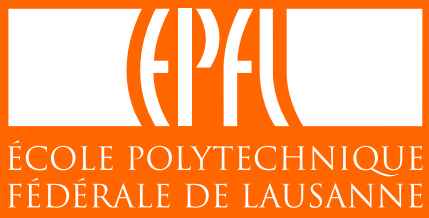

School of Computer and Communication Sciences IC

Computer Science Section

Master Thesis Project Report

Flok: Collaboratively solve problems through participatory design thinking

Author: David Sandoz

EPFL Supervisor:
Prof. Denis Gillet
Coordination & Interaction Systems
Group REACT

Company Supervisor:
Bastiaan van Rooden
Nothing GmbH

March 2016

Acknowledgments

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Contents

1 Introduction

Humans have ideas. Not a lot of those ideas end up being applied, no matter if they are good or bad. Sometimes they just stay in the head of the person who had one and are not developed further because the person thinks it is not a good idea. She might be right but she can't really know as long as she has not shared her idea. And of course it happens that people do share their ideas. That is something good to do because it can bring a lot of valuable input that we do not necessarily think about by ourselves. This makes the idea evolve; it might go in one direction or another, change shape, or even generate new different ideas. This can also be seen as what is called *brainstorming*. This process is in general quite messy. A lot of information is generated and not structured, which makes it difficult to highlight the most important items. For brainstorming, teams sometimes use a ticketing system that they already use for other projects related tasks. Tickets are great for development, but not good for creative brainstorming.

Therefore, what we want to achieve is to design and develop a platform that significantly improves collaboration around ideas within a team or a small to medium-sized company by getting considerably close to the cognitive working reality of a team. We want to have a more human experience. This will enable the users to have an effective way to bubble up the good ideas among all the information, and also to drive the sharing of new ideas. All this should be highly intuitive and straightforward to use, by being particularly careful about the overall user experience of the platform.

1.1 Flok

Nothing Interactive developed an internal web platform called *Flok*. It was also about improving collaboration within a team or a medium-sized company, but rather by providing various components such as a "to do" app, a time tracker or a global activity stream to which events can be aggregated from external services. However, Flok has been rescoped to match this Master thesis goals. What stays, in addition of the name, is mostly the general idea of collaboration and respect of the human behavior. The original Flok is still accessible on GitHub[1].

1.2 Hypothesis

It can be proven that a truly real-time approach to create, read and update information within on-site or remote, (inter-)disciplinary teams significantly improves their shared know-how and overall collaborative spirit thus leading to a verifiable increase of their creative potential.

1.3 User-centered design

The approach taken to create the platform is based on the *user-centered design* concept. The goal is to focus first on the user need and to start by designing the user interaction with the product to then define what the content is going to be and which technologies are going to be used. The reason why we took this approach is because we really want the product to be intuitive for the end-users, that it matches their expectations regarding what they need, what they can do with the platform, rather than making them adapt their behavior.

[1] `https://github.com/nothinginteractive/flok`

To this end, different processes were used, such as *User Story Mapping* to define the user needs, *Wireframing* and *Prototyping* to quickly test if the design of a functionality matches those user needs, and *User testing* to have feedback from real users in order to adapt the platform to their expectations. Moreover, we are not going through these different processes sequentially, but rather iteratively. Each of these steps enable us to discover new issues, new opportunities and we have then to reflect those in every step.

2 Personas

In order to embrace the user-centered design concept, we have to put ourselves in the shoes of the users we expect to use the platform. To do this, *personas* were created. They are fictional characters build up from the ground who represent the different type of users that we might have. We made three of them for the project. All three work in the same startup. *Andrew McAllister* is the CEO, *Melanie Carter* a developer, and *Sergei Fleming* an interaction designer. These personas were not defined in much more details, as part of the research was to determine more clearly for which purpose Flok is going to be used.

3 User Story Mapping

User Story Mapping (USM) is a tool which help teams developing software to stay focused on users and their needs [1]. It is based on user stories and story maps. *User stories* are descriptions of how users are interacting with the whole product and not only with one of its feature. *Story maps* are a two-dimensional visual representation of stories with *cards* as atomic parts. In general, the top row of cards represents the backbone of the story (from left to right), and the cards below give more details. In addition to the focus it gives on the users, USM enables the discussion within the team who builds it to create a shared understanding of the product. User story maps can be done with software tools which make it easier to edit and share. However, team collaboration is enhanced when people are facing a physical user story map made of sticky notes, which is what has been done for this project.

The user story map constantly evolves throughout the development of the project. In figure 1 you have an overview of how it evolved for Flok. Figure 2 shows you its state at the time of handing in this report.

The green sticky notes represent actions by users and the blue ones indicate which user are doing the actions. In the latest versions of the user story map, we can notice that the orange sticky notes entitle *slices*, of the story. As said, the first row is the backbone. Below we have three slices R1, R2 and R3, *R* meaning *release*. It helps to define clearly which part of the story are the most important and therefore need to be possible to do for the user in the earliest versions of Flok.

Building this user story map was a bit more tricky than it can be for most others. Indeed, as Flok interest resides in the real-time interactions between its users, the story has to jump often from a user to another. This makes it also more difficult to follow when reading the story map.

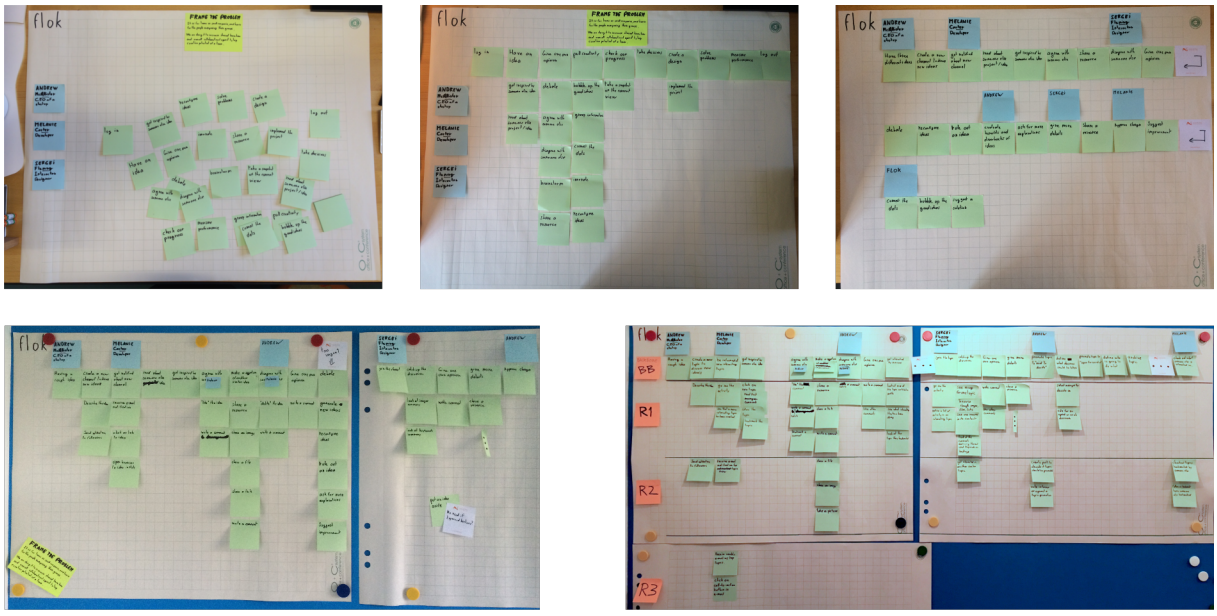

Figure 1: Evolution of the user story map for Flok

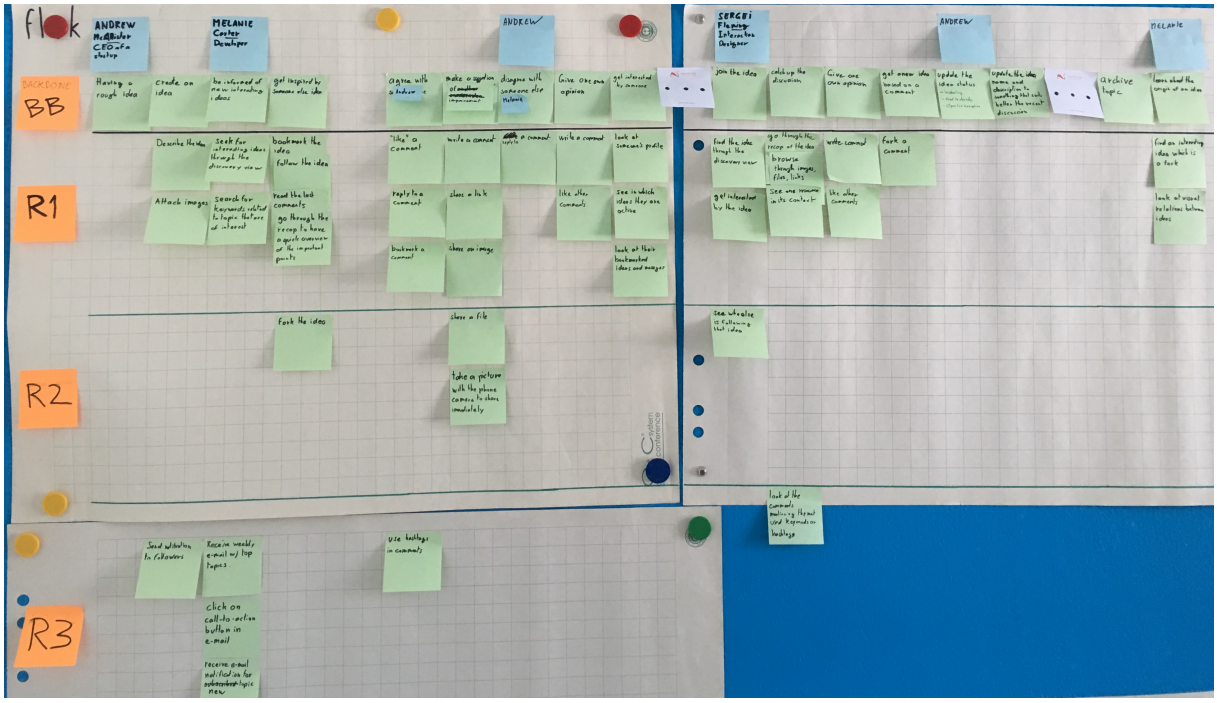

Figure 2: Current state of the user story map for Flok

Story description

The story simply start with Andrew having a rough idea that he wants to share in order to discuss about it and develop it. He then adds the idea on Flok and describe it with text, but images as well. From here, other users can take part to the idea. For instance, Melanie is looking for interesting ideas in the *discovery* section of the app and she get inspired by the one from Andrew. From here, she has several possibilities. She can simply bookmark the idea in order to have it saved and hence, easily come back to it later. She can also follow the idea in order to get informed of the activity occurring within it. Then, as she is interested, she goes

through a page which recaps the discussion around the idea. This helps her to quickly have a good overview. If she wants to actively take part to the idea, she can go to the discussion section and read the last comments before interacting. As it happens, she agrees with Andrew's comment and show it by *liking* it and replying directly to it. Moreover she wants to find that comment later so she bookmarks it. Then she gives her input by writing comments and sharing documents and images. However, Andrew disagrees with the input from Melanie and he says it by replying to the corresponding comment.

Among the comments and other inputs of team mates taking part to Andrew's idea, he gets interested by the views of one specific person. He looks at this person profile and more precisely in which other ideas she is active and that might interest him.

Back to Andrew's idea, after a while, Sergei joins the idea that he found through the *discovery* section of the app. He found the idea interesting and noticed that among the followers there were team mates he usually like their thinking. He catches up the discussion through the recap and among the highlighted items one image appealed him. He brings up the image in its context to see in more details what it is about. This makes him think about another idea so instead of replying at this point of the discussion, he rather decides to fork from the image to create a new idea in Flok.

Initially new ideas are classified under the *Incubating* label. This is not the case anymore for Andrew's idea which has evolved and became more mature and precise. Hence he decides to update the classification to *Need to decide* where the discussion should be more about what are going to be the next step to implement the idea. Moreover, the original name and description of the idea do not suit its content anymore. Therefore Andrew updates them accordingly.

After a while, the idea reached the *Open for execution* classification where it got successfully implemented. At this point the idea can be archived, which is what Andrew does.

4 Information architecture

To accompany and solidify the project it is important to have a good information architecture. This helps to have a well defined vocabulary for the different elements making the product, and how they interact with each other. In our case, we made an *entity-relationship diagram* that you can see in figure 3. In the rectangle are the different entities in Flok. The ovals their attributes and the diamonds are actions representing the relation between entities. We have to read the relations from top to bottom (e.g. *Persons are actor of Activity items* or *Ideas contain Messages*). The thickness of the lines and the fact they are an arrow or not also has a specific meaning, described in the bottom right of the figure. For instance, a person can post none or more messages, but a specific message is posted by exactly one person. Also, a message must be either an image, a file, a system information or a comment, and any of these four entities has to be a message.

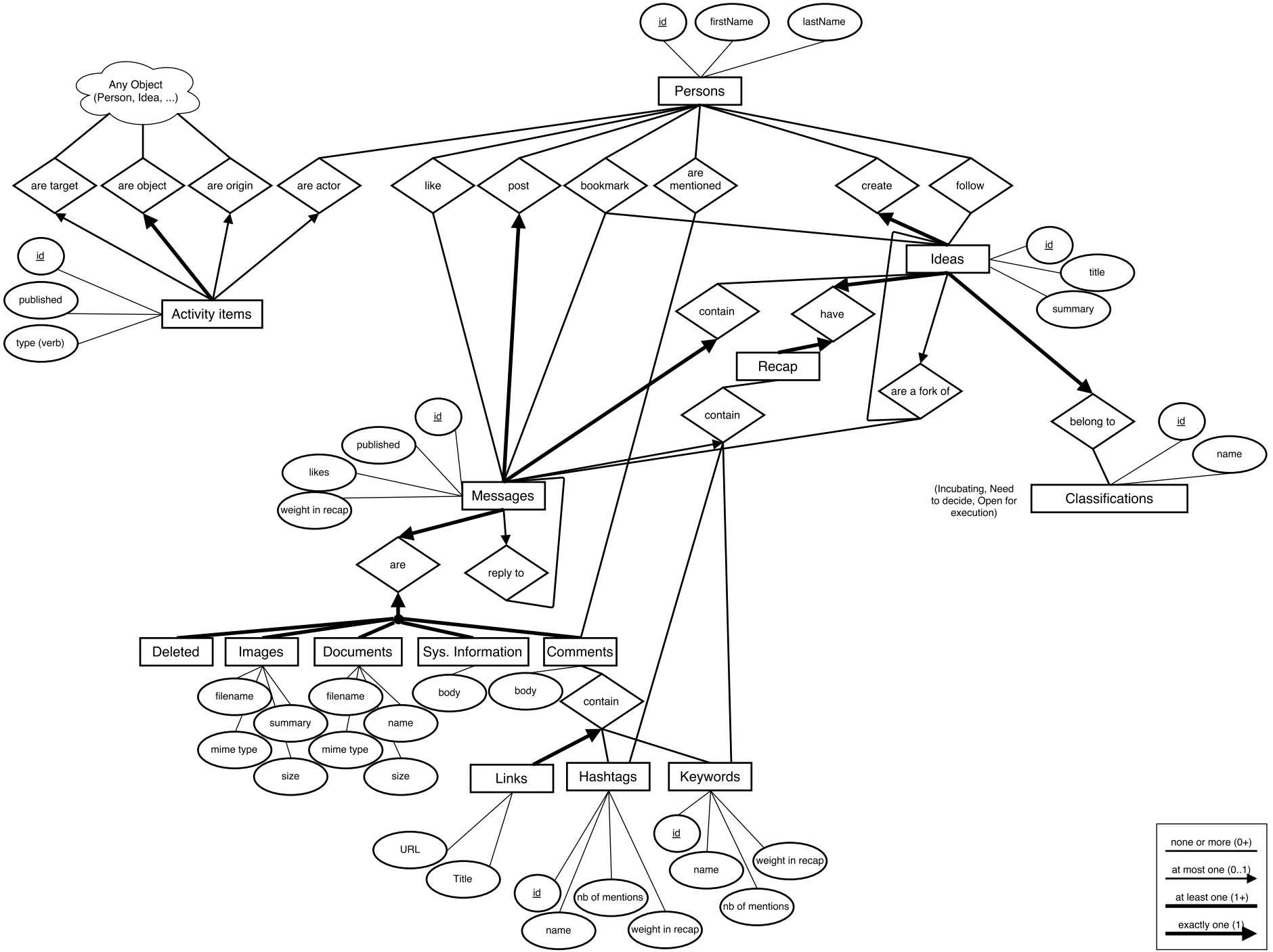

Figure 3: The entity-relationship diagram of Flok, representing its information architecture

5 Wireframing

Once we had a first version of the user story map, the next step was to build wireframes of the user interface. Wireframes allow to quickly have a very rough view of the components layout and how they fit in the available space. They enable us to see changes that need to be brought even before we start designing or implementing, and hence save us some precious time.

For instance, initially ideas were called *topics* and when I designed the first wireframes of recap page for a topic, it was a kind of tag cloud with all the ideas discussed in a topic – The most discussed ones being more prominent (see figure 4). This made us realize two things. First, there was a lack of shared understanding regarding the scope of the discussions we expect to be held in one topic and therefore also regarding the content of the recap page. Secondly, the name *topic* was not a good name to describe this concept. This is what made us change the name for *idea*. It is also more human – Instead of creating a topic on Flok when you have an idea, you just add an idea on Flok when you have one. This shows that we were able avoid having to change a single line of code to apply this change as it was detected before we start the first prototype.

Figure 4: Original wireframe design of the recap page. From left to right: the main recap page, the recap page of one idea in the topic, the dialog asking if we want to see an item in its context, an item highlighted in its context.

You can notice in the wireframes of figure 4 that we choose to represent the app in a mobile screen. This decision has been made because having less space forces us to think about what is the most important to show to the user. It is then easier to design a desktop version from the mobile version than the other way around. The *Mobile First* development is also getting more important as more people are accessing the web from their mobile device before doing so from a desktop computer.

A few other design decisions were taken during wireframing. For instance, it was initially possible to unlike a message in a discussion. This has been removed to only keep the like button. Indeed, it is not necessary to explicitly express a disagreement with this action which does not bring any constructive feedback and we also want to foster positivity. A disagreement can still be express with a comment which should be more constructive.

This is also while designing the wireframes that we asked ourselves the question regarding the order the messages in a discussion. Some researches [2] concluded that having the oldest ones at the top and the more recent ones at the bottom was leading to more engagement from the participant of a discussion. Following this order is also more human as it follows the natural reading order. Moreover, as the input field to add messages is at the bottom, having the new messages appearing right above it is also expected.

6 Prototyping

The step following wireframing is prototyping. In this case, the initial goal was to make it mostly visual in order to have a good idea of the look and feel, but also of the possible interactions from the users. The content would consist of static mock data. To avoid having to create a full visual design, we decided to use the component library *Material Design Lite*[2], which is an implementation of *Material Design*[3]. This lightweight library, with the use of some HTML, CSS and simple Javascript, enabled us to quickly have a prototype with a solid visual design (see figure 5).

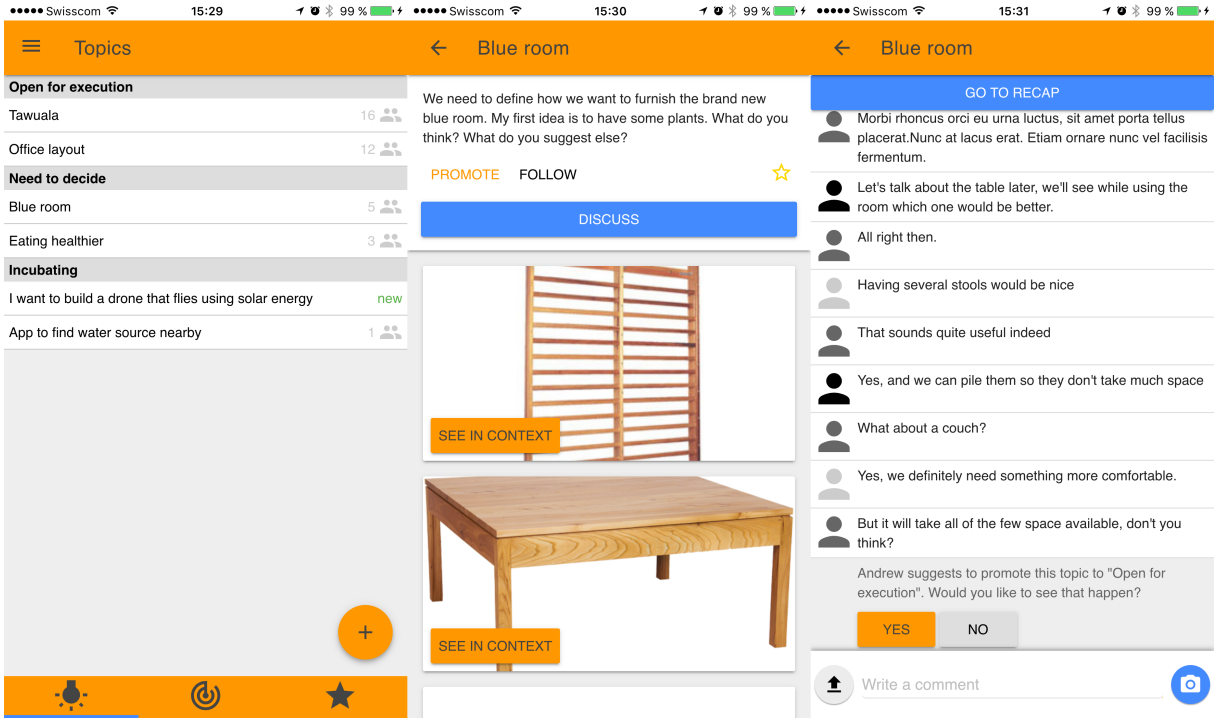

Figure 5: Screenshots of Flok first prototype using *Material Design Lite*

Improvements and features have been progressively added to the prototype. With, all the necessary code to keep it interactive. However, as it was initially planned to be light, having a bigger code base was not adapted with the prototype structure making it quite messy and therefore more difficult to add new components. The visual result being good, it seemed to be the right moment to start working on the functional front-end of the platform, based on the prototype.

[2] http://getmdl.io
[3] http://google.com/design/spec

7 Front-end

7.1 Architecture

We wanted the platform to be fully usable from any device. The easiest way to implement that is to have a responsive web application. Hence, Javascript and the *MEAN stack* seemed quite appropriate. These are also technologies in which *Nothing Interactive* has good experience with. This means that regarding the front-end we went for a single page application made with *AngularJS*.

As we were satisfied with the result of using *Material Design* for the prototype, we wanted to continue following those guidelines. Obviously, *Material Design Lite* was not enough, therefore we had to use another implementation. Looking at the different alternatives and knowing that we were going to use *AngularJS*, the decision to use *Angular Material*[4] was straightforward.

There are different ways to build an *AngularJS* application. Therefore it was important that we clearly define how we were going to structure it. *Nothing Interactive* has some internal guidelines, but with the arrival of *Angular 2*, it seemed to be a good time to re-think those guidelines in order to write code that would enable an easy transition to this disruptive new version of the framework. Hence, we made some adaptations based an Angular Style Guide written by John Papa[5], which is endorsed by the Angular team.

Unfortunately, our great focus on building the front-end and having the best user experience possible did not gave us time to build a functional back-end for the app. At some point we had to take the decision between either building the back-end, or doing the user testing. The latter seemed more important as it would (in)validate our assumptions regarding the interaction of the user with the platform. We also estimated that it would bring us valuable feedback to improve the front-end.

Not having a back-end implies some restrictions. First, it means that once the Angular app is loaded in the browser, everything is happening in the browser only and there is no communication with the server, except to get static files such as images. Therefore, interaction between two users is not possible. Then, no back-end also means no database and no saved data. Fortunately, as the front-end is a single page application, as long as the page is not fully refreshed, all the data created by the user while using Flok is saved in memory, including uploaded files (see figure 6). Therefore the user can still use and evolve in the app.

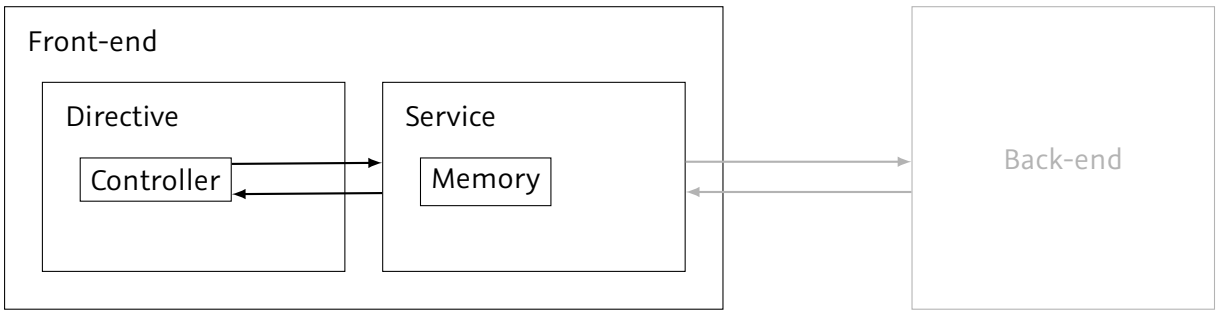

Figure 6: Schema illustrating data flow for Flok. In the front-end, the controller of directives ask for data to services. Those service should ask the back-end for the information stored in the database. But as there's no back-end, services get the information from their own memory.

[4] https://material.angularjs.org

[5] https://github.com/johnpapa/angular-styleguide

We were not trying to build a fully working application without backend. Therefore we still kept in mind an architecture that would communicate with a back-end by simulating calls to it, even though we just get the data from memory.

7.2 Implementation and design decisions

We are not going to describe the implementation of every component as some of them are quite straightforward. We are rather going to explain the components where some reflexion had to be made or decisions had to be taken.

7.2.1 Ideas

Ideas are the main components of Flok. They are created by users and they have three different views. The main view gives the general information about the idea: author, creation date, description, classification (section 7.2.2), followers and relations with other ideas. From here you can also trigger actions on the idea such as following, bookmarking, archiving and forking (section 7.2.4). The available actions depend of the status of the user (creator, follower) toward the idea. Then there is the discussion view where all the messages (section 7.2.3) related to the idea can be found. This is also where messages can be added to the discussion. Finally there is the recap view (section 7.2.5) which highlight the most important messages of the discussion.

7.2.2 Ideas classification

Here there is actually not much to say regarding the implementation itself. We introduced these classification in order to easily visualize the state of an idea. Initially, to update the classification, it has either to be promoted or demoted, respectively to the next or previous classification. To do so, one of the idea follower had to start a poll asking all the followers to vote for or against and change of classification (see figure 7). The goal here was to reflect a common decision.

Then we realized the process can actually get complicated. What happens if not all followers vote? Should we wait? Should we define a time limit to vote? Should such a time limit be define by the poll initiator? What happens if there is a draw? We thought about how to reply to these questions and we could have implemented solutions, but in the end this might be a hassle for the user to make that change of state. Therefore we went for a drastic simplification. No more poll, no more promotion or demotion. We simply let any follower update the classification to any available (see figure 8). The change is instantly applied. Now there might be disagreement within the followers, but they are all in the same team and they can discuss that. And as it is so simple to change the classification, they can easily reversed their decisions. Having simplified this action also enabled us to implement an interactive way for the user to update the classification from desktop, by drag and dropping ideas in *Kanban*[6] columns (see figure 9).

[6] `https://en.wikipedia.org/wiki/Kanban`

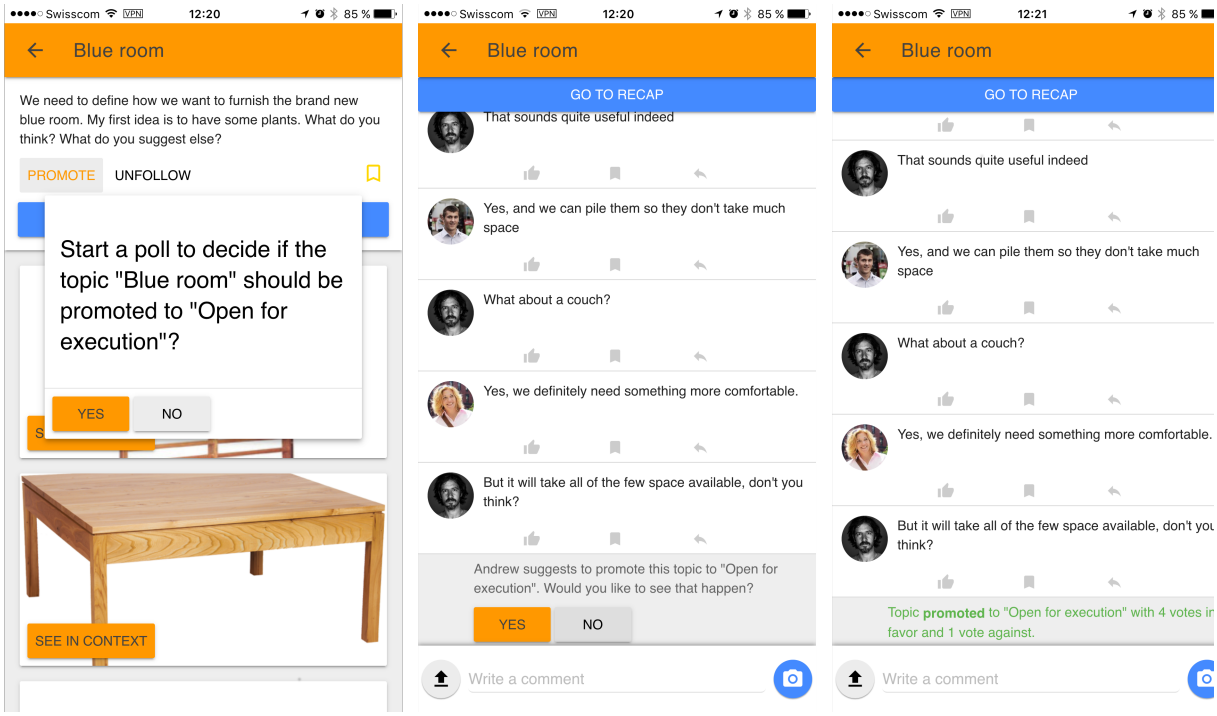

Figure 7: Screenshots of Flok prototype featuring the promotion poll

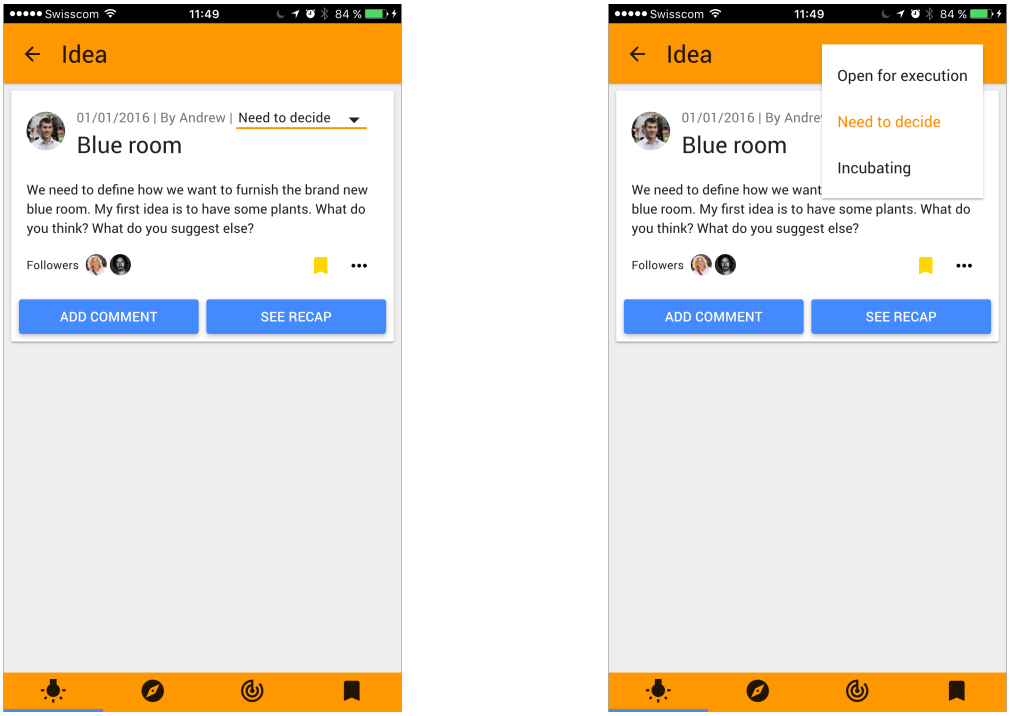

Figure 8: Screenshots of Flok front-end featuring the simple classification update of an idea

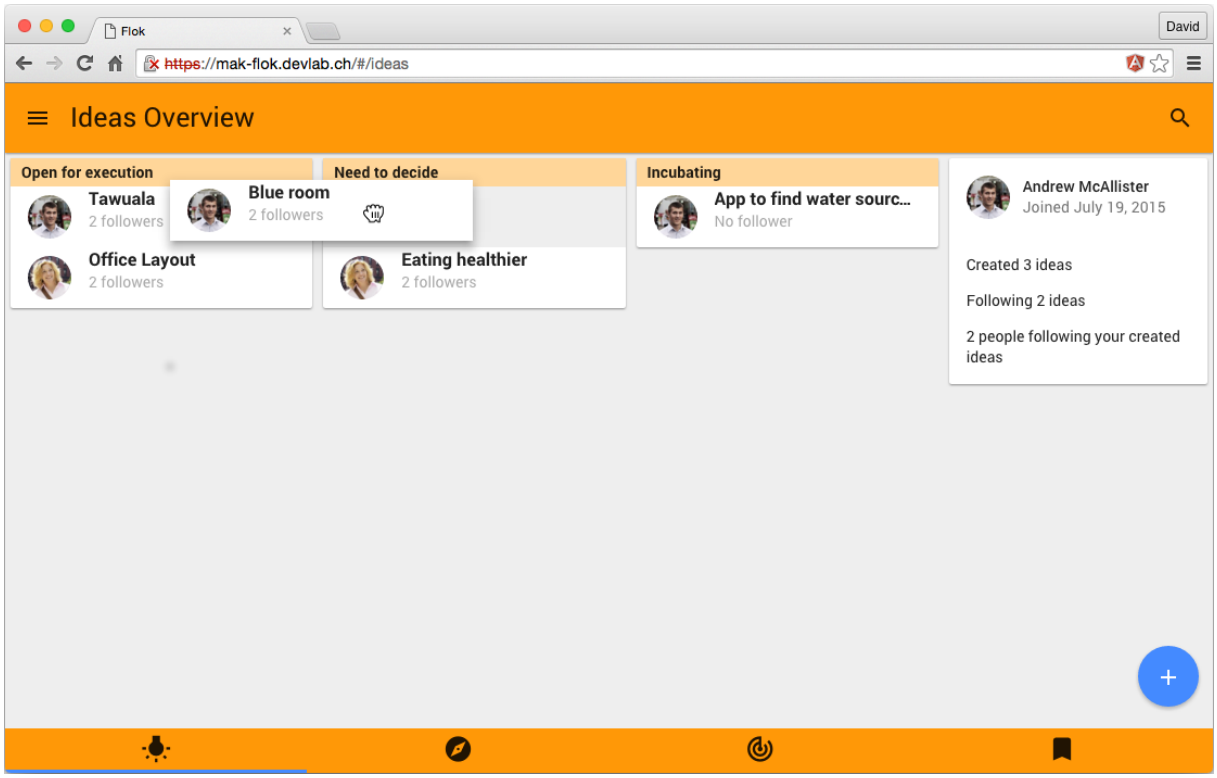

Figure 9: Screenshots of Flok front-end featuring the drag and drop of an idea from a classification to another

7.2.3 Messages

Messages are one of the central components of Flok. They have one author, they are part of one idea and they are either a comment, an image, a document, a system information or a deleted message. When adding a message to a discussion, it can be the reply to another message and thus help to establish relations among messages (see figure 10). For more details on the properties of each of these type and on the relations of messages with other entities of Flok, we send you back to the entity-relationship diagram in figure 3.

When we introduced the possibility to delete a message, it was really completely deleted. However, it also broke relations between messages and ideas, which are made through replies and forks. Therefore we decided to add the *deleted* message type, where the content of the message is deleted, but the object itself not. Like this the relations are kept.

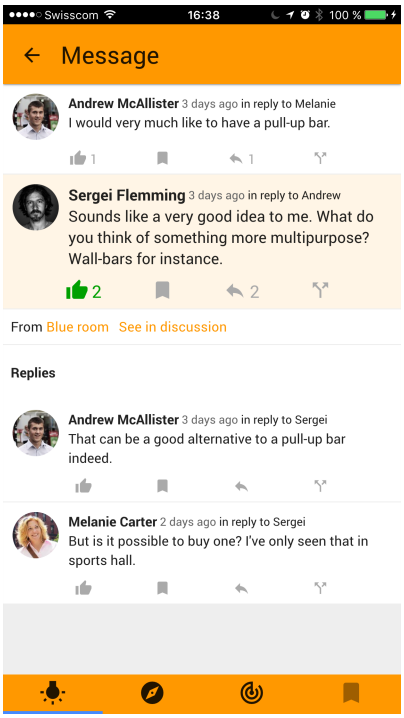

Figure 10: Screenshot of Flok front-end featuring the message view. The message is highlighted, above we have the message it is replying to and below we have the replies.

7.2.4 Fork

If we try to look at how humans think, they initially have an idea that they brainstorm by themselves or with other people. It then generates new ideas based on the brainstorm and it can go on until an idea really gets developed and executed. We tried to represent that in Flok with the *fork* feature. Basically, you can create an idea by forking another idea or a message. The word comes from the software development world, when a copy of source code is made to start developing on it independently. However it works a bit differently in Flok. The main goal is to create the relation between the initial idea or message to the newly created idea. Nothing is automatically taken from the source which means that the new idea form is blank and the user still has to describe her idea. Once the idea is created, the discussion can begin, as any new idea. The difference lies in the relation created, which goals is to represent the human way of thinking. We wanted to make these relations more visible and therefore implemented

visualizations showing forks relations (see figure 11). Clicking on the fork symbol in the Ideas Overview will bring to the fork navigation. We can also get there by clicking on the number of forks from the view of an idea.

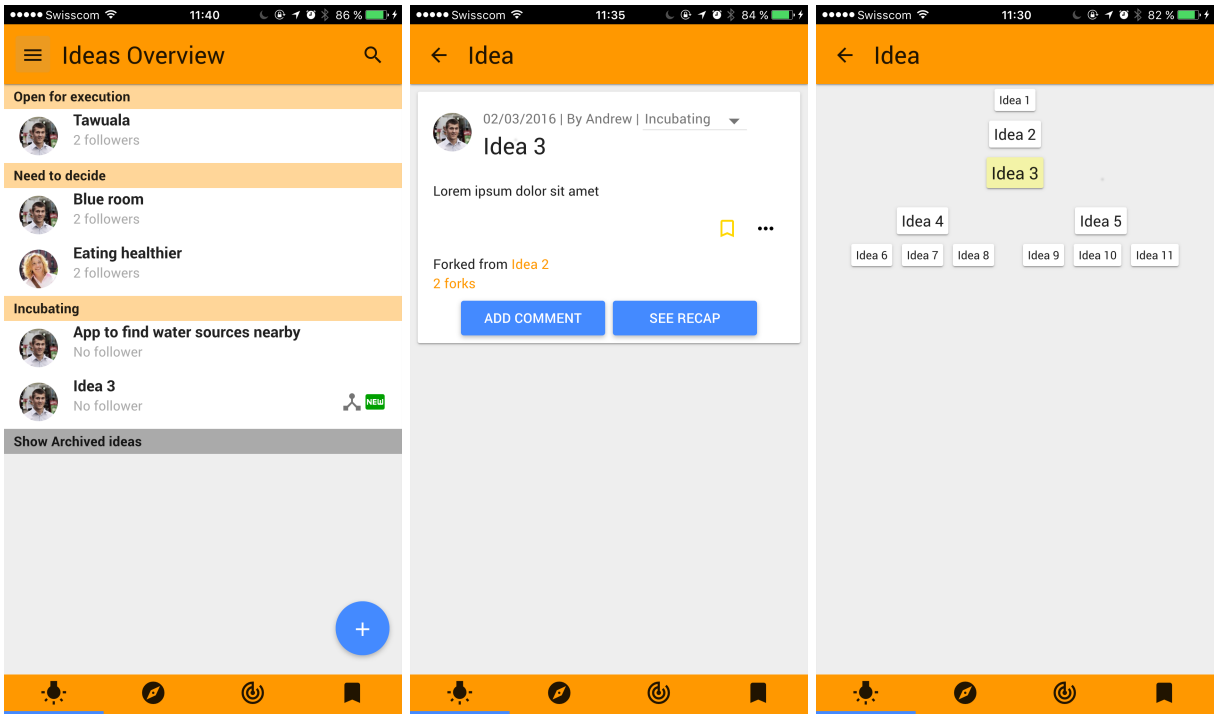

Figure 11: Screenshots of Flok front-end featuring forks visualization. On the left, the ideas that are forks or that have been forked are noticeable by a specific symbol symbol on the right. In the middle, it is indicated from which idea an idea has been forked from, as well as how many forks it is the source of. On the right, the forks navigation shows up to two parents or children fork level.

7.2.5 Recap

We mentioned earlier the fact that brainstorming is a process which can easily get messy, which is normal and natural. As we try to be close to the human way of doing, discussions in Flok give the same liberty, which can bring the same messiness. However, this liberty let the user express her thoughts and that's why we want to keep it. Nevertheless, we want to highlight the most important points of the discussion, bubble up the good ideas. This is what the recap do. And to do so, it runs an algorithm which returns the most relevant messages and order them by importance.

Such an algorithm should obviously run in the back-end of the system. But as explained in section 7.1, we unfortunately don't have any back-end. That's why we implemented the recap algorithm in the front-end.

Initially we wanted to take into consideration the text content of the messages and use some topic modeling to discover the different main topics discussed. The *Latent Dirichlet Allocation* [3] model seemed to be a good one to use for the algorithm. Nonetheless, as we are running the algorithm in the front-end, we went for something simpler. We give a weight to each message of the idea's discussion based on the following properties, to which we give more or less impact.

- number of replies r
- number of bookmarks b
- number of likes l
- number of forks f
- is an image $i = 1|0$
- is a document $d = 1|0$
- text length t

The weight w of message is then computed as follows.

$$w = 4r + 3b + 2l + f + i + d + \frac{t}{500} \tag{1}$$

The minimum weight for a message to be included in the recap is 1. This means that all images or documents are by default included. We are doing so in order to very rapidly have some content in the recap when an idea is created. As the messages in the recap are ordered by weight, if images or documents don't get more weight, they will simply stay at the bottom.

You will notice in equation 1 that some of the message properties are weighted with different factors. Replies have the most importance with a factor 4. Indeed, replying to a message means that this message generated engagement and new content in the idea. Therefore it is certainly good to have it in the recap. If a user bookmark a message, it means he want to have it saved for himself. This shows a strong interest for the message, thus this being the second most important property with a factor 3. When users bookmark a message, it affects their bookmark list and they will therefore be careful regarding what they bookmark. The likes however, are only affecting the message which receives them and not the users who give them. Hence the reason we give them less importance than bookmarks. They still have a factor 2 as they are used to show support or agreement. Forking a message means creating a new idea. Therefore, we could say that a message that has been forked is of great importance. It certainly is for the new created idea, but for the idea containing this message, it does not necessarily need to be highlighted in the recap. If it does, then it would get replies, bookmarks or likes that are going to make it appear there. This is why forks only have a factor 1 in the computation of a message weight. Then, we are adding 1 to the message weight if it is a document or an image. As these types of messages are more visual or bring more content than a simple comment, we decided they deserve from the beginning to have the minimum weight to appear in the recap. Finally, the text length of a comment also influence the weight of the message. If it has enough content, i.e. 500 characters, then it will appear in the recap. And the more it has, the bigger its weight will be, the higher it will be in the recap.

Of course, for this simple algorithm, these factors are based on assumptions and only testing in a real-life situation would show us if the result is matching users expectations.

7.2.6 Activity

The activity section of Flok shows the user the recent events occurring in ideas she follows and not triggered herself (see figure 12). This allow her to have a quick overview of everything that happened in the ideas she is interested in. If she wants to see her own activity, it is visible within her profile. The actions that are logged in the activity are the following:

- creating an idea
- forking an idea
- following an idea
- moving an idea from a classification to another
- renaming an idea

- adding a message (comment, image or file)
- liking a message

This activity feature is something we often see in social applications and therefore, it has been implemented in various ways. In prevision of any future integration with other external platforms, we decided to follow as closely as possible the *W3C Activity Streams 2.0* standard [4]. Currently, it is still a working draft, and hence subject to change. The main efforts have been focused in respecting the properties of an *Activity* object and in using the *W3C Activity Vocabulary* [5] when applicable.

Activity can be logged from anywhere in the application by using the `log` method from `activityService`, an AngularJS factory. All the activity is stored in this service and we can either get the global activity using the `getActivity` method, or get the activity specific to one person using the `getPersonActivity` method.

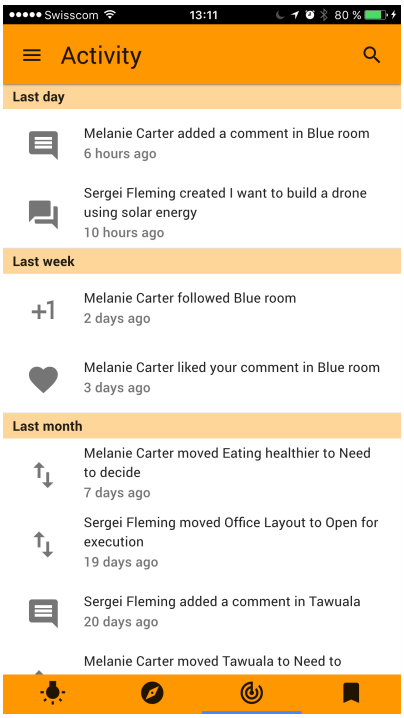

Figure 12: Screenshot of Flok front-end featuring the activity of the platform

8 User testing

9 Conclusion

References

[1] J. Patton and P. Economy, *User Story Mapping: Discover the Whole Story, Build the Right Product*. O'Reilly Media, Inc., 1st ed., 2014.

[2] A. Mabande, "Designing for dialogue-how the design of web commenting systems affect the conversation," Master's thesis, Malmö högskola/Centrum för teknikstudier, 2010.

[3] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *the Journal of machine Learning research*, vol. 3, pp. 993–1022, 2003.

[4] J. M. Snell and E. Prodromou, "Activity streams 2.0," W3C working draft, W3C, dec 2015. https://www.w3.org/TR/2015/WD-activitystreams-core-20151215/.

[5] J. M. Snell and E. Prodromou, "Activity vocabulary," W3C working draft, W3C, dec 2015. https://www.w3.org/TR/2015/WD-activitystreams-vocabulary-20151215/.