# Chat App

Just what you need!

Szabo Arnold | Szép Zoltán | Vecsei Szilveszter
SAPIENTIA HUNGARIAN UNIVERSITY OF TRANSYLVANIA | Faculty Of Technical And Human Sciences, Târgu Mureș

12/19/2016

## Table of Contents

# Chat App Project

## Introduction

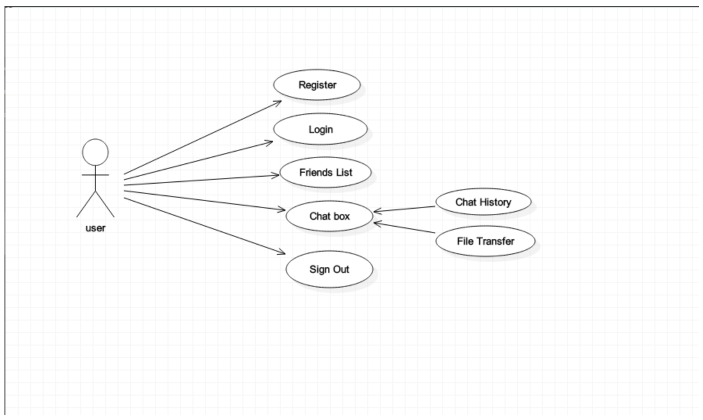
### *Instant messaging (IM)*

Instant messaging (IM) is a type of online chat that offers real-time text transmission over the Internet. A LAN messenger operates in a similar way over a local area network. Short messages are typically transmitted bi-directionally between two parties, when each user chooses to complete a thought and select "send". Some IM applications can use push technology to provide real-time text, which transmits messages character by character, as they are composed. More advanced instant messaging can add file transfer, clickable hyperlinks, Voice over IP, or video chat.

### *Why is it important ?*

Chat apps are especially appealing to publishers because they allow these brands to tap into users' "dark social" activity. Dark social traffic stems from people sharing content privately through IM programs, messaging apps, and email, among other means.

Because chat apps were once primarily used for peer-to-peer communications, publishers have an opportunity to reach audiences on these platforms through a more conversational exchange.



## Proposed Solution

Our project may offer solution for simple instant messaging with file transfer for free on multiple platforms. You get a simple instant messaging app with the mos useful feature and a beautifoul user interface.

### *Platforms*

- Web
- Android

### *Features*

- Real time messaging
- Real time file transfer
- Private message
- Group message (post)
- Login functionality
- Offline messages
- Offline file transfer
- Message history

# Implementation

## *Chat App – Server*

The Chat App is running on a nodejs server (currently on localhost), which is using the express web framework, socket.io framework, mongodb database with mongoose package.

The first goal was to setup a simple HTML webpage that serves out a form and a list of messages, for this was used the Node.JS web framework express.

Sockets have traditionally been the solution around which most realtime chat systems are architected, providing a bi-directional communication channel between a client and a server. This means that the server can push messages to clients. Whenever you write a chat message, the idea is that the server will get it and push it to all other connected clients.

Socket.IO is composed of two parts:

- A server that integrates with (or mounts on) the Node.JS HTTP Server: socket.io
- A client library that loads on the browser side: socket.io-client

During development, socket.io serves the client automatically for us, so for we only had to install one module: socket.io.

The main idea behind Socket.IO is that you can send and receive any events you want, with any data you want. Any objects that can be encoded as JSON will do, and binary data is supported too.

In order to send an event to everyone, Socket.IO gives us the `io.emit`:

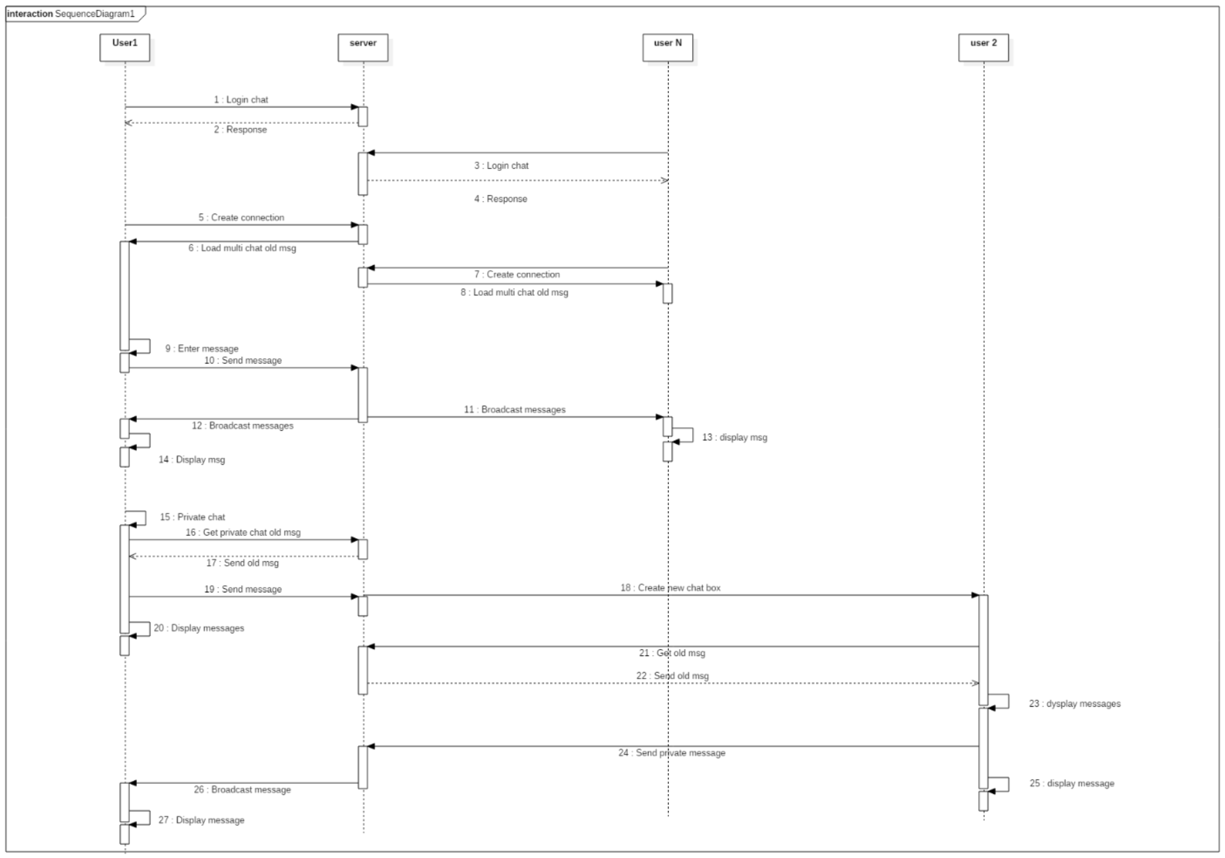
```
io.emit('some event', { for: 'everyone' });
```

If you want to send a message to everyone except for a certain socket, we have the `broadcast` flag:

```
io.on('connection', function(socket){
  socket.broadcast.emit('hi');
});
```

In this case, for the sake of simplicity we'll send the message to everyone, including the sender.

```
io.on('connection', function(socket){
  socket.on('chat message', function(msg){
    io.emit('chat message', msg);
  });
});
```



## *Chat App – Web*

The web client has the following features:

- Sending a message to all users joining to the room.
- Sending private messages.
- Notifies when each user joins or leaves.
- Notifies when an user start typing a message.
- Login

Web client side the first goal was to setup a simple HTML webpage that serves out a form and a list of messages. For this we used the Node.JS web framework express.

When the user types in a message, the server gets it as a chat message event. Client side when we capture a chat message event we'll include it in the page.

## *Chat App – Android*

The app has the following features:

- Sending a message to all users joining to the room.
- Sending private messages.
- Notifies when each user joins or leaves.
- Notifies when an user start typing a message.
- Login

Socket.IO provides an event-oriented API that works across all networks, devices and browsers. It's incredibly robust (works even behind corporate proxies!) and highly performant, which is very suitable for multiplayer games or realtime communication.

First, we have to initialize a new instance of Socket.IO, IO.socket() returns a socket for http://chat.socket.io with the default options. Notice that the method caches the result, so you can always get a same Socket instance for an url from any Activity or Fragment. And we explicitly call connect() to establish the connection here (unlike the JavaScript client). In this app, we use onCreate lifecycle callback for that, but it actually depends on your application.

Sending data looks as follows. In this case, we send a string but you can do JSON data too with the org.json package, and even binary data is supported as well!

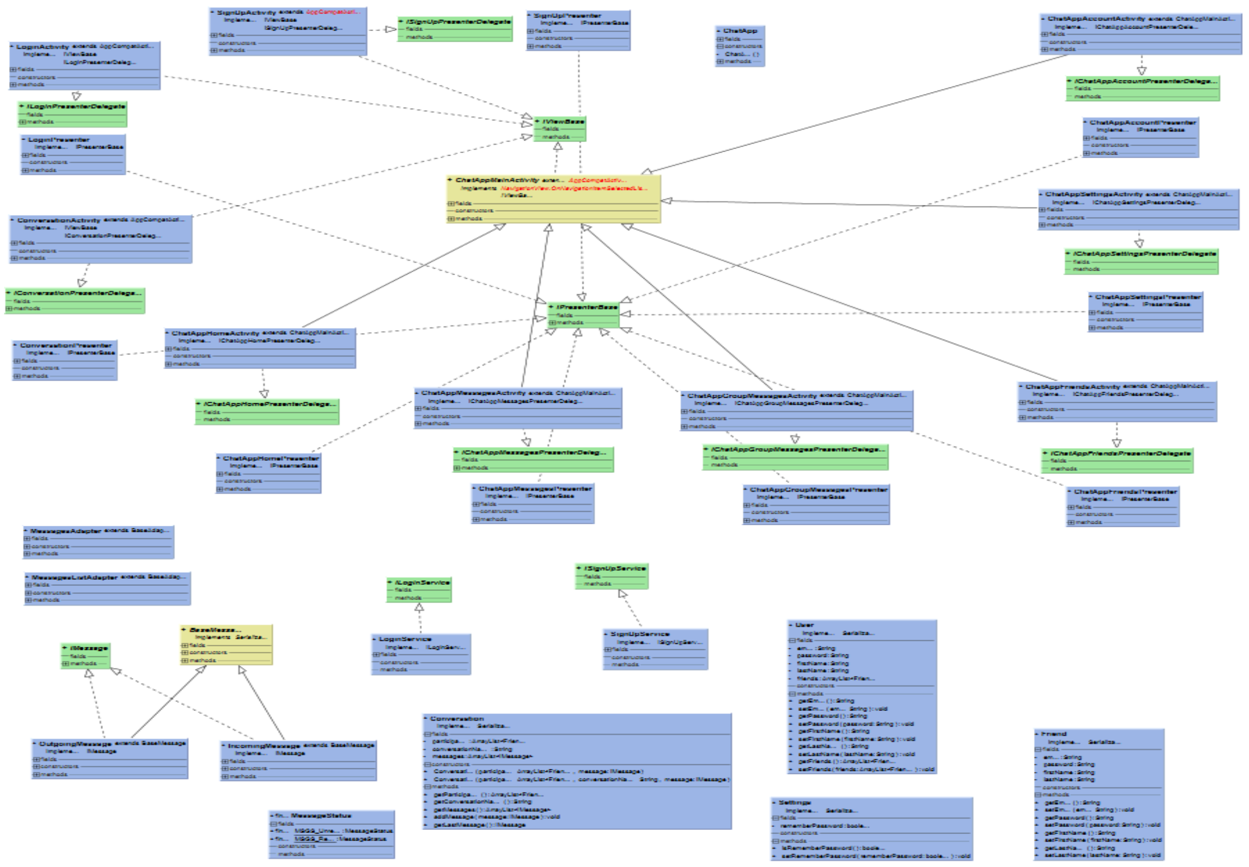
```
private EditText mInputMessageView;

private void attemptSend() {
    String message = mInputMessageView.getText().toString().trim();
    if (TextUtils.isEmpty(message)) {
        return;
    }

    mInputMessageView.setText("");
    mSocket.emit("new message", message);
}
```

Socket.IO is **bidirectional**, which means we can send events to the server, but also at any time during the communication the server can send events to us. We then can make the socket listen an event on onCreate lifecycle callback. A listener is an instance of Emitter.Listener and must be implemented the call method. You'll notice that inside of call() is wrapped by Activity#runOnUiThread(), that is because the callback is always called on another thread from Android UI thread, thus we have to make sure that adding a message to view happens on the UI thread.

Since an Android Activity has its own lifecycle, we should carefully manage the state of the socket also to avoid problems like memory leaks. In this app, we'll close the socket connection and remove all listeners on onDestroy callback of Activity.



## Conclusion

Want your messages to be read quickly? Want to make sure your messages are actually read? You’ll probably want to avoid sending emails or leaving voice mails. Did you know that 97% of text messages are read within three minutes of delivery? Versus 22% of emails? Text messaging is a highly effective and faster way to read messages. It’s become an essential social & business communication tool.

Chat App can help you, an instant messaging app solution across Android & Web.

## Bibliography

https://nodejs.org/en/

http://socket.io/

https://www.mongodb.com/

https://developer.android.com/index.html

http://hannesdorfmann.com/mosby/mvp/

https://code.tutsplus.com/tutorials/how-to-adopt-model-view-presenter-on-android--cms-26206