

The system architecture of this distributed messaging platform is centralized system architecture. Client side of the application consists of a user-friendly messaging interface developed using HTML, CSS, AJAX and so on. When the user sends a message, the client process requests the application server to store the message in the database and then forward it to the destined clients using Redis and Laravel Echo Server. A single application server handles the requests from multiple clients by storing the messages and sending the messages to appropriate users when requested and hence it is centralized architecture. To get the application start working, the following steps need to be performed:
i. Start WampServer
ii. Start Redis Server
iii. Start Laravel Echo Server
iv. Start Redis

3.2 Features of ChatBox

The main features of ChatBox application are as follows:

- Persistent storage of messages
- One-to-one communication
- Group chat
- Message broadcasting
- User authentication
- Group creation and management

3.2.1 Persistent storage of messages

To maintain the persistent storage of messages, a relational database known as MySQL is used. Based on whether a user sends a message to all the users, a group of users or a specific user, the message details are stored in an appropriate table in the shared storage. The messages are also stored to provide users notification on the receipt of messages and show some latest messages from different users and groups. In addition to this, information required for user authentication and information about groups and its members are also stored in the database. The following diagram represents the relationship between tables (logical grouping of data) stored in the common repository.

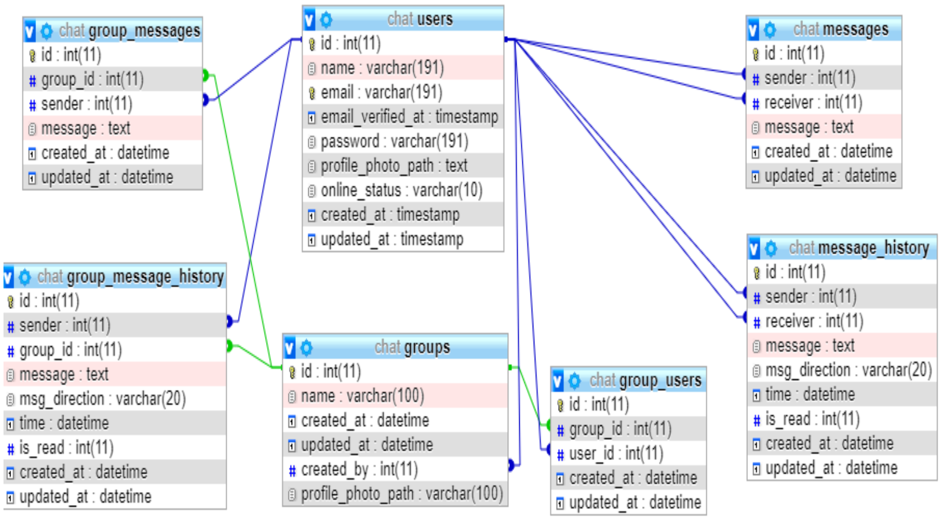


Fig. 2. Relationship between tables in database

3.2.2 Individual chat (One-to-one communication)

Every user when successfully logs in to the ChatBox application, is authenticated for a channel to indicate that the user is online and joins the channel. In addition, a communication channel is created for each user for messaging to individual users and another channel for messaging to a group of users. Every user can choose any other user to send an individual message.

In fig 2, User 1, User 2, User 3 and User 4 and User 5 are currently active users whereas User 6 is not online. User 2 types in a message M1 to send to User 3. This message consists of the actual text to be communicated, the intended recipient and the sender's information along with the time of message creation. The message is then sent to the application server which stores the message M1 to appropriate table ("messages") in MySQL database. After successful data publish in persistent data storage, an event is raised to broadcast the message in the private channel of the recipient. The server receives the data delivered from the database and broadcasts the message in the form of an event to the destination channel. Each event defines the private channel to broadcast on and the appropriate message details to broadcast with. This broadcast is facilitated by Redis server (by pairing Redis broadcaster with a Socket.IO server) and it also employs queues so that the events are broadcasted in First-In First-Out order. All the events broadcasting is done via queued jobs; therefore, the response time of the application is not seriously affected. The client-side uses Socket.IO client to connect to Socket.IO server and Laravel Echo to listen to the events broadcast. All the channels used in the application are private channels; hence, the users must be authorized to access them. Since User 3 is online and authenticated to listen to its channel, M1 gets delivered to the client-side of application and User 3 can receive the message in real time. Despite being online and having joined their respective communication channels, other active users do not receive a copy of M1 because no event is created to transmit the message to their channels. At the same time, several other users can communicate with each other individually and a message from a user is broadcasted to a specific user only. One-to-one communication between users occur independently of other users.

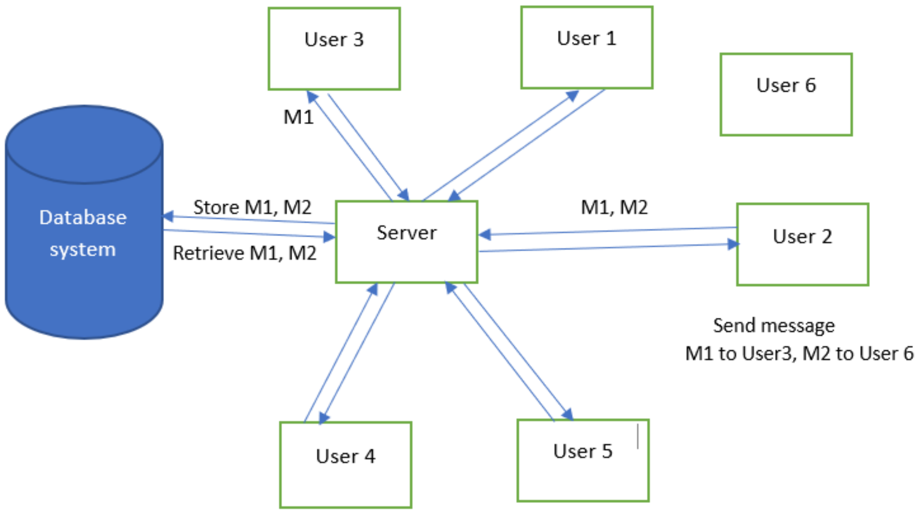


Fig. 3. One-to-one communication

In another scenario, User 2 sends message M2 to User 6. This message is stored similarly in the database. Then, a broadcast event attempts to deliver M2 to the intended recipient. Since User 6 is offline and has not joined the communication channel, the message M2 does not get delivered to User 6. In this case, the event is broadcasted but not delivered. However, the message M2 is published to database regardless of the status (online/offline) of the recipient. The data generated by users is guaranteed to be available in the common repository from where the consumers of the chat can retrieve the messages, but events may or may not delivered; therefore, the underlying software architecture is data-centered architecture. When User 6 logs in to application with valid credentials, s/he can view all the messages sent to her/him because the server fetches the messages sent to that user from the persistent storage and broadcasts the message to the private channel of User 6.

3.2.3 Group Chat (Many-to-many communication)

Group chat is a communication between a specific group of users usually referred to as members of the group. Each user has the privilege to create a group in which the members can be added and edited from the list of registered users. Let us consider following scenario for group messaging where User 1 sends message M1 to a group:

Members of a group = User 1, User 2, ..., User 6
Online users = User 1, User 2, User 3, User 4, User 5, User 8
Offline users = User 6, User

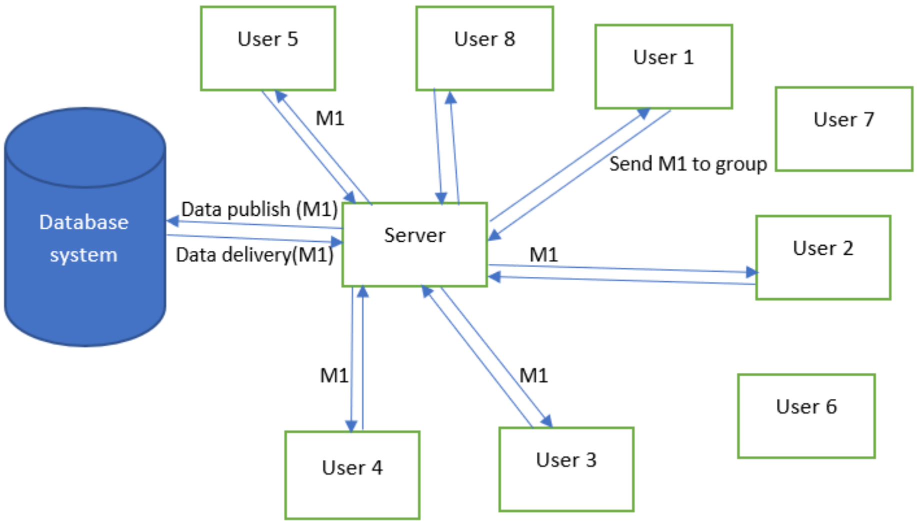


Fig. 4. Group chat among users of a group

Group chat is similar to one-to-one communication in the sense that a message sent to a group is transmitted to each member of the group. When User 1 creates a message M1 to send to the group, the client process for User 1 requests the application server to store the message in common central repository. The server publishes M1 to designated table ("group messages") in database with details such as sender of the message, the group to be delivered to and time of message sent. In one-to-one communication, the receiver is a specific user but for group message, it is a group. The server then creates an event for each member of the group specifying the communication channel to be broadcasted on and the details of the message (sender, chat, sent time etc.) to be broadcasted with. All these events are passed to message queue which is processed by Redis server one by one thus delivering the message in real time to all the active users. Since User 6 is not online at the time the producer of chat produced M1, the event cannot get delivered to User 6. However, M1 is retrieved from the database and delivered when User 6 comes online and requests the server for the message sent to it. At the same time, User 8 is also online and connected to a communication channel, but s/he cannot see M1. Because s/he is not a member of the group, the server does not create an event for broadcasting M1 to User 8 and the channel User 8 has joined is not authenticated to receive M1. In this way, the server dispatches a separate copy of the message to each private communication channels of the members that is created for receiving group messages. Like one-to-one communication, destination clients/users do not necessarily need to be available for the source client to originate message and the destination clients can independently retrieve the messages sent to them regardless of the availability of sender. Group chat is also known as multicasting which is a special form of broadcasting as the destination users form a subset of all the registered users.

3.2.4 Message Broadcast (All-to-all communication)

A user can send a message to all the registered users in the application. Message broadcast is a special form of group chat because all the users that have successfully registered themselves in the application are the members of a larger group ("All"). No members can leave that group. The underlying working mechanism is fundamentally similar to that of a group communication.

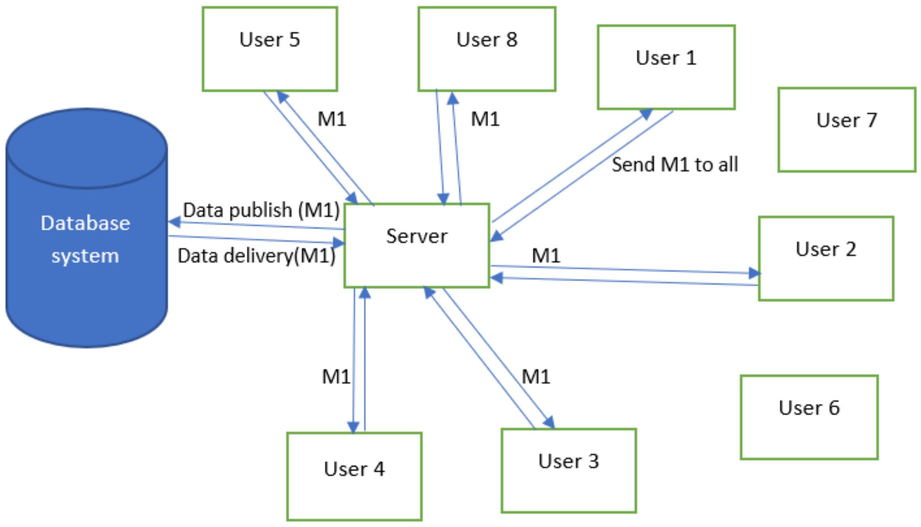


Fig. 5. Message broadcasting to all users

In this scenario, User 1 creates message M1 to send to all the registered users in the application. The client process request server process to publish M1 to common repository. Since the chat is intended to be delivered to every user, the details stored in database ("group messages" table) include the text message, the sender, the message creation time and the receiver information is set default to "All" group. On successful saving of the M1, server then creates an event for each user designating their private communication channel to broadcast on and the message and other details to broadcast with. All these events are queued in message queue by the Redis server and then processed in FIFO manner. On successful processing of the queued events, the message is delivered to every active user. Every other user in the above diagram other than User 6 and User 7 consumes a copy of M1 which is retrieved from database. When users 6 and 7 come online, they are authenticated to join the private channels for individual and group communication. When the client process requests M1, the server retrieves it from persistent storage and sends a copy to the client. All the users can still access the messages sent to them by logging in the application again whenever they need to and by requesting the server.

3.2.5 User authentication

In order to access the features of ChatBox application, every user needs to complete user registration process. The valid registration details entered by the user are sent from the client side of the application to the server side. The server then stores the user information in database. The passwords are stored only after hashing (bcrypt hashing algorithm) for security reasons. When a user tries to login to the system, the login credentials entered in the login window are sent to the server which then retrieves the information provided by user during registration process from the common repository and compares them. Only after the verification of entered login details, the user can get access to the features provided by the distributed messaging platform. The application is protected against unauthorized access due to screening of users by login (authentication) process. The messages, on the other hand, are protected against unauthorized access using application authentication middleware classes.

3.2.6 Group creation and management

A group is a collection of users formed by a user so that the members belonging to that collection can all communicate together at the same time. A creator of a group chooses members from a list of registered users to form a group. Only the creator of the group has the privilege to add or remove users from the group whereas the members of the group can choose to leave the group. However, all the registered users are by default a member of "All" group used for broadcasting messages to all users and no user can leave that group.

3.3 Design Experiment

My experiment was to send text messages to and from users. For this, I had registered some users, created some groups and sent some messages. I then used three different web browsers to login three valid users in the application. User access to messaging facilities was denied when a user entered invalid username or password. From the list of users, I clicked on an individual user and all the messages in previous conversations were loaded. Moreover, when I sent a text message to the selected one, only that user received the message in real time. Similarly, when I sent a message to a group, all the online group members received the messages and I could also see the messages sent to me. I also added some users to the group and the newly added users could also see the messages previously sent. In addition, I sent a message to all the users and received messages from all other users too. The testing of the application demonstrated that all messaging and authentication functionalities worked as expected. This real time receipt and delivery of messages indicated the success of this application.

4 Analysis

In order to test the application, I did the experiment multiple times by creating several users/groups and then sending and receiving messages to and from a single user, a group of users and all the users. In each iteration of the experiment, the application worked successfully producing the results as intended. This demonstrates that the application works properly.

5 Conclusion

Chat is an everyday requirement for people nowadays to communicate with each other for personal, teaching-learning or work-related activities. For this purpose, a distributed messaging platform, ChatBox, is developed where the independent users of the distributed system can send/receive messages to/from a specific user, a group of users or all the users in the system. The analysis section shows that the application functions successfully. The main objective of this project was to demonstrate a mastery in at least one of the concepts of distributed system. This project clearly exhibits how more than one independent computer/system in the distributed system can share the same storage resource (database) without having to maintain a dedicated data store of their own. In addition, the concept of data-centered architecture is well implemented by using a relational database to act as a common repository for storing text messages generated by multiple clients in the system. Based on what I promised to deliver, I have exceeded the goal because some extra features are added for user authentication, group creation and management. Working on this project has really been a significant accomplishment because I got an opportunity to implement some important concepts learned in the class and demonstrate that I have mastered the concepts of message-oriented communication and data-centric architecture of distributed system.

6 Future Works

There are some features that can be added to this messaging platform to upgrade its services. Currently, users can only send text messages to each other. However, in real life, we may need to send messages in the form of links, documents, photos and many more. Therefore, adding that feature to the application would be a significant upgrade. In addition, there are some minor improvements that can be done in the user interface. Had the time constraints permitted, I would have included some of these improvements such as showing pictures of some users combined in the group picture, listing newly registered user in users list in real time, implementing message "seen" status, allowing to tag people, showing that somebody is "typing" and so on.

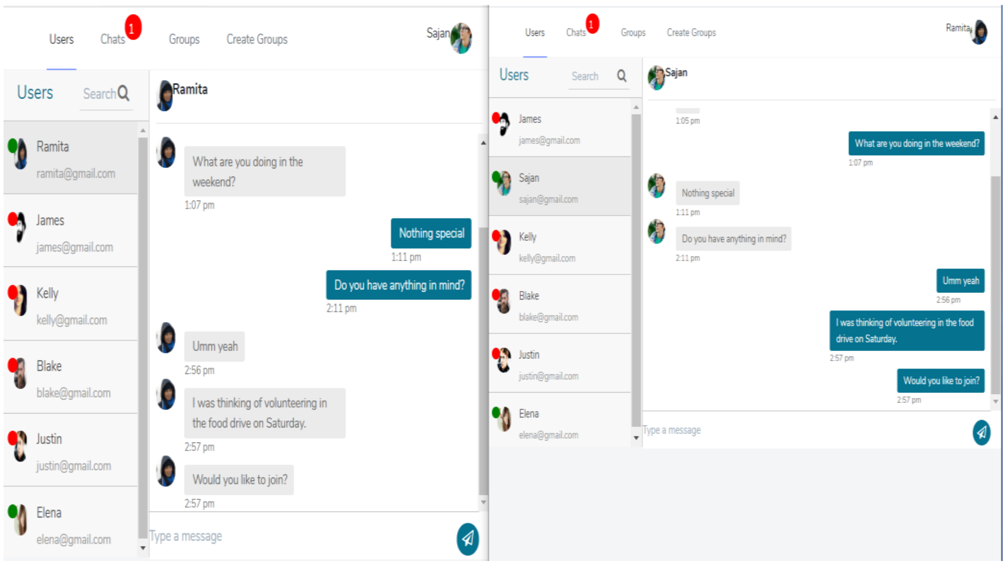
References

1. Abba I., Cruz Mia., Eaganathan U., Gabriel J (2013). Development of LAN Chat Messenger (LCM) using Rational Unified Process (RUP) Methodology with Object Oriented Programming (2013)
2. Network socket, https://en.wikipedia.org/wiki/Network_socket
3. https://www.whoson.com/our-two-cents/the-history-of-live-chat-software/
4. El-Seoud, Samir & Taj-Eddin, Islam. (2016). Developing an Android Mobile Bluetooth Chat Messenger as an Interactive and Collaborative Learning Aid. 10.1007/978-3-319-50340-0_1.

5. techopedia.com/definition/5392/session-computer-science

Appendices

The following are some snapshots that show the exchange of messages between users during one to one communication, group chat and message broadcasting to all users of the chat application:



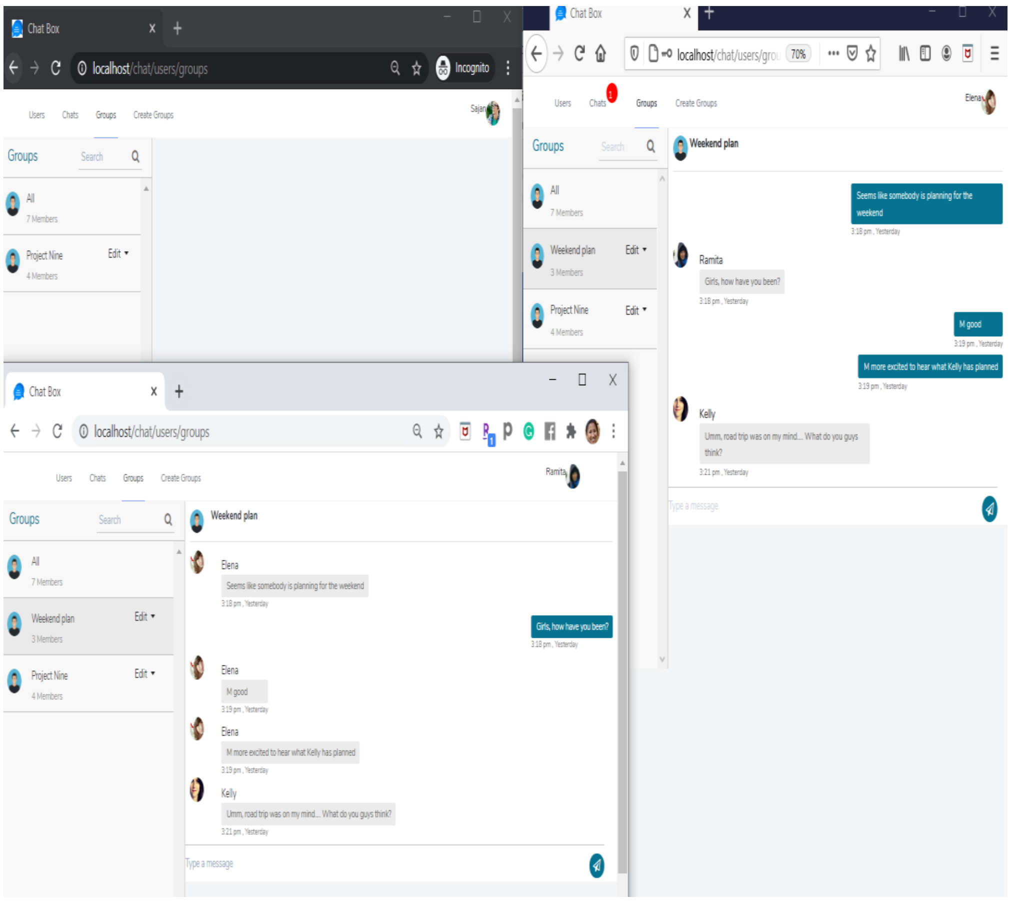
Ramita
ramita@gmail.com
James
james@gmail.com
Kelly
kelly@gmail.com
Blake
blake@gmail.com
Justin
justin@gmail.com
Elena
elena@gmail.com

What are you doing in the weekend?
1:07 pm
Nothing special
1:11 pm
Do you have anything in mind?
2:11 pm
Umm yeah
2:56 pm
I was thinking of volunteering in the food drive on Saturday.
2:57 pm
Would you like to join?
2:57 pm

James
james@gmail.com
Sajan
sajan@gmail.com
Kelly
kelly@gmail.com
Blake
blake@gmail.com
Justin
justin@gmail.com
Elena
elena@gmail.com

1:05 pm
What are you doing in the weekend?
1:07 pm
Nothing special
1:11 pm
Do you have anything in mind?
2:11 pm
Umm yeah
2:56 pm
I was thinking of volunteering in the food drive on Saturday.
2:57 pm
Would you like to join?
2:57 pm

Fig. 1: Individual chat



Weekend plan

Elena
Seems like somebody is planning for the weekend
3:18 pm , Yesterday

Girls, how have you been?
3:18 pm , Yesterday

Elena
M good
3:19 pm , Yesterday

Elena
M more excited to hear what Kelly has planned
3:19 pm , Yesterday

Kelly
Umm, road trip was on my mind... What do you guys think?
3:21 pm , Yesterday

Weekend plan

Seems like somebody is planning for the weekend
3:18 pm , Yesterday

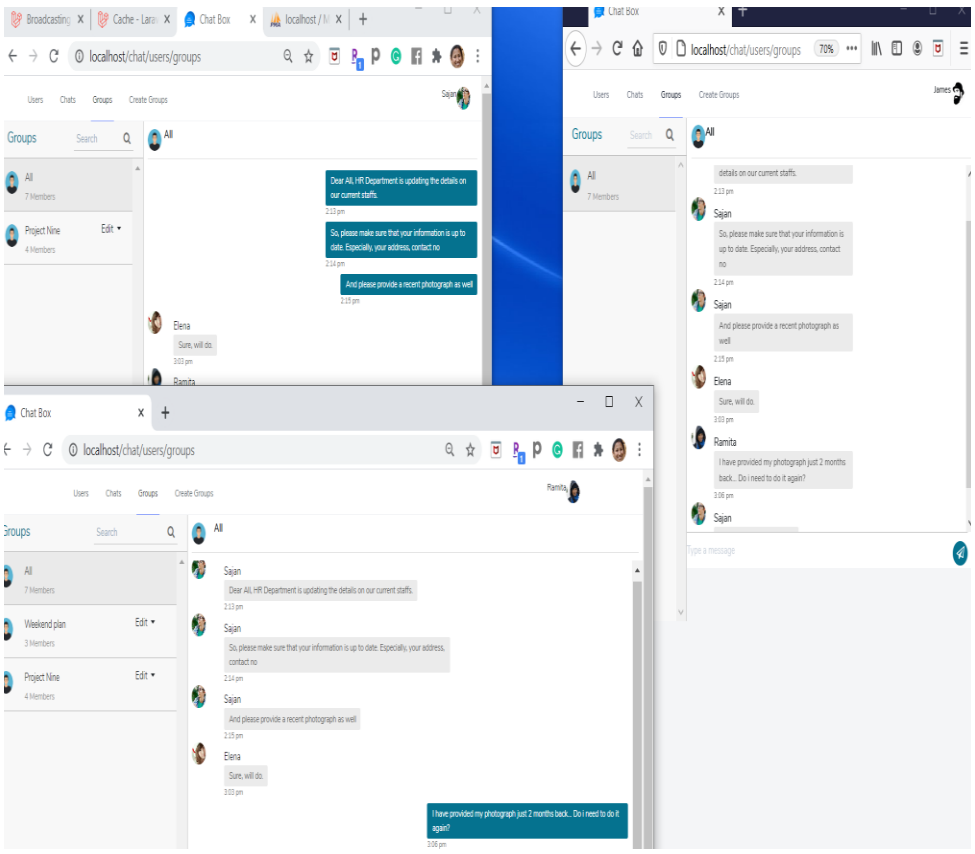
Ramita
Girls, how have you been?
3:18 pm , Yesterday

M good
3:19 pm , Yesterday

M more excited to hear what Kelly has planned
3:19 pm , Yesterday

Kelly
Umm, road trip was on my mind... What do you guys think?
3:21 pm , Yesterday

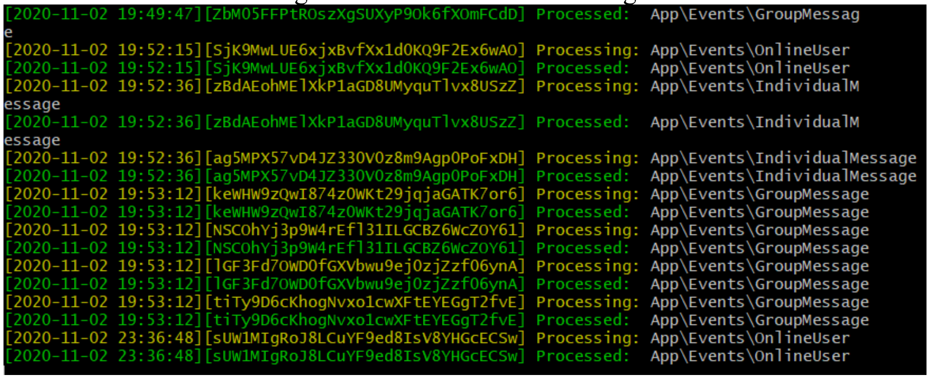
Fig. 2: Group chat



Dear All, HR Department is updating the details on our current staffs.
2:13 pm
So, please make sure that your information is up to date. Especially, your address, contact no
2:14 pm
And please provide a recent photograph as well
2:15 pm
Elena
Sure, will do.
3:03 pm
Ramita
I have provided my photograph just 2 months back.. Do i need to do it again?
3:06 pm

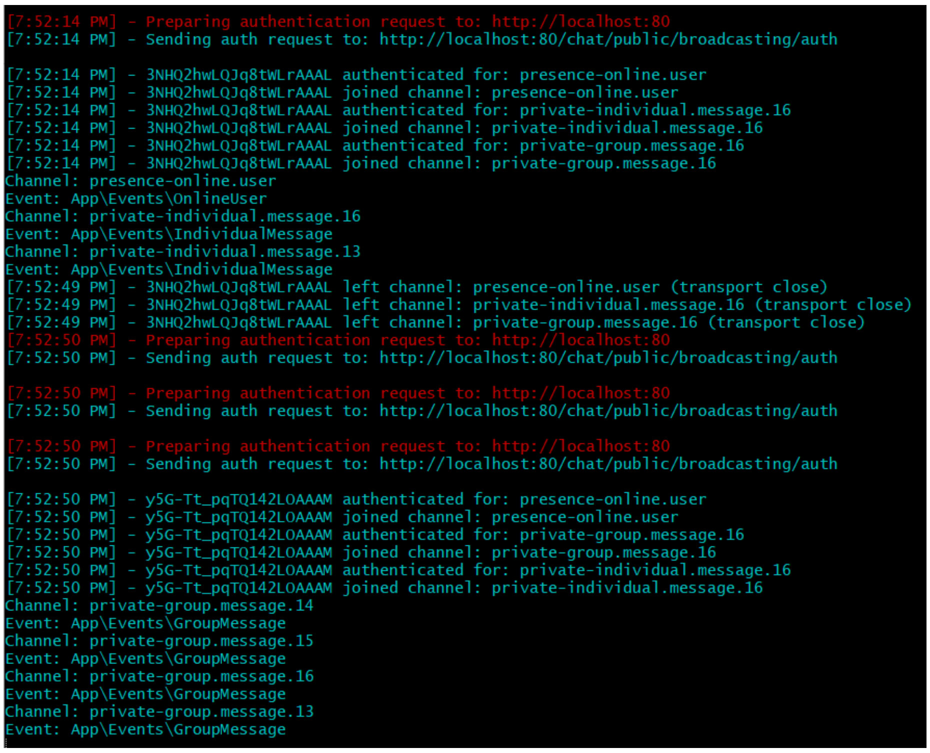
Fig. 3: All-to-all chat

Meanwhile, the background processing that is carried out in the server side to store, fetch and deliver the messages as demonstrated in the images below:



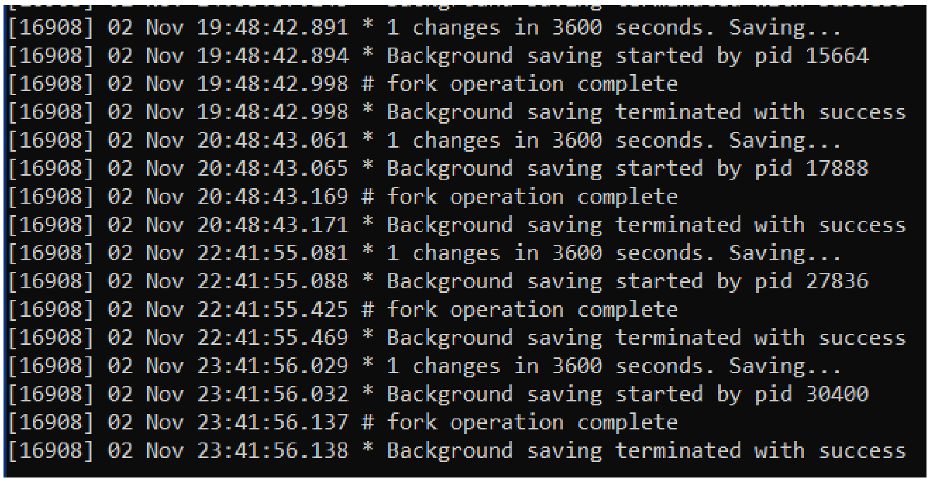
```
[2020-11-02 19:49:47][ZbM05FFPtROszXgSUXyP9Ok6fXOmFCdD] Processed:  App\Events\GroupMessag
e
[2020-11-02 19:52:15][SjK9MwLUE6xjxBvfXx1dOKQ9F2Ex6wAO] Processing: App\Events\OnlineUser
[2020-11-02 19:52:15][SjK9MwLUE6xjxBvfXx1dOKQ9F2Ex6wAO] Processed:  App\Events\OnlineUser
[2020-11-02 19:52:36][zBdAEohMElXkP1aGD8UMyquTlvx8USzZ] Processing: App\Events\IndividualM
essage
[2020-11-02 19:52:36][zBdAEohMElXkP1aGD8UMyquTlvx8USzZ] Processed:  App\Events\IndividualM
essage
[2020-11-02 19:52:36][ag5MPX57vD4JZ330V0z8m9Agp0PoFxDH] Processing: App\Events\IndividualMessage
[2020-11-02 19:52:36][ag5MPX57vD4JZ330V0z8m9Agp0PoFxDH] Processed:  App\Events\IndividualMessage
[2020-11-02 19:53:12][keWHW9zQwI874zOWKt29jqjaGATK7or6] Processing: App\Events\GroupMessage
[2020-11-02 19:53:12][keWHW9zQwI874zOWKt29jqjaGATK7or6] Processed:  App\Events\GroupMessage
[2020-11-02 19:53:12][NSCOhYj3p9W4rEfl31ILGCBZ6WcZOY61] Processing: App\Events\GroupMessage
[2020-11-02 19:53:12][NSCOhYj3p9W4rEfl31ILGCBZ6WcZOY61] Processed:  App\Events\GroupMessage
[2020-11-02 19:53:12][lGF3Fd7OWD0fGXVbwu9ej0zjZzf06ynA] Processing: App\Events\GroupMessage
[2020-11-02 19:53:12][lGF3Fd7OWD0fGXVbwu9ej0zjZzf06ynA] Processed:  App\Events\GroupMessage
[2020-11-02 19:53:12][tiTy9D6cKhogNvxo1cwXFtEYEGgT2fvE] Processing: App\Events\GroupMessage
[2020-11-02 19:53:12][tiTy9D6cKhogNvxo1cwXFtEYEGgT2fvE] Processed:  App\Events\GroupMessage
[2020-11-02 23:36:48][sUW1MIgRoJ8LCuYF9ed8IsV8YHGcECSw] Processing: App\Events\OnlineUser
[2020-11-02 23:36:48][sUW1MIgRoJ8LCuYF9ed8IsV8YHGcECSw] Processed:  App\Events\OnlineUser
```

Fig. 4: Redis queue processing events



```
[7:52:14 PM] - Preparing authentication request to: http://localhost:80
[7:52:14 PM] - Sending auth request to: http://localhost:80/chat/public/broadcasting/auth

[7:52:14 PM] - 3NHQ2hwLQJq8tWLrAAAL authenticated for: presence-online.user
[7:52:14 PM] - 3NHQ2hwLQJq8tWLrAAAL joined channel: presence-online.user
[7:52:14 PM] - 3NHQ2hwLQJq8tWLrAAAL authenticated for: private-individual.message.16
[7:52:14 PM] - 3NHQ2hwLQJq8tWLrAAAL joined channel: private-individual.message.16
[7:52:14 PM] - 3NHQ2hwLQJq8tWLrAAAL authenticated for: private-group.message.16
[7:52:14 PM] - 3NHQ2hwLQJq8tWLrAAAL joined channel: private-group.message.16
Channel: presence-online.user
Event: App\Events\OnlineUser
Channel: private-individual.message.16
Event: App\Events\IndividualMessage
Channel: private-individual.message.13
Event: App\Events\IndividualMessage
[7:52:49 PM] - 3NHQ2hwLQJq8tWLrAAAL left channel: presence-online.user (transport close)
[7:52:49 PM] - 3NHQ2hwLQJq8tWLrAAAL left channel: private-individual.message.16 (transport close)
[7:52:49 PM] - 3NHQ2hwLQJq8tWLrAAAL left channel: private-group.message.16 (transport close)
[7:52:50 PM] - Preparing authentication request to: http://localhost:80
[7:52:50 PM] - Sending auth request to: http://localhost:80/chat/public/broadcasting/auth

[7:52:50 PM] - Preparing authentication request to: http://localhost:80
[7:52:50 PM] - Sending auth request to: http://localhost:80/chat/public/broadcasting/auth

[7:52:50 PM] - Preparing authentication request to: http://localhost:80
[7:52:50 PM] - Sending auth request to: http://localhost:80/chat/public/broadcasting/auth

[7:52:50 PM] - y5G-Tt_pqTQ142LOAAAM authenticated for: presence-online.user
[7:52:50 PM] - y5G-Tt_pqTQ142LOAAAM joined channel: presence-online.user
[7:52:50 PM] - y5G-Tt_pqTQ142LOAAAM authenticated for: private-group.message.16
[7:52:50 PM] - y5G-Tt_pqTQ142LOAAAM joined channel: private-group.message.16
[7:52:50 PM] - y5G-Tt_pqTQ142LOAAAM authenticated for: private-individual.message.16
[7:52:50 PM] - y5G-Tt_pqTQ142LOAAAM joined channel: private-individual.message.16
Channel: private-group.message.14
Event: App\Events\GroupMessage
Channel: private-group.message.15
Event: App\Events\GroupMessage
Channel: private-group.message.16
Event: App\Events\GroupMessage
Channel: private-group.message.13
Event: App\Events\GroupMessage
```

Fig. 5: Broadcast events and channels in Laravel Echo Server



```
[16908] 02 Nov 19:48:42.891 * 1 changes in 3600 seconds. Saving...
[16908] 02 Nov 19:48:42.894 * Background saving started by pid 15664
[16908] 02 Nov 19:48:42.998 # fork operation complete
[16908] 02 Nov 19:48:42.998 * Background saving terminated with success
[16908] 02 Nov 20:48:43.061 * 1 changes in 3600 seconds. Saving...
[16908] 02 Nov 20:48:43.065 * Background saving started by pid 17888
[16908] 02 Nov 20:48:43.169 # fork operation complete
[16908] 02 Nov 20:48:43.171 * Background saving terminated with success
[16908] 02 Nov 22:41:55.081 * 1 changes in 3600 seconds. Saving...
[16908] 02 Nov 22:41:55.088 * Background saving started by pid 27836
[16908] 02 Nov 22:41:55.425 # fork operation complete
[16908] 02 Nov 22:41:55.469 * Background saving terminated with success
[16908] 02 Nov 23:41:56.029 * 1 changes in 3600 seconds. Saving...
[16908] 02 Nov 23:41:56.032 * Background saving started by pid 30400
[16908] 02 Nov 23:41:56.137 # fork operation complete
[16908] 02 Nov 23:41:56.138 * Background saving terminated with success
```

Fig. 6: Events log in Redis server