Whisper Project
- Final Report -

Fernando Villa Khalid Alobaid Mariya Abdiyeva
Nicolas Cabuli Oluremi Obolo Santiago Arrubla

30 March 2017

1 Introduction

Whisper is a distributed chat system with one-to-one secure connection between three different clients (Android, iOS and Web) that use Firebase as a backend with a JSON database, which is specifically designed to transmit rapid messages.

Our main challenge was to create a final architecture that works properly, while dealing with a group project organization of multiple professionals from different backgrounds than software engineering. For us, this also involves multiple challenges, since learning Git-Hub and its methodology to collaborate as a group, learning coding across different languages, and follow a decision-making and conflict resolution methodology.

2 Review

Chat system applications is a very well famous problem in software engineering. Popular examples, just to mention ones, are chats like Whatsapp, Slack, Telegram, Skype, Snapchat, Facebook Messenger, among many others. About the platforms, Stack Overflow Developer Survey of 2017 [rtrtryrtyhl] establishes that the most popular technologies are Android, iOS and Web. In addition, the market share of iOS and Android in 2016 is about 12.9 and 86,2 percent, respectively [rury346]. We also decided to go for the Web Client because web development has been always attractive for us as we believe thats the way the old big companies like Google, Amazon and Facebook started.

One of the most important reviews we research at the beginning, was proper methodologies to work as a group. Specifically, the agile methodology, proposed by Alistain Cockburn and Jim Highsmith [47474yr] was our first approach. However, this implies also to have a proper division of each single task of our software. Because of this, we struggled at some point, and we must return to the foundations of this methodology or the classical one, where testing is conducted at the end and no in each single division. In other words, our main challenge here, was the integration test.

We picked Firebase because we were thinking of focusing a lot on the User Experience inside the clients. We wanted users to have a good experience using our app and make them want to use it. Since the beginning, we knew that the Authentication is a strong feature in Firebase. It lets you control who can read your data and let you pick different services such as Facebook, Gmail, GitHub, and others to register and login. One of the main things we found that Firebase has is the real-time database. Thanks to this we can see all the messages in real time in all the clients. Some of the backend development gets passed to the client to have faster reads. However, we decided that we preferred this model. Another of the advantages is that Firebase is backed by Google and that has a lot of documentation on their website. Finally, it takes minimum setup to begin developing, which was a key point for us as we wanted to finish the project on time. Even with this concern in mind we had some trouble finishing it.

As part of the creation of this project, we ultimately decided that in order to get the most out of this project, we wanted to learn more about different tools. We needed to pick a challenging technology with the robustness needed but also with vests amounts of information on the internet and in books to try and

reduce the learning curve the most we could. As we mentioned before, Firebase was the best choice for our intended purposes, not only is a robust platform backed up by Google. Aside all the benefits such as Web Analytics, Push Notifications, file storage, etc., Firebase gives you the flexibility to do a lot of backend code in the client side. Even though Firebase leverages a lot to the client, we realize it was interesting to test new ways of developing applications, as we only knew the basics around Object Oriented Programming.

3 Requirements, Architecture and Design

3.1 Requirements

Figure 1 presents our requirements by priority level. Our challenge was to develop as many requirements as we could. We are happy to said we manage to develop all the priority 1 level requirements, however, also we must say that we would have wanted to cover more of them.

Table 1: Requirements by priority level

Priority 1	**Priority 2**	**Priority 3**
-Secure connection -Encrypted messages -IOS, Android and Web platforms (at least two platforms) -Contact list -Search for contacts -Sign in by email or Gmail -One to one conversation (text message)	-Send images -Group chat -Chat bubbles instead of username -Notifications of received messages	-Sign in by Facebook -Availability to send other type of media (e.g. video, voice notes, etc.) -Online status of other users -Acknowledgment of received, delivered and read messages, including date time -Ability to use the chat offline

3.2 Architecture

According with the Architecture definition of Eden and Kazman [rtrtr], the next figure presents the general design. Thus, we implemented three clients, using Firebase as a backend, and a JSON data base designed to transmit faster. This involves a tradeoff, where we prioritize velocity over storage.

3.3 Data Base and Backend

As we have mentioned, we use Firebase as a backend. This strategy implies to simplify the logic in the backend and translate more work to the clients. However, the connection velocity is faster than a common design, due Firebase provide optimized sockets and a secure protocol with HTPPS.

Firebase consist of a real-time database with an API that allows to synchronize and store data in the cloud with multiple clients. It provides a client library that in our case allow the integration with Android, iOS, JavaScript, Java, and Swift. Other services of Firebase as a platform is the storage, hosting (which support static files such as CSS, HTML, and JavaScript). To sum-up, firebase plays a role as a real-time backend, suitable for scalable applications.

Other interesting characteristics of Firebase, are the authentication service, which supports social login providers, such as Facebook, GitHub, Google, etc. In addition, the crash reporting, as well the analytics tools, caught our attention for two main reasons: i) the possibility to extract information in the future to make more friendly the application (e.g. with performance measures like the average time in the application), and ii) the possibility to test improvements and track changes, with for example A/B tests.

JSON (JavaScript Object Notation) is a data interexchange format, easier to generate for machines. It is based on i) a set of name and value duple (i.e. an object), and ii) an ordered list of values (i.e. an array) [1sejkfhs]. JSON is more lightweight than XML and thus is preferable for web applications. Also [2sdjkhds] include a comparison between the advantages of JSON vs XML.

The following Figure QQQ presents the structure of the designed database. The idea behind this structure is to prioritize the processing velocity, so messages can be transmitted faster (thus, we duplicate in some

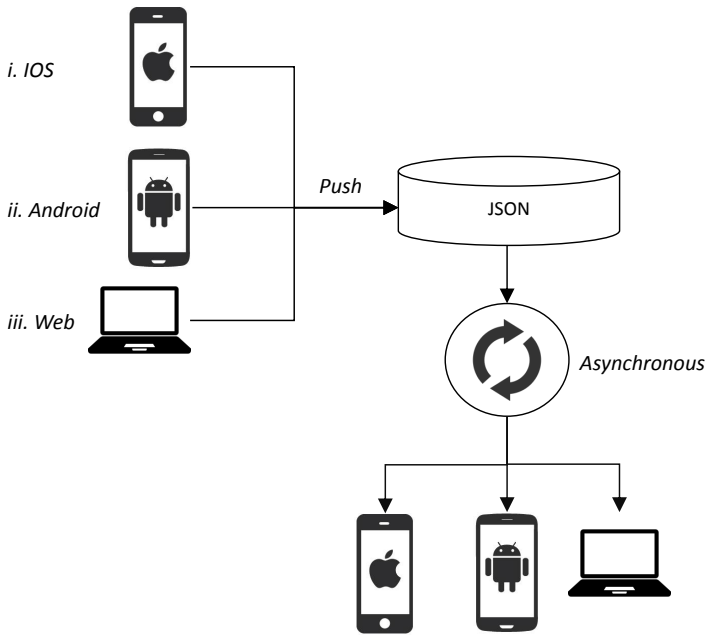


Figure 1: Tasks allocation

cases the data across the tables). However, the trade-off with this approach is that we are using more store capacity to priorice the velocity to transfer messages.

Conversation	Messages	Participants	Users
id. conversation	id conversation	id. conversation	id. users
last message	id message	id. receiver	conversation
id. sender	content	id. sender	id. receiver
timestamp	receiver		id. sender
	sender		email
	timestamp		last seen
	type of content		name
			profile picture

Figure 2: Data Base

As a comparison exercise, we start from a classic design of relational tables, which is showed in Figure XeXX. Then, we migrate the design to JSON an optimize it to priories the velocity of transfer messages.

The design specified in Figure QQQ was also proposed considered the following interaction across the tables, based on three fundamental functionalities: i) sign in, ii) sign up and iii) send messages. In the case of sign in, the user information is update with the last seen. Sign up interaction, create a new user, what add a new whole register (i.e. id, name, email, and profile picture). Finally, send message is the more interactive functionality, which started by recovering a conversation id or creating a new one and create a new message which is appended to the respectively conversation. To make easy the query of information, the table conversation keeps the conversation id, with the respectively sender and receiver id. These keys are also store in the object users.

The next section give more details about the implementation of each specific client. However, the previously three described functionalities and interaction with the database is hold for all of them.

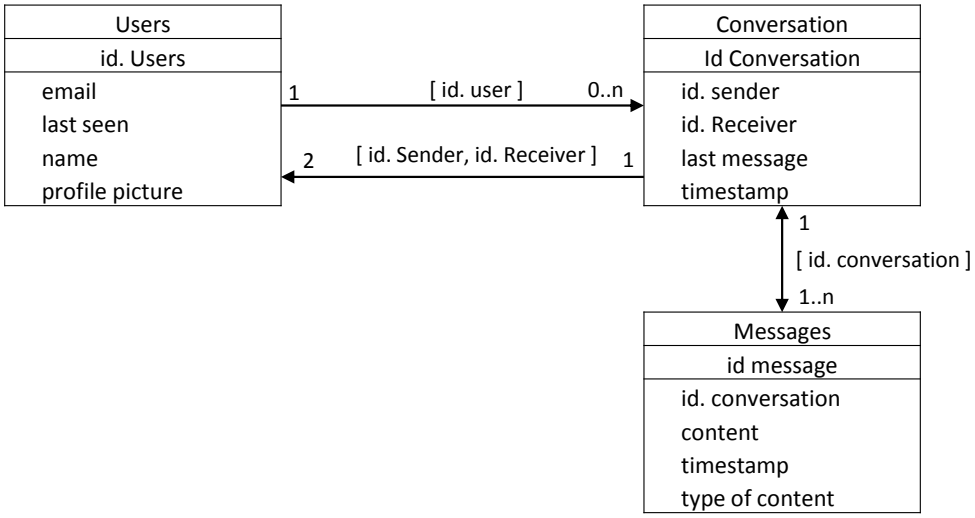


Figure 3: Relational Database initial structure

4 Implementation

This section enters in more details for specifically aspects in the implementation of each client and the backend. The next snippet code contains the rule to access the database. Thus, to read and write in the data base, the authentication process must be succeeded, otherwise is not possible to have access for these two basic functionalities. This is the way to keep suavely the users information, while security connection is guaranteed with HTPPS through Firebase.

4.1 iOS Implementation

The next figure presents the specific architecture of the iOS client.

One of the most challenging aspects were how to fetch users and check if the users have chatted before.

The next are some of the most relevant specific details across iOS development.

Fetch users: The sender must ensure that the sanded message go only to the intended receiver using the first UI (left side in Figure 2). To resolve this, we fetch the data of users as an array of array (as shown in the UI on the right of Figure 2) from the backend to have the ID of the selected user.

Check if they chat before: When the user is trying to set up a new chat, it was difficult to find out if there is a previous chat with the receiver. There were several attempts made using different approaches to figure out a solution. Finally, we were able to come up with a function that carries out the logic shown in figure 3. Once a user (User1) selects another user (User2) to chat with, the function checkIfTheyChatBefore() compares the IDs of conversations from both sender and receiver records. If there exist a common ID, then it implies that they had chatted before and the same conversation ID is used to store messages. Otherwise, a new conversation is created in the database.

External Library: We used two libraries namely; UIKit and CocoaPods. UIKit is a library already integrated in xCode which was used to make text fields, buttons, labels, and tables. However, using Cocoa Pods is mandatory for Firebase as a backend.

Code by other: We use a code written by @Esqarrouth from http://stackoverflow.com/ to write the function hideKeyboardWhenTappedAround() in SharedFunc.swift (Github: Whisper/iOS/iOS/) which hide the keyboard when user taps anything else on the screen other than the keyboard.

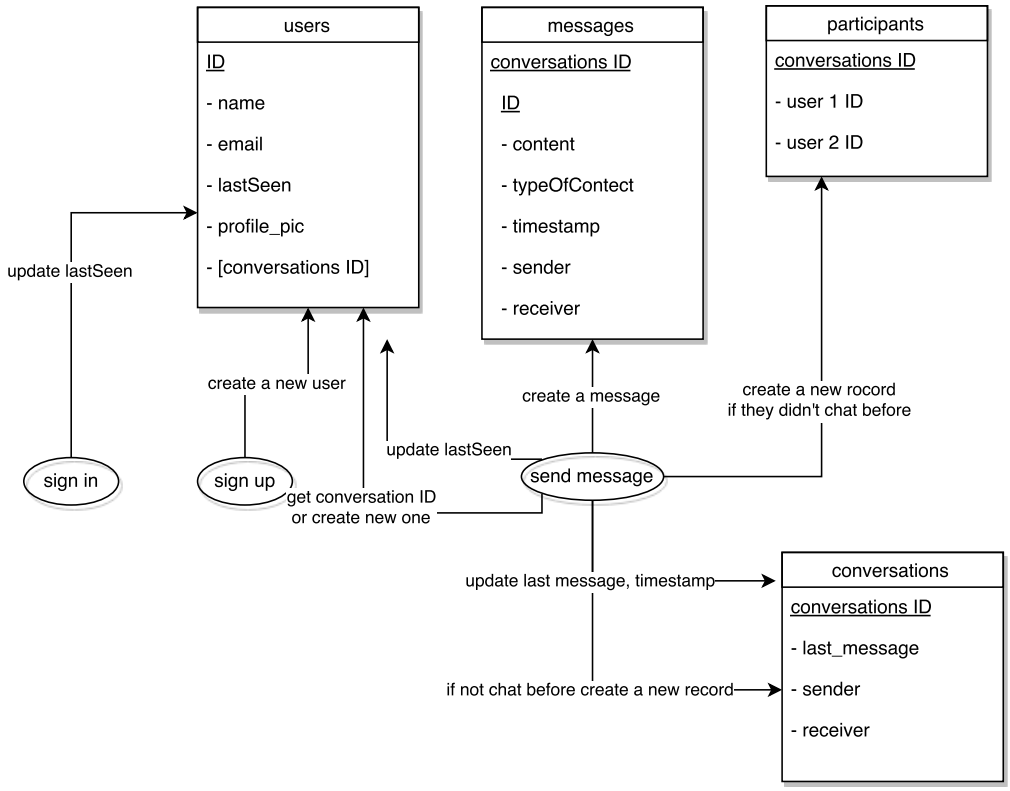


Figure 4: Database Interactions

4.2 Web Implementation

Even though web applications are all around since a long time ago. We knew that a lot of power users still prefer to handle chat on a real keyboard and not a small screen. The WhatsApp Web Client also inspired us to create the desktop version.

To create our app, we give more local insights in the implementation section of this document, however, some of the main aspects of our web app are as follows. For the registration, we use a small plugin that helps us connect with Firebase and select the providers we want to choose. We used Mail, Google and Facebook because those are the most popular.

For the whole App Wrapper, we used React JS which is a JavaScript Library developed by Facebook. React basically works as a multi component oriented framework that aims to simplify the logic of web applications by making apps more modular. It handles all the information, including user interface state in a state object that can be shared with other components or only available to one component. After some change occurs, react re-renders the changed component to reflect the new state. Along with it, we learn the hard way that a Chat application needs a lot of rendering of elements in both direction (from parent to children and backwards). Therefore, we struggled a lot and we changed our main objective by doing some research and finding out that precisely, a lot of developers had the same problem. They were communicating bidirectionally between parents and children in the DOM Tree way too many times. Thats how we learned that Whisper was an excellent use case for Redux.

Redux is another library that dispatches actions from the user interface to a reducer, which in turn sends the new state to the store (which is a global state for the app). Also with this, we learned the hard way that Redux couldnt manage asynchronous calls to a server by itself. Therefore, we used a middleware called redux-thunk which takes care of this by simply dispatching actions on demand. That is, dispatch an action when we finished, for instance, fetching elements from firebase.

Finally, for the styling we used stylus, CSS, and bootstrap to accelerate our development and learn the newest technologies and trends in the software development community. It is important to mention that we

```
"rules": {
 ".read": "auth != null",
 ".write": "auth != null",
   "users":{
     "$uid":{
       ".write": "$uid === auth.uid",
       ".read": "auth!= null"
     }
   }
 }
```

Figure 5: Database Interactions

learned a lot of web development thanks to this project. We focused in these technologies because they are among the most used online in communities like Stack Overflow, Hacker News, and Reddit Web Development related subreddits. Of course, there exists people with different opinion and we understand that the tool depends on the job.

To create the web client, we were sure that the options were the opposite of limited. Contrasting with iOS and Android, there are so many options out there to begin Web App Development. Of course, HTML and CSS are just not enough for this kind of project. As we know, HTML only serves the purpose of giving the structure to the web. That is, structuring all the information inside the DOM hierarchically and in order, tag by tag and element by element. As the name suggests CSS (Cascading Style Sheets), give the style or appearance to each HTML tag inside the DOM, properties such as position, color, font weight, variant and sizes, background images, padding, margin, etc., are the ones that CSS controls so users can have a much better experience.

However, we knew that this project was going to be heavy on the scripting side; we started looking out for technologies to take our development to the next level. Some of our research suggested that we used a powerful framework that took care of the rendering of elements almost in real time. To take advantages of the asynchronous functionality of Firebase, we decided to try it with the following various contestants: Vanilla JavaScript, AngularJS, ReactJS, jQuery and Ember.

Pure JavaScript is always needed for every heavy front-end development app, because thats the scripting part of it. However, pure JavaScript can lead to messy code and just recently with ES6 and ES7, with modular exports, is getting more friendly towards heavy applications. Although we are not experts at all and some of us have a little bit more experience using Python and Django for Back-End software development, we see a lot of resemblance between JavaScript classes and exports with Django Class Views.

AngularJS was interesting because is powered by Google and we consider this as a plus. Also, there are tons of material out there to learn, from community blog posts and books to complete courses on Coursera. On the other hand, jQuery is a JavaScript Library that has been out there since 2006, and although front-end frameworks change continuously [5656rtyrt], we knew that jQuery was a leader at the time and for some basic stuff like AJAX is still the go to for full-stack developers.

React; being developed by Facebook, we knew that had a lot of benefits thanks to the hype around internet blogs and the Stack Overflow Community Survey of 2017 [5657g6rrt], however we didnt know the potential until we read the excellently well written documentation of it.

To be honest, Reacts documentation, ease of use and hype, gave us the confidence to pick it up as a great library/framework to learn from. Thats why we choose it for the Web Client development.

At first it was very intimidating because React is used mainly by good software engineers in the front-end development industry. React facilitates most of the job on front-end development because it suggests to create your web applications using components for each task that needs to be done by your app. Treating every bit of it as an isolated component that can even be reused on different future apps. These isolated components just fulfill one simple task to make your application easier to understand to other developers and for your colleagues.

React is a little bit intimidating, some of the information out there can be confusing because to develop for it you need a little extra setup. Not like iOS or Android which only need Xcode and Android Studio

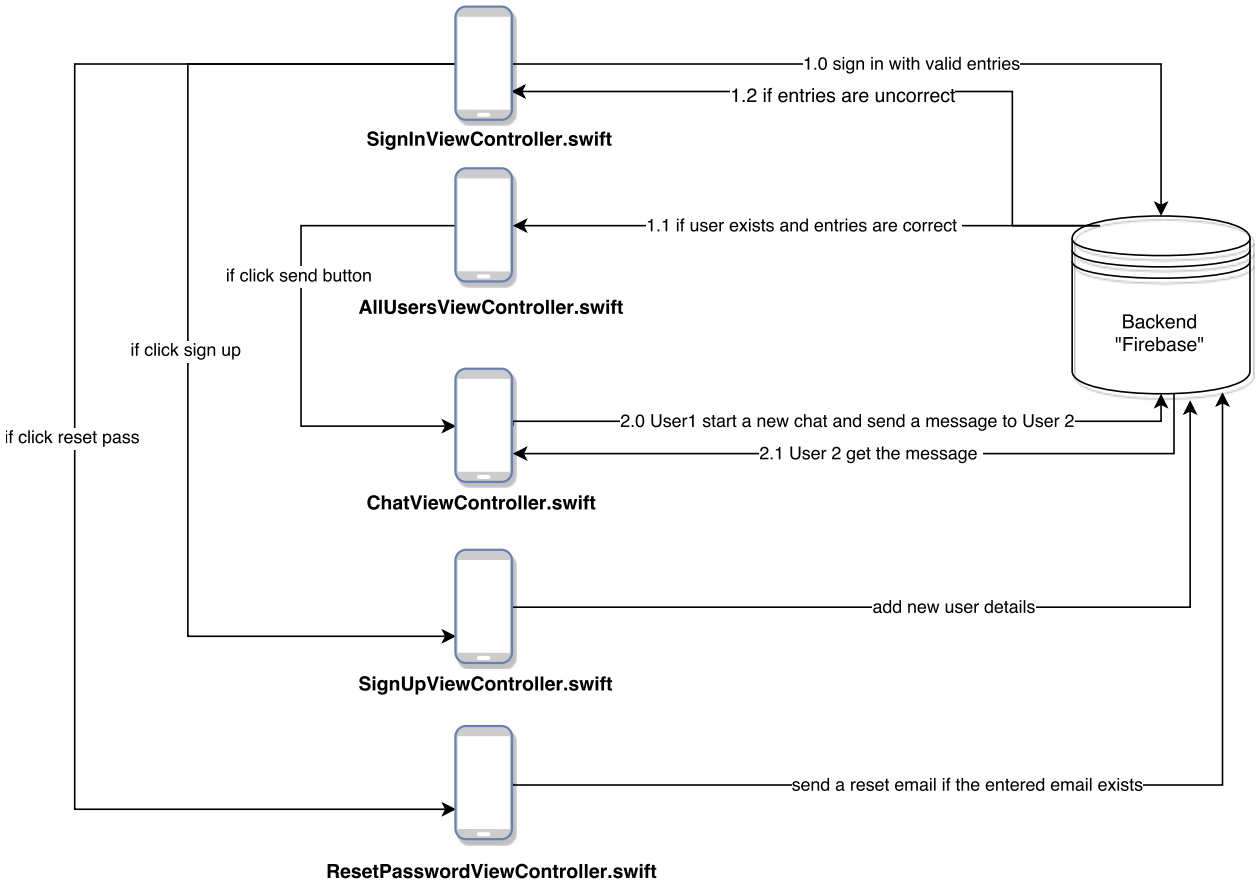


Figure 6: Database Interactions

respectively.

In contrast, for React, if you really want to take advantage of all the features you need to install NodeJS, npm (included) and Babel to transpile JavaScript syntax for older browsers. Also, they always suggest in all the tutorials to use Gulp or Grunt to create tasks that automate a lot of the repetitive and tedious tasks inside front-end development such as compiling the different JavaScript source files, merging together all the CSS or style files, and running web pack to refresh your website with hot reloading whenever a new change is detected (basically to avoid refreshing the browser window).

All these tools that we have just mentioned are just needed for the development. For the actual code to make its job and suggested by React inside their documentation, they advise to use JSX, which is basically a way of writing JavaScript and HTML inside your JavaScript files.

Modular components and state are the most important aspects behind React development. For our case, it was very useful because our code was structured this way to achieve way more readability (on most of the cases) and to think about the big picture easily. And state management was very helpful because we could see state in real time thanks to the Chrome React extension. By this, the debugging process is less painful and much more friendly.

Some aspects of JavaScript language specifically made us struggle a lot, mainly because we did not have enough experience but mostly because of the ES5/ES6/ES7 flavor and how to use some of the code that you find online is using a different standard than the actual web browsers can render. That is why many people suggest using a library called Babel which was created by one of Facebooks employee [4646eyyeye].

This library gives the developer the ability of using fat arrow functions:

Some of these features of the JavaScript programming language can get a novice developer a lot of headaches. The one problem that really messed up everything was the this keyword in our functions.

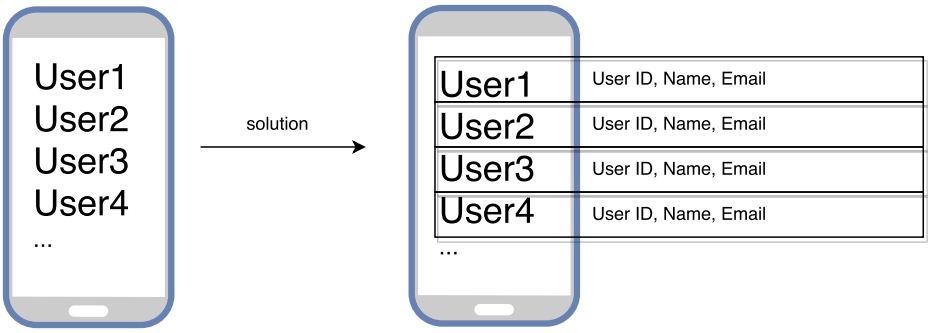


Figure 7: Database Interactions

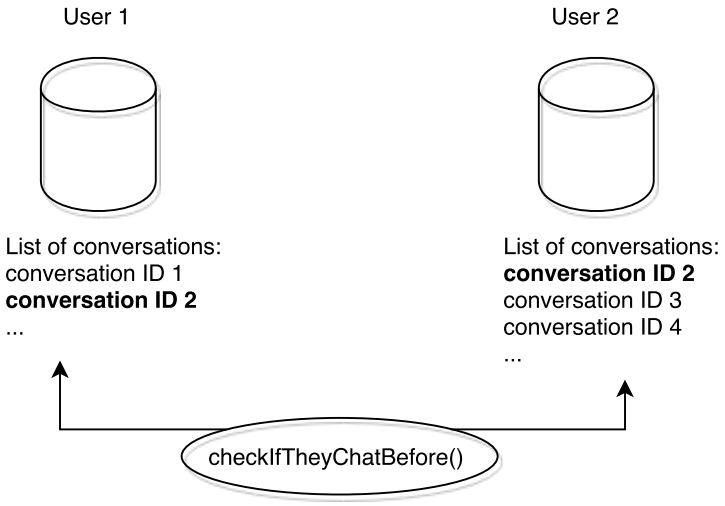


Figure 8: Database Interactions

Apparently in JavaScript this is not the same as, for example, self keyword in Python. More on that later when we explain Firebase integration.

In React, we opted to have the following main components which can be found inside the components directory:

1. **AppWrapper**: The whole container wrapping all the components together and passing the state from Redux (more on that later) to the rest of the components.
2. **ConversationsSidebar**: To render on the left side of the app, the conversations that each user have.
3. **ConversationPanel**: The right-side panel that shows all the messages sent between users inside the selected conversation of the ConversationSidebar.
4. **UserDrawer**: In charge of showing all the users available inside our application in the Users JSON (except the current logged user).
5. **AddMessage**: This is the main component that takes care of dispatching most of the logic whenever is triggered. This is the main ¡form¿ element that adds the message to the Messages, Conversations, and Participants JSON when needed.

Previously, we mentioned Redux. This is another important library inside React Development community. Basically, we tried it the hard way at first. We were using pure React to develop our app but because in

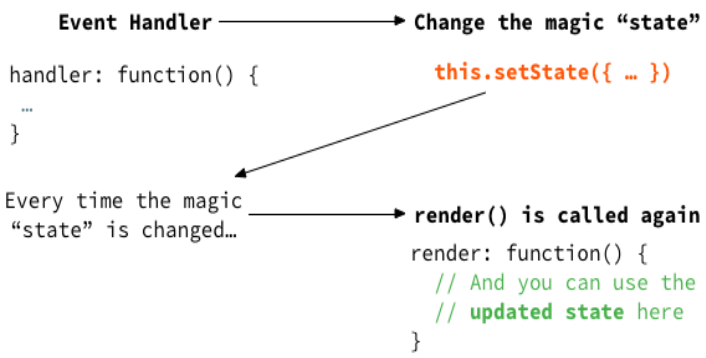


Figure 9: React.js style

Before:
Function doThis(){
return "this!";
}

After:
doThis () =>{
return "this!!;
}

Figure 10: before

react you must pass all the state from the parents all the way through the children that needs that piece of state, code can get messy. This image illustrates in a practical sense how Redux solves a lot of the passing props from one side to another easily by using an independent store that handles all the data.

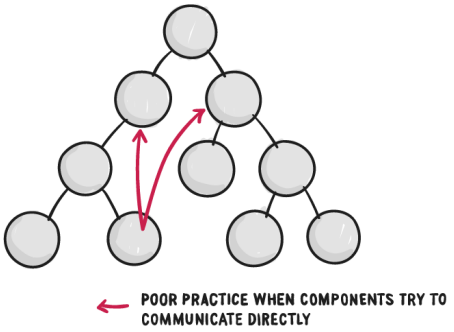


Figure 11: Poor Practice

We were really advanced with the application when we realize that code started to smell a lot. Thats why in our commit history we show a drastic change of code to implement Redux.

Using Redux was not as easy as we thought. The theory behind it is simple, use a store that holds all your app state (conversations, users, messages, and participants), fill that state when needed by using Actions and Dispatchers and then return the modified state with the Reducers. Actions is a very simple concept, every time something happens in the client, an action is dispatched. When this action is dispatched, all the reducers listen, if they own the given action, then they update the state accordingly. One of the most important parts of redux is that Reducers need to be pure functions. The whole state of your app is stored

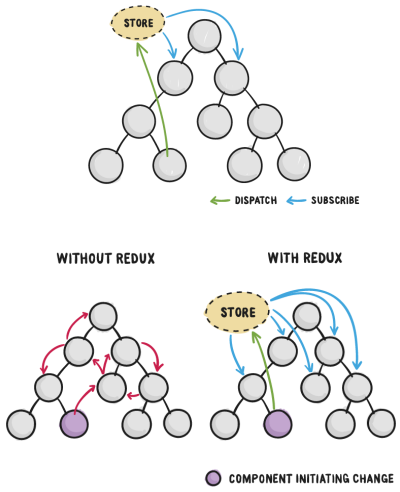


Figure 12: Poor rtrtrtryeyeyeye

in an object tree inside a single store. The only way to change the state tree is to emit an action, an object describing what happened. To specify how the actions, transform the state tree, you write pure reducers. From Redux Documentation http://redux.js.org/

Pure functions mean that you should not mutate the state object, but return a new object if the state changes. This is tricky at first and we ended up making a lot of mistakes on the first implementation, however, Reacts and Redux documentation explain it well, however we didnt focus too much at first which in the end resulted in these errors of strange behavior between state changes.

All our logic for Reducers was added to a folder called Reducers. The same thing for Actions, all the actions are inside the actionCreator.js and the Constants for it are on the file with the same name.

The next diagram explains how our Whisper Web Client handles actions and state.

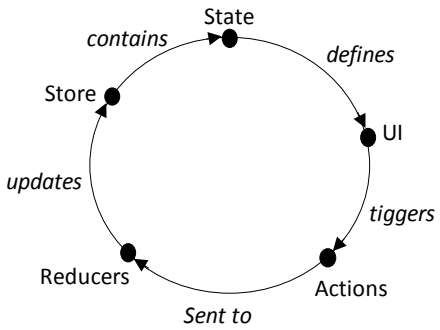


Figure 13: Web client actions and states cycle

Integrating Firebase was one of the most complex tasks because even though the documentation is excellent, there is not enough information on how to integrate Firebase with React and Redux. When we started, our app using only React, everything was handled by functions inside the components. However, because of Redux nature of pure functions Asynchronous calls where a struggle for us. Thats why we needed to add a middleware called redux-thunk [46rtrtet]. This is another library widely used by the React Community. However, there is also another one gaining popularity called redux-saga [5656ry] which solves the same issues, doing asynchronous calls inside redux, that is, fetch and update firebase whenever an action is dispatched via redux.

Previously we mentioned how we needed to use Babel to compile our code because of the new fat arrow

functions and the this keyword. With Firebase, some of the methods didnt work as expected inside React. After much scratching, we found a solution on StackOverflow which told us that with React and Firebase you had to either bind this keyword or use a fat arrow which does this automatically for you. At first the fat arrow looked intimidating, but it was less boilerplate code for the rest of the app, so we decided to use this newer feature inside JavaScript.

Once we got this issue solved we got on speed, we were getting all the information from firebase as we needed. The first part was fetching the conversations from firebase. We used some sample data following the structure we mentioned before. With Firebase, we fetched all the conversations that the loggedUser had in their conversations object. This was one of the most important highlights of our project, because as opposed to a regular SQL Database, there is no such thing as SELECT * FROM Conversations where conversationID in (SELECT Conversations FROM User WHERE User = X) With firebase we had first to read the loggedUser, which by that time was a hard-coded uid which is the Id that Firebase gives to every new registered user. After that, we needed to fetch all the conversations from that User. And when we finally got those conversations in the same step we needed to fetch all of to the Conversations of the Conversation reference object that were the same as the Users. This code was really hard to achieve, so we are really proud we got it:

```
// FIREBASE conversations
function fetchConversationsFromFirebase() {
  return (dispatch, getState) => {
    let loggedUser = getState().auth.uid;
    var conversations = {}
    if (loggedUser !== null) {
      var userConversations = ref.child('users/' + loggedUser);
      userConversations.on('value', (userSnapshot) => {
        userSnapshot.child('conversations').forEach((conversationKey) => {
          var conversationRef = ref.child('conversations').child(conversationKey.key);
          conversationRef.orderByChild('timestamp').on('value', (conversationsSnapshot) => {
            var conversation = conversationsSnapshot.val();
            conversations[conversationKey.key] = conversation;
          });
        });
      });
    }
    dispatch({
      type: C.FETCH_CONVERSATIONS,
      conversations
    })
  }
}
```

Figure 14: Fetch conversation from Firebase

This was difficult and we wasted a lot of time on learning how to do this. A lot of the Firebase documentation recommends doing a flat database structure precisely because of this, but we didnt quite get the concept of NoSQL databases until we stumble in a post [tutyuty] which mentions denormalization, a concept that we completely understand the opposite way when learning about relational databases.

This part of our development was the holy grail. Thanks to this code we didnt have more difficulties with firebase and our structure proved fruitful. That is, according to our needs and scope.

Firebase is a back-end tool, framework or even platform that is often referred as a Backend as a Service. The real-time database is one of the most used features for projects that want speed and real time exchange of information, thats why we choose it as we mentioned before. However, another important aspect of this is that a lot of the back-end code gets done on the client. To create a new user on Firebase, we needed to register in Firebase, after that, we needed to create the user inside the Users database and filling all the fields correctly. It contains validation rules, however most of them are handled on the front-end.

For the Participants JSON, we needed to get the Sender and the Receiver id. After that, we created a Conversation Key using Firebase Database Push method which gives the developer a unique key. After that, we added both users to the participants object as well as the conversation key to each of the Users object. This kind of seems repetitive but thats one of the main reasons why Firebase is so fast. Because it only stores simple objects of string data. The Messages functionality was the most fun for us. We wanted to feel like a professional application, thats why we grabbed a lot of inspiration from the WhatsApp Web Client. We even created our bubbles inspired by theirs. Even though CSS is just a styling language it also gave us some lessons.

It was a little bit complex to learn how to add new messages to the Conversation inside the Messages dictionary, again, it sounds counterintuitive but it did work as intended, we erased some of the sample information inside our Testing Firebase App, however we learned a lot in the process. Adding new messages

was one of the most complex tasks, first we needed to get a new key from Firebase. Then if the conversation didnt exist we needed to create it. However, if it existed, we needed to get it from the Participants dictionary. The dictionary that contained both the sender and the receiver was the conversation that needed to get a new message. Again, this was another tricky part that we are proud of having solved.

Some remarks of our software development experience are also that the community in stack overflow and other IRC channels, were helpful. Also, there is a lot of people out there showing their contributions and doing open source code. That excites us a lot and makes us feel more challenged for the future.

Our conclusions for the implementation of the Web client are that we are very happy that we got the minimum functionality working, we wanted to do more as we showed on our initial timetable, however we now feel more experienced and we learned by firsthand the joy of developing with React. Which sounds like in the future is going to keep being useful (see React Virtual Reality [4545etet]).

4.3 Android Implementation

As new programmers, we started by looking at existing messaging apps that use firebase as backend so we could take ideas on how to start building our app. There is a course in Udacity made by Google that makes a basic chat (all to all) that we had as reference for our app. [https://www.udacity.com/course/firebase-in-a-weekend-by-google-android–ud0352].

Later on, we found out that this app was very different to what we needed. We needed a different database structure to be able to map each message to the appropriate conversation. With this came many complications. We had to change the whole structure of our program! We tried changing our code, but we found out that it was a hard task so we decided to start a new project from zero because it would be easier.

We had many problems during the way but only one that we were not able to fix. We didnt manage to append a new message to the previous list of messages of a specific conversation.

We think that the steps to do it are as follow, but we dont know how to implement them:

Get the user id (the key) of the sender [Done] Get the user id of the receiver Search inside each participants children if the userid of the sender and receiver match the keys stored with a value of true. Get the participants children that found before. This is the key for the conversation for these 2 participants. Go to messages $key_{f}ound_{i}n_{p}revious_{s}tepappendthemessage.Theandroidappdoesnthavearegisterfunctionalitybecausetheprioritywast$
$Chat : Containingtheinstancesofsender, receiver, senderUid, receiverUid, messageandtimestamp.Chat_{r}oom :$
$Thisclassisinchargeofwhathappensinsidetheconversation.Italreadyknowswhothesenderandreceiverofthemessageareand$
$Thisclassisinchargeoftheloginprocess.Incontrastwiththeotherclients, anewusercanonlysigninandnotsignup.Theusercano$

Class containing the instances of name, userId and email. UserListActivity:

Class containing a method to get all the users from Firebase. This class is also responsible for creating new chats. If the user tries to open a new chat with an existing UsersArrayAdapter Special class made for android to be able to map a list of users to a list view. We display thanks to cellprototype.xm

4.4 Testing

For the test, we are inspired in the verification and validation model (V- model), which is based on the relationship between each phase of a software development life cycle, were and associated testing phase is added. The next figures summarize each phase of the methodology, which also is an integral part of our developed philosophy.

Despite we are aware of the agile test methodology model (where each little division or task of software is tested while developing [rtyeteey]), in our case this was not at all useful, considering that we have a priority requirements list. This means that the divisions of our software, as well tour architecture, changes according with our learning curve across this project. Thus, our initial plan or division of the software were extremely dynamically to adequality divided since the beginning. In fact, we can say that this is the main challenge our group face. The next table presents the results of the testing process. An iterative process was conducted to guarantee a proper integration of the different clients. Also, this process is challenging, considering that while we are trying to develop each client, at the same time we must be aware the whole structure of the system.

In addition to the previous methodology, we can also have said that our testing process was iterative across the development in the following sense:

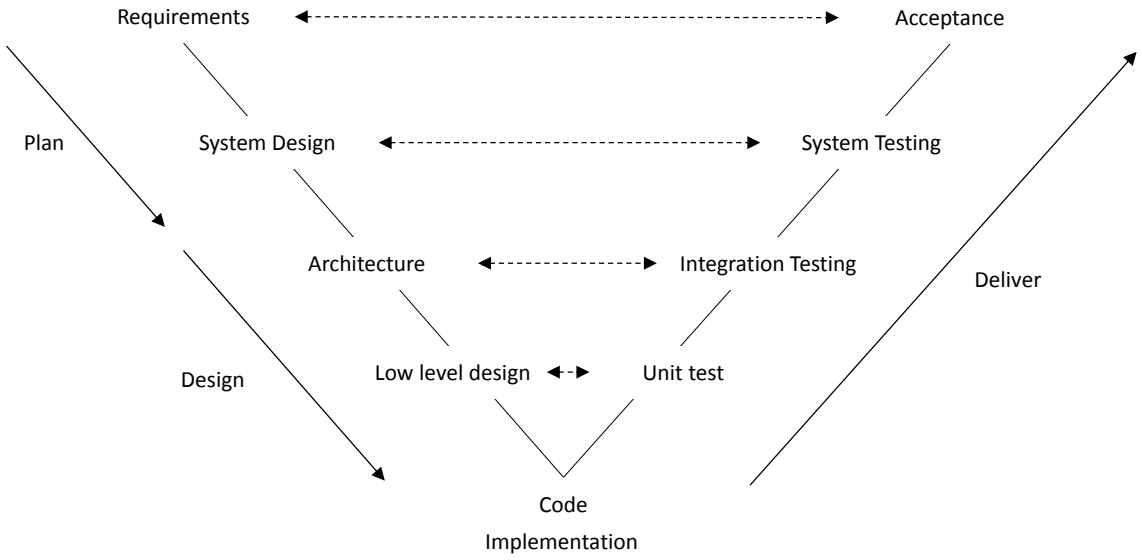


Figure 15: Test foundation Methodology

Initial Testing. Initial testing was the first test conducted on the application. The testing process began with the testing of the login unit/page that was demonstrated during the initial presentation. At that point the testing was platform specific. There was no need for testing together across other platforms then. However, as the development progressed, our testing methodology was modified.

Development Phase Testing. These are the testing conducted through the development phase of the project. The testing done during the development phase was such that the codes were shared amongst the team as soon as they were observed to be working and deleted or overwritten once it was found not to be working or not to meet our desired end. The testing was done intermittently as the project was being developed. The result of each test determined progression to the next phase. No member of the group was dedicated as a test engineer neither was any sub-team assigned the responsibility of testing the software. However, as each sub-team was working on its project, the test was also going on. Whoever was writing the code was a test engineer in his own right. Then when the team meets, the codes were run across platforms to query its compatibility and flexibility across platforms. The team members from the different platforms then criticize the codes based on their observations (Red Teaming) and suggested necessary modifications.

Final/Integration Testing. The final test is an integrated testing at the end of the project. The final testing was done to ensure that all the users were able to carry out one to one conversation across the three clients platforms. At this stage, we encountered a challenge of the inability of the three clients bases to communicate directly with each other. However, the database structure was re-examined and the sub-teams developing the clients were made to restructure their backend to follow the same format. Then, the intercommunications issue across the three platforms was resolved.

References

[1] Cohn M. Succeeding With Agile-Software Development Using Scrum. *Pearson Education: Boston*, 2009.

[2] Wooldridge M. An Introduction to Multiagent Systems. *Department of Computer Science, University of Liverpool, UK, John Wiley Sons*, 0-471-4969 I-X, 2002.

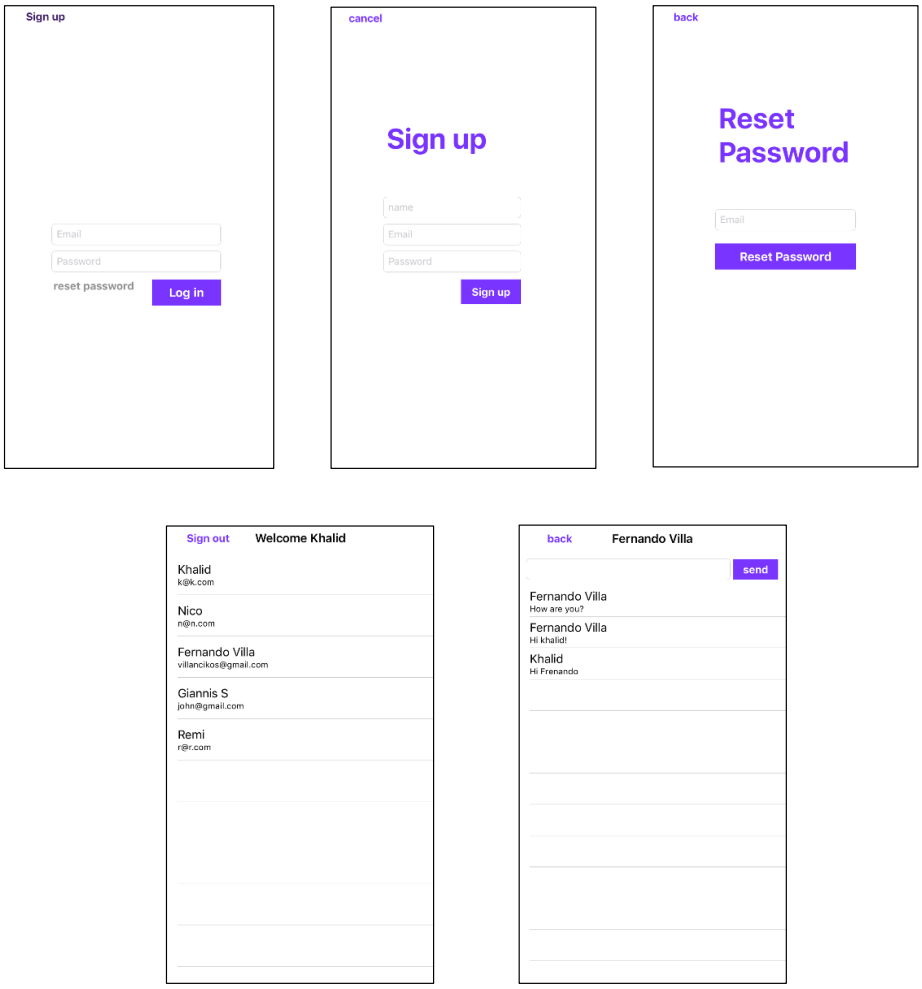


Figure 16: iOS Screenshots

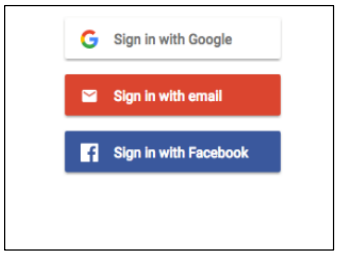


Figure 17: weblog

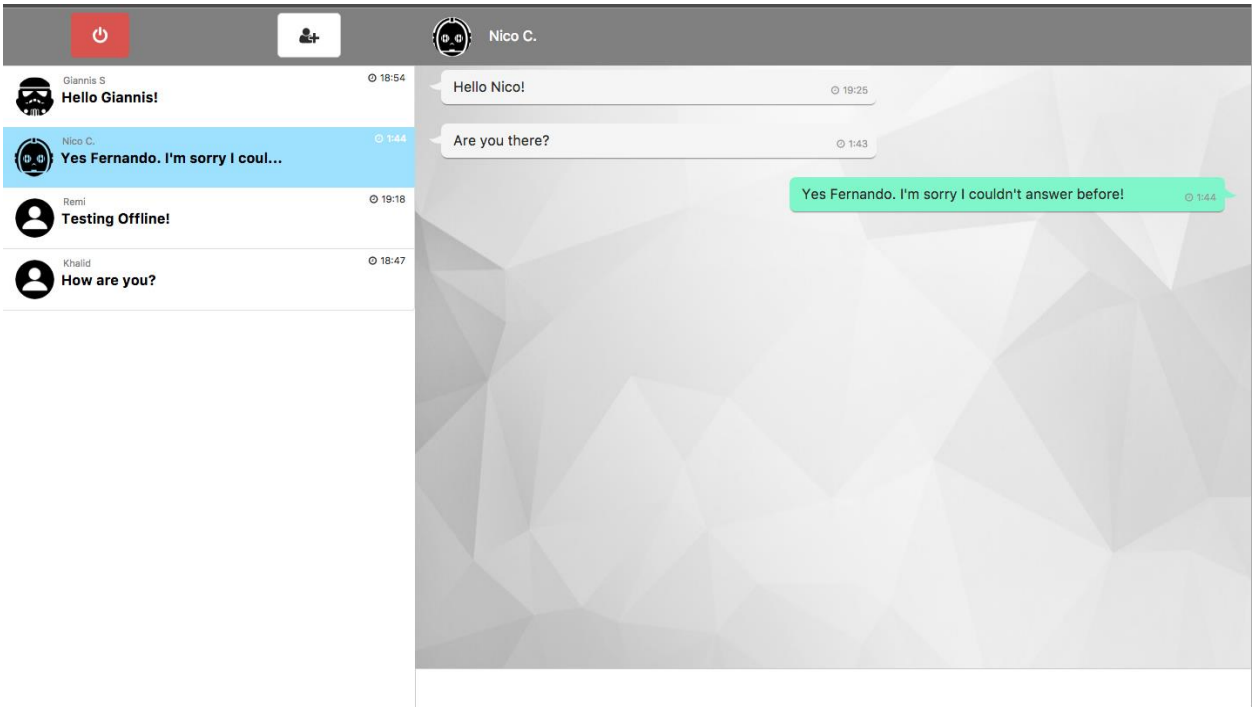
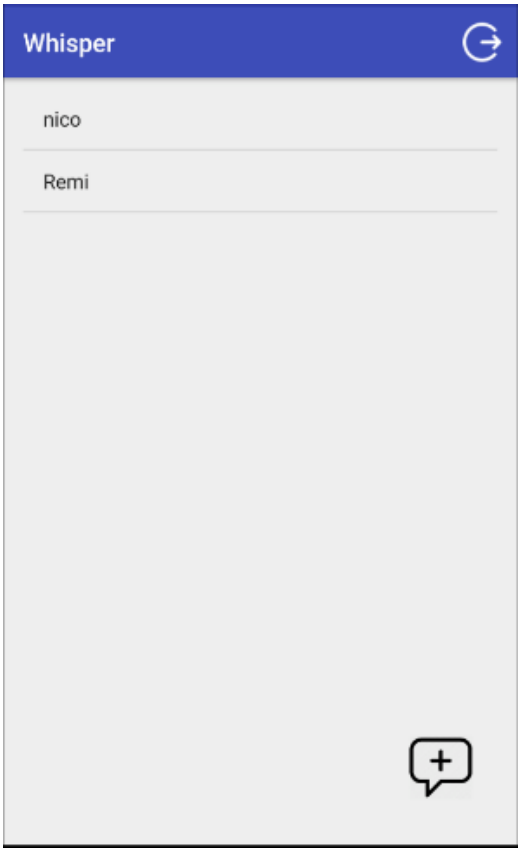
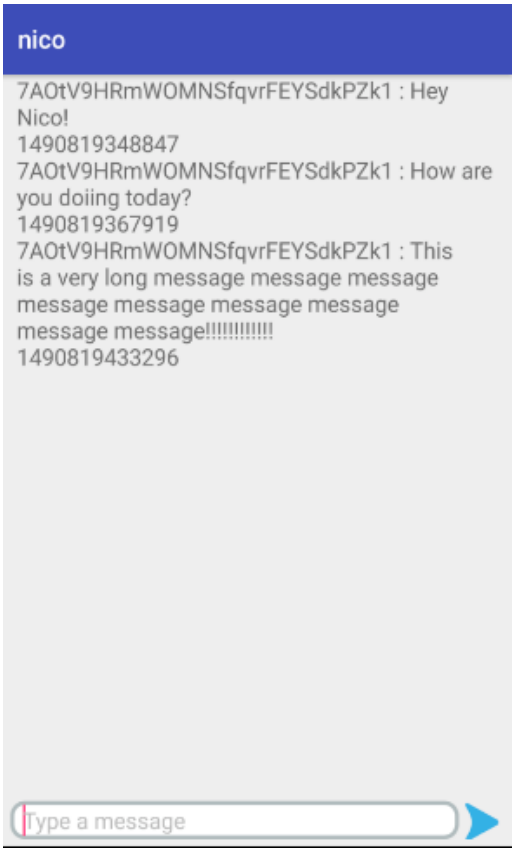


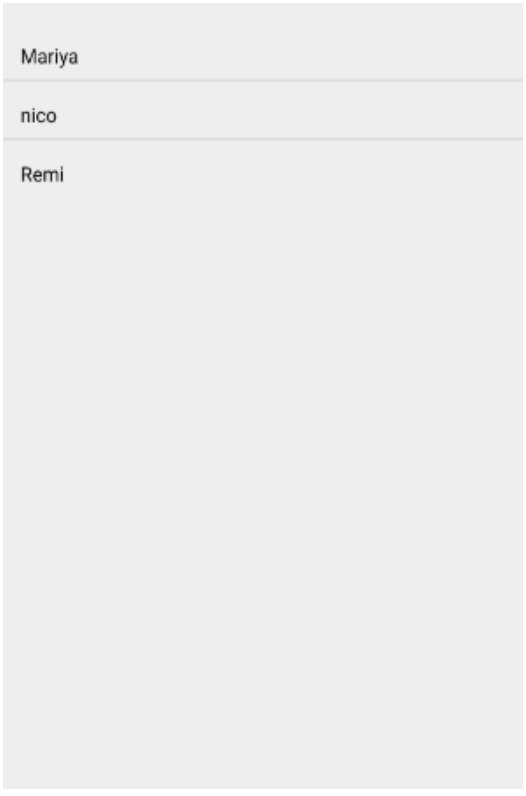
Figure 18: Chat-sample-Conversation



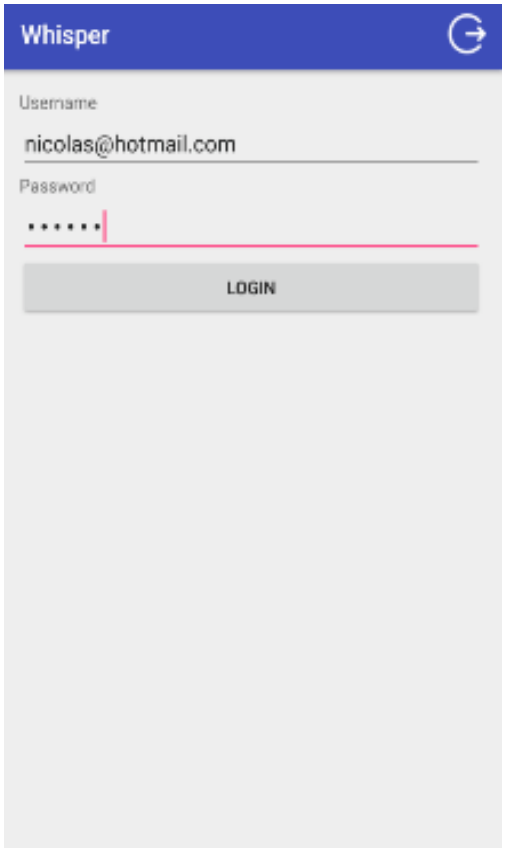
(a) Main Screen



(b) Chat



(c) Create New Conversation



(d) Create New Conversation

Figure 19: Android Screenshots