Monolith

by Red Babel.

the dawn of chat.

Red Babel.

1 Introduction

Today's most important communication channel is the chat enabled by a variety of chat apps. We use chats in every aspect of our lives: with friends, for business, with people we collaborate with, to make a reservation etc etc. Even though the concept itself is quite old, it hasn't changed much over the years. Only in recent times[1], the business world found in the chat the perfect way to get closer to their customers, simply because they are communicating with them by using the same channels that people use everyday in their life.

Both businesses and customers need to be "efficient" in this interaction. They need to get a sale or a piece of information as speedily as possible, by using less resources as possible. On the other hand, the user experience must be pleasant and smooth.

Similarly, during our day to day experience, it happens that we need to interact with groups of people sharing pieces of information via chat. The only way to do it, at this stage, is to exchange a fair amount of messages, and this leads to a big waste of time and energy.

In order to resolve this problem we are observing the evolution of chats into something more interactive. The main players in the market are all moving towards similar solutions:

1. Slack, message buttons: https://api.slack.com/docs/message-buttons;
2. Facebook, interactive messages: https://developers.facebook.com/docs/messenger-platform;
3. Apple, interactive messages: https://developer.apple.com/reference/messages.

In the current tender we are inspired by a small company: Cola.io[2]. They propose the concept of **interactive bubbles**.

Interactive bubbles allow adding more features to the chat app, without installing a new one. We can think of interactive bubbles as mini-apps inside the chat app.

[1] https://hbr.org/2016/09/messaging-apps-are-changing-how-companies-talk-with-customers
[2] https://www.cola.io/bubbles

2 Interactive bubbles

The goal of the project is to create a framework that allows the developer to create interactive bubbles easily. The interactive bubble, created using Monolith, must be able to work inside the Rocket.chat environment.

Rocket.Chat[3] is a Web Chat platform and is built as an open source clone of the Slack platform. It is developed in JavaScript using Meteor[4], a free and open-source JavaScript web framework written using Node.js[5]. Rocket.chat allows a high degree of extensibility and customizability. As such is able to cover a wide range of use cases.

2.1 Types of interactive bubbles

The interactive bubble developer, using Monolith, will be able to create bubbles that belongs to one of the following category.

1. Rich media bubble;
2. Self-updating bubble;
3. Editing bubble.

Rich media bubble

Rich media bubbles are shortcuts for sharing some contents. A good examples is a bubble that gives the preview of a Youtube video or animates a gif.
Some examples:

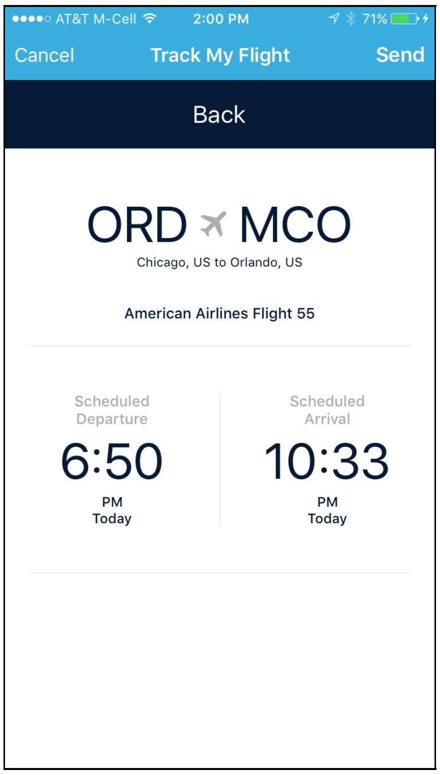
1. Video: when a participant sends a link from youtube to the chat, the link is visualized as an interactive bubble, allowing participants to play the video;
2. Link preview: when a participant send a link from youtube in the chat, the link is visualized as an interactive bubble, showing the preview of the website/content;
3. Gif: a bubble that shows an animated gif in the chat.

[3] https://rocket.chat
[4] https://www.meteor.com
[5] https://nodejs.org

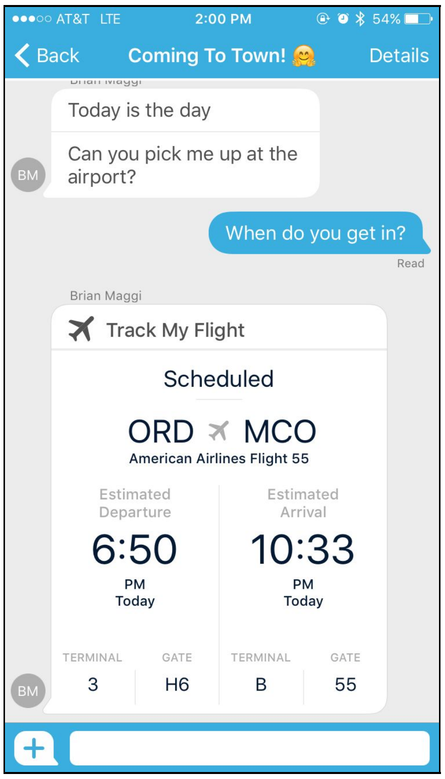
Self-updating bubble

This type of interactive bubble provides pieces of information that can change over time. A typical example is a flight tracking information bubble. It updates the displayed info about a flight every once in awhile. Some examples are:

1. Flight tracking: this interactive bubble shows the information related to the specified flight. Bubble updates the information automatically;
2. Weather forecast: this interactive bubble shows the weather information as specified by the creator. The map (or whatever the information is displayed) updates periodically;
3. Price tag. The bubble shows the price and the features of a product. The price might change over time, the bubble updates accordingly.



Flight tracking in editing mode (from cola.io)

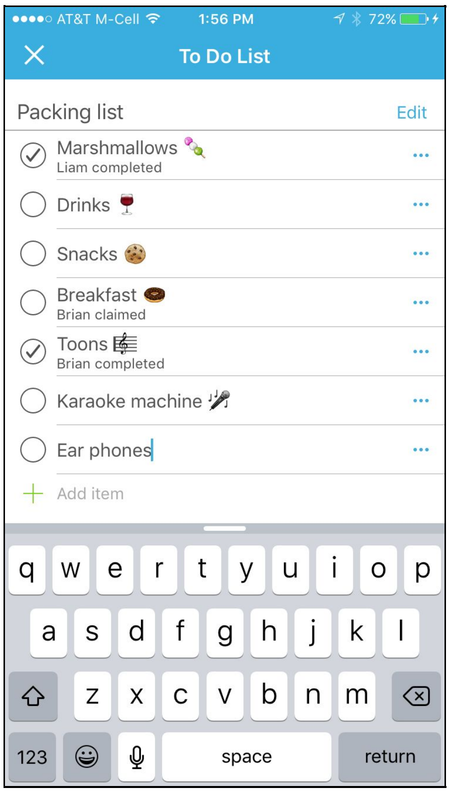


Flight tracking in view mode (from cola.io)

Editing bubble

This type of bubble allows the recipients of the bubble to make changes and updates in a collaborative way. A fit example is the quick poll. Someone asks for the other people's opinion by laying down a list of choices. The recipients have to vote for one of the choices from the list, everyone in the room can see the outcome of the poll in real time.
Some examples are:

1. Poll: one of the participant in the chat, the creator, casts the poll by creating the pool of things to vote from. The others can upvote or downvote one or more element in that list;
2. Todo list: the creator of the todo list casts the list in the chat, he can add/delete/edit the various elements. The creator can decide whether or not the list can be edited by the other participants;
3. Upload file: the creator allows the other participants to upload one or more files to a third party server. Only the creator can see whether or not each participant uploaded the requested file(s).

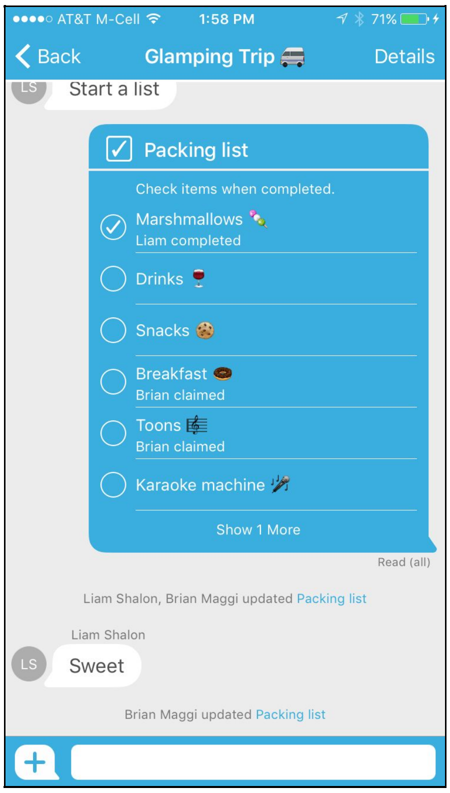

●●●●○ AT&T M-Cell 1:56 PM 72%

To Do List

Packing list — Edit

- ✓ Marshmallows 🍡 — Liam completed
- ○ Drinks 🍷
- ○ Snacks 🍪
- ○ Breakfast 🍩 — Brian claimed
- ✓ Toons 🎼 — Brian completed
- ○ Karaoke machine 🎤
- ○ Ear phones
- Add item

Todo list in edit mode (from cola.io)


●●●●○ AT&T M-Cell 1:58 PM 71%

Back — Glamping Trip 🚐 — Details

LS Start a list

Packing list

Check items when completed.

- ✓ Marshmallows 🍡 — Liam completed
- ○ Drinks 🍷
- ○ Snacks 🍪
- ○ Breakfast 🍩 — Brian claimed
- ○ Toons 🎼 — Brian claimed
- ○ Karaoke machine 🎤

Show 1 More

Read (all)

Liam Shalon, Brian Maggi updated Packing list

Liam Shalon

LS Sweet

Brian Maggi updated Packing list

Todo list in collaborative mode (from cola.io)

3 Use case ideas

Bubble as a Service

Provide a software as a service (SaaS) that can deliver the bubbles on demand. The Rocket.chat administrator should be able to install a package provided by the SaaS. This package is then configured via an admin interface provided by the package itself. For example, the configuration should include what bubbles are fetched from the SaaS, and, if needed, the admin can configure specific bubbles.
On the SaaS side, users can manage their interactive bubbles (e.g. upload/remove/update) and some degree of administration should be made available. For example, a bubble could be available to every Rocket.chat server (i.e. public bubble) or only to the allowed ones (i.e. private bubble).

Customer communication dashboard

Consider a company that uses a chat system to communicate with their users. The company offers loans by chat. If the user wants to borrow some money, she starts chatting with the operator. On the other side, the operator has to check if the user has all the requirements to pay back the loan. In order to assess the customer's credit score, the operator asks her to upload a list of documents (ex: the payslip from the last year), fill up forms and so on. Once the operator has everything needed for the deal (and all prerequisites are met), the money can be wired to the customer's bank account.
In order to have this kind of interaction in the most efficient way, the operator sends the customer different kinds of interactive bubbles. By reading the history of the chat, the operator (and the customer) can easily check if all pieces of information have been delivered correctly. All the files and contents sent by the users are stored in the company's database, different from the chat database.

Smart bot

Provide a platform where a user can interact with a system, without (or with very little) human intervention by an operator. The system, nicked SAL, can ask the user some questions and understands the user's questions in return. SAL answers accordingly using, also, interactive bubbles. SAL can be programmed to become able to interact with the user for a specific business case.

4 Requirements

Monolith

1. Implementation of the framework Monolith as a package for Rocket.chat.
2. Monolith as a standalone package should be easily installable and deployable in a Rocket.cat server instance (i.e. as the moment of writing using the meteor package manager: `` `meteor add monolith` ``);
3. The bare functionalities that Monolith should provide are:
    1. a set of predefined bubbles to be readily available to the end users;
    2. a set of API for developers to develop and release new bubbles.

Showcase demo

The vendor is tasked to show the Monolith capabilities by implementing a meaningful use case. In section 3 of this tender some examples are provided. Those ideas should be considered more of a starting point than a requirement and the vendor is highly encouraged to expand and explore beyond those ideas.
The demo is therefore to be considered integral part of the tender.

4.1 Technology specification

1. The framework and the demo must be developed using Javascript 6th edition (ES6) using a promise[6] centric approach. The usage of callback must be limited and thoroughly justified (i.e. do not use them);
2. The Airbnb Javascript style guide[7] must be used and enforced using ESLint[8] throughout the development process;
3. The framework and the demo should follow as close as possible, when relevant, the *12 Factors app*[9] guidelines. The application of the 12 guidelines must be documented;
4. The usage of a frontend framework is recommended, when applicable (i.e. : Angular 2, React).
5. The usage of SCSS[10] is preferred;
6. The demo should be deployable to Heroku, a cloud platform as a service (PaaS) that supports several programming languages. Deploying applications to Heroku generally means pushing an app to Heroku using Git, a distributed version control system for software development;
7. The source code of both Monolith and the demo must be published and versioned using either GitHub or Bitbucket.

[6] https://exploringjs.com/es6/ch_promises.html
[7] https://github.com/airbnb/javascript
[8] https://github.com/eslint/eslint
[9] https://12factor.net
[10] http://sass-lang.com

4.2 Documentation

Along with submission of the implementation of the framework and the demo, it has to be submitted: the user manual and every necessary documentation that allows the end user to use the system. All the external documentation must be written in English.

4.3 Warranty and maintenance

The vendor has to demonstrate during the R.A. *(Revisione di accettazione)* that the product works correctly and accordingly to the specified requirements. Fixing bugs and flaws not compliant to the requirements are entirely at the expenses of the vendor.

4.4 Credits and Licence

RedBabel has interests into this project as proof of concept using the technologies specified above. The system will be distributed under the MIT license, the vendor will be mentioned in the copyright credits. Also RedBabel will be credited under the section credits in the README file. Please refer to the Italian law about the management of public bids for what is not specified in this technical specifications document.

5 The proponent

RedBabel is a Webcraft consulting firm based in Amsterdam formed by Alessandro Maccagnan and Milo Ertola. The firm operates in Amsterdam and Italy where it holds close relationships with the respective startup ecosystems.
Contacts:
Alessandro Maccagnan: alessandro@redbabel.com
Milo Ertola: milo@redbabel.com