Draft for master thesis

Henrique Rocha

1 Backend

1.1 XMPP Approach

One of the advantages that XMPP based applications offer is the possibility to send messages across different servers, an XMPP user is identified by its *Jabber ID* which is definined by the pair username and server name in the form *username@server*.

The user profile is represented in terms of Data Forms (XEP-0004) protocol and managed (registration and update) through *Info/Query* (IQ) [rfc3920/IQ Semantics] requests to the XMPP server which responds with an IQ response containing information to retrieve the registration fields needed to fill its profile (eg username, password, telephone).

An important aspect of this model is the ability to define a user as a set of attributes that can change dinamically in order to fit the IM applications needs.

The connection to the XMPP server could be done through the same web server as the user is connected, but this web server would handle a lot of connections and that would have a great impact on the overall system performance. Our proposed solution distributes the XMPP connections among users relieving this extra work to the web browsers.

The first working prototype of our solution implemented the user registration and authentication components through *Strophe.js*. The next step on implementation would be the access to the user profile and consequentally all the other features needed for our solution. Typically a web application consists in multiple web pages which are navigated by the users. Each time a webpage is accessed, either from a new context or a transition from a previous page, its context is cleared except for local storage and cookies, the javascript context is cleared, including the XMPP connections performed by strophe.js, an automatic mechanism is required for avoiding the user implicit reauthentication.

One solution for reauthentication can be achieved by storing the user's JabberID and password on local storage and everytime a page is accessed the authentication is performed without the users knlowledge. Clearly this solution represents security flaws, the local storage can be easily accessed (HOW? explain!) locally or by performing a cross site scripting attack and reveal all the local storage, the same problem arises if the credentials are stored on cookies.

An alternative solution is known as Session Attachment (see https://metajack.im/2008/10/03/getting-attached-to-strophe/), which requires that a *session identifier*(SID) together with a *initial request identifier (RID)* is passed to strophe.js in order to re-connect to the same stream on XMPP server. Either SID and RID are unpredictable and, particularly, *RID* changes on every request which is worthless if a user mantains more than one tab opened, for example for multiple conversations at the same time.

Another alternative would be the development of our solution in a single web page, which would increse drastically the complexity of our web application.

We know that an *XMPP* based application could simplify our work just by using functionalities that XMPP already implemment, but from all the *XMPP* features, we don't need all of them, in fact we just need a subset of XMPP features, namely a simple way to register users, access and edit user's information, create chat rooms, send messages and access to presence information.

1.2 Our approach

In order to implement a solution without the *XMPP*'s limitations that we presented on the previous section, we decided to implement our own backend.

One important aspect that we should take into account is the scalability of our solution. All the information that we need to store relative to the users and chat rooms should be stored on a scalable database.

As our database grows, we may need to distribue the load among multiple servers, which can be achieved on SQL servers by spliting tables among servers, but this could be a challenge operation when doing relations between tables. For this reason we opted for using a NoSQL database. Many of NoSQL databases are built having the scalability functionality from start.

One option for a NoSQL dabase is MongoDB, there are others but this one seems to have popularity, it is free and well documented. There are no strong reasons for excluding options like Apache Cassandra or others.

The authentication problem that XMPP raised when implementing a client on a web browser is easily solved by using cookies to identify authenticated users. A cookie is mantained as long as is specified by its expiration date and the user only needs to reauthenticate when it logs out or the cookie expires, changing from page to page does not raises any reauthentication problem.

On our solution the actions that are performed by clients are directly sent to web servers that control the validity of the actions and performs the respective changes to the database. This logic on the XMPP approach would be distributed among XMPP servers and not the webservers, so now our webservers will increase their complexity by implementing the features we needed from XMPP.