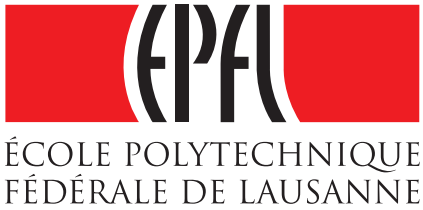


École Polytechnique Fédérale de Lausanne

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

I would like to thank first Bastiaan van Rooden aka Spot, my supervisor at Nothing Interactive, for his guidance and for the time he dedicated to be there for me from the beginning to the very end of my Master thesis project.

I would also like to thank Prof. Denis Gillet, my EPFL supervisor, for the various feedback he gave me throughout the course of the project and for taking the time to read all my fortnightly reports.

I am also very grateful to Tobias Leugger aka Vibes, Head of development at Nothing Interactive, who gave me precious advices on the technology aspects of the project.

Many thanks to the whole crew of Nothing Interactive: Cast, Cyclodex, Fitts, Iadu, Lang, Lex, Linz, Nodz, Pulse, Sense, Spot and Vibes. They may not all have been involved in my work, but they were part of the environment in which I was working for this project. The *rocket* was a very comfortable, productive and healthy environment which definitely helped me reach the end of this adventure.

Author/student's contact details
David Sandoz
Les Allées 29
2300 La Chaux-de-Fonds

contact@davidsandoz.ch

Abstract

Humans have ideas and they are generally improved when those are shared and discussed among several people. However, it can get quite messy and our goal is to design and develop a platform – Flok – that significantly improves collaboration around ideas, while providing an experience as human as possible. To do so, we took a user-centered approach by using processes such as the creation of personas, user story mapping, wireframing and prototyping. Based on those processes and a well defined information architecture, we built a front-end application. The back-end also has an important role in order to have a final fully working product. However, we let the experience drive the product, not vice versa. Therefore, to make a good experience, we prioritized user testing over implementing the back-end as the given time frame was not allowing us to do both. We were able to improve the front-end and verify some of our assumptions by doing three test sessions. The feedback we got is promising and tells us we are going in the right direction. Building the back-end is one of the important next steps that will allow us to have a fully working application. Combining that with more user testing will allow us to validate or invalidate further assumptions.

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. And we want this experience to be as human as possible. 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

Flok is the name of the platform we want to build. This name comes from the term *flock*, representing a large number of entities – people – moving together. It fits well as we really want to make use of the natural swarm intelligence of a group of people. We want to focus on the collaboration and how, together, they make ideas evolve, while respecting human behavior. The term *flock* can also be assimilated to a network effect, where the value of the service is influenced by its users, making it more attractive to other people.

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.

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 needs. 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 other 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 behavioural patterns 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 technique 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. For the Master project, we implemented the front-end up to R2 and left in R3 parts of the stories we did not deem necessary to have in an early version 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.

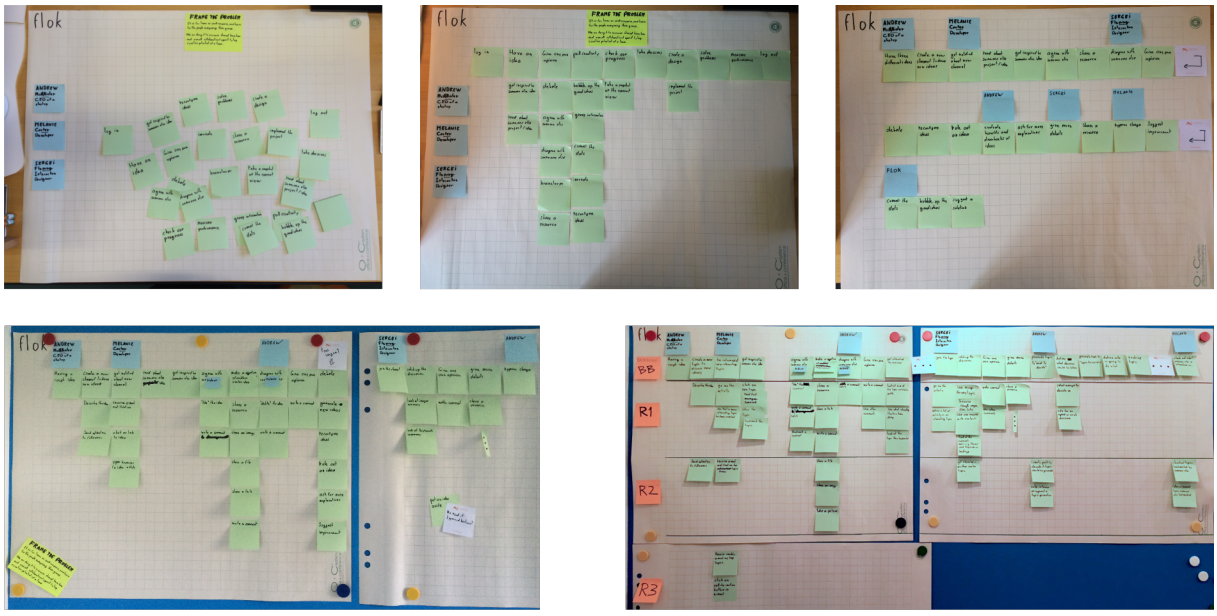


Figure 1: Evolution of the user story map for Flok

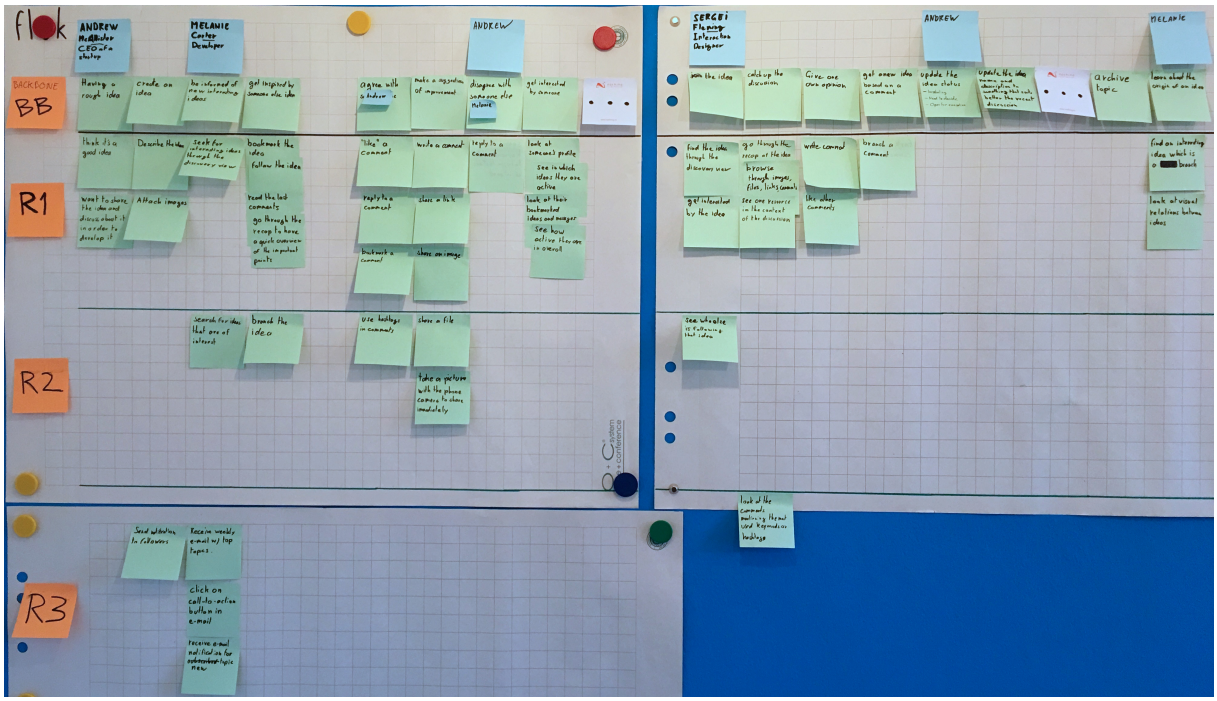


Figure 2: Current state of the user story map for Flok

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 a small text and images. From here, other users can take part to the idea. For instance, Melanie is looking for interesting ideas in the *discovery* view 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

the recap view, which highlight the most important elements of the disscussion. 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 Andrew's team mates taking part to his 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, in the discussion, 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 branch 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 state 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* state 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 are 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 Comments*). The thickness of the lines and the fact they are an arrow or not also has a specific meaning, described in the key, at bottom right of the figure. For instance, a person can post none or more comments, but a specific comment is posted by exactly one person. Also, a comment must be either an image, a file, a system information or a note (a text comment), and any of these four entities has to be a comment.

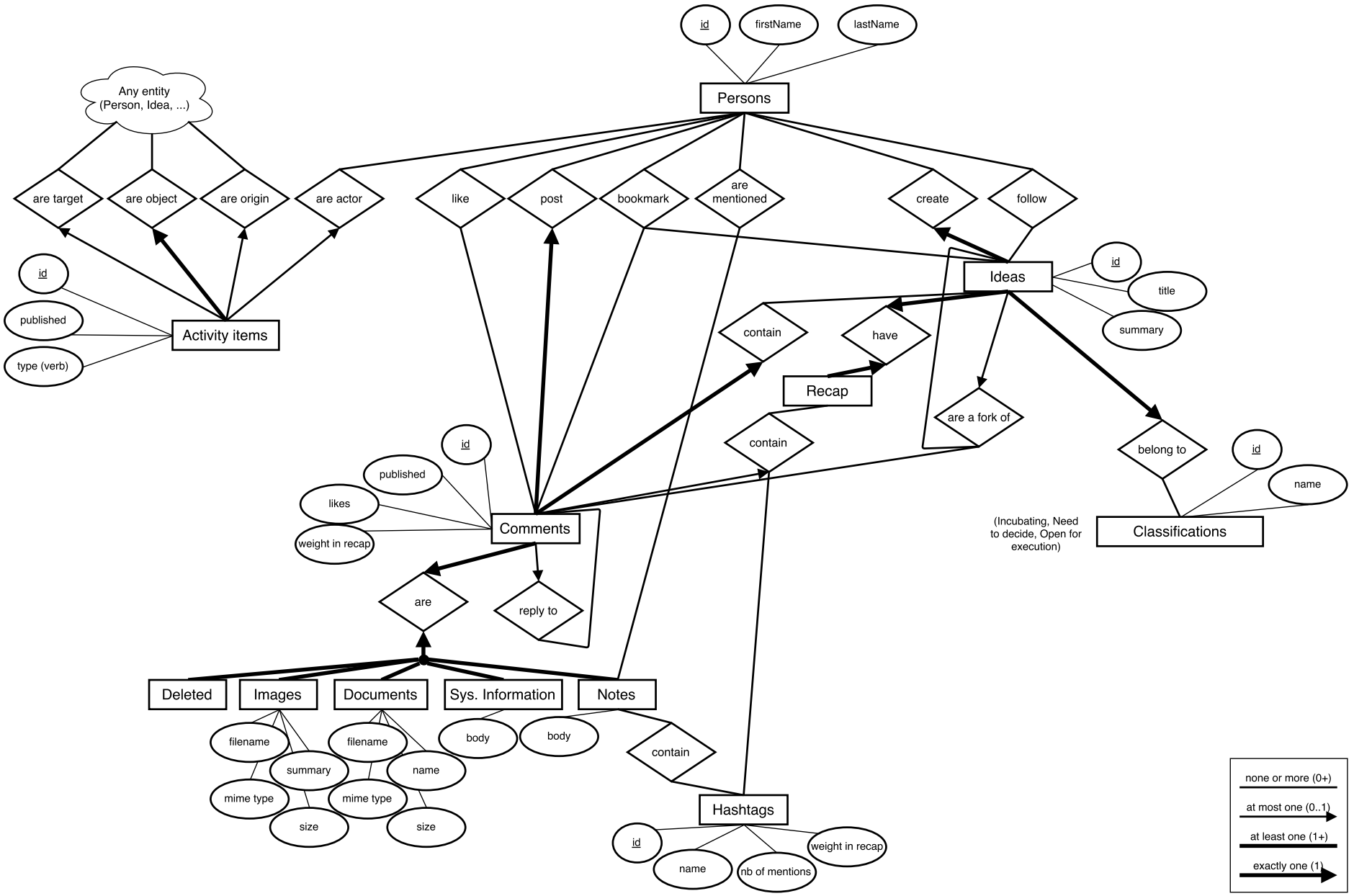


Figure 3: The entity-relationship diagram of Flok, representing its information architecture

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 we 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 the leftmost wireframe of figure 4). This made us realize that here was a lack of shared understanding regarding the scope of the discussions we expect to be held in one topic. The first wireframes showed a design where topics would contain several ideas, and each idea of a topic would have its recap page. We finally agreed that one topic should be one idea, and to even simplify that, we changed the name *topic* 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 comment 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 comments 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 comments is at the bottom, having the new comments appearing right above it is also expected by the user.

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*[1], which is an implementation of *Material Design*[2], a well established design language developed by Google. 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).

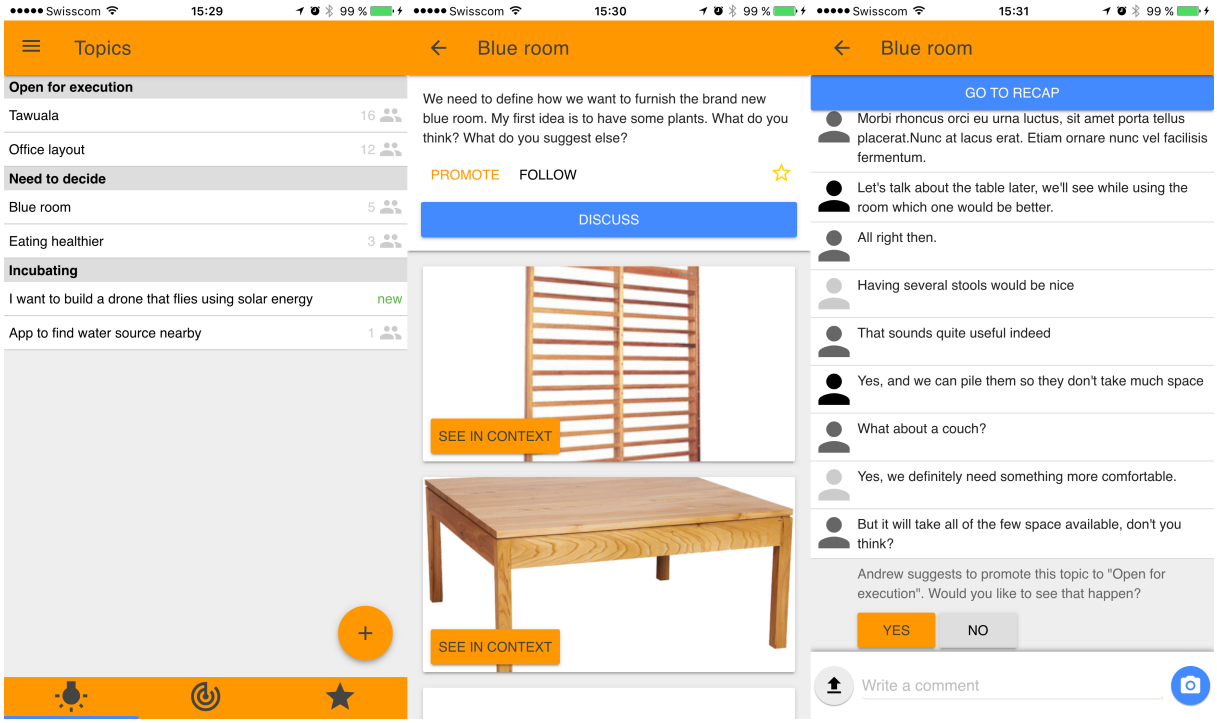


Figure 5: Screenshots of Flok first prototype using *Material Design Lite*

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

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

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.

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, using Javascript seemed quite appropriate. A lot of frameworks and libraries exist and among all of those, for the front-end, we chose to build a single page application with AngularJS[3]. There are several reasons to this decision. First, it is one of the most known Javascript framework, well documented and with a huge community. It allows a good interactivity of the end user which is a good fit with the nature of the application we want to build. Moreover, I personally already have a good experience with it, and Nothing Interactive as well.

As we were satisfied with the result of using Material Design for the prototype, we wanted to keep following those guidelines. Material Design Lite being too light, it was not enough. Looking at the different alternatives implementations 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. The nature of those adaptations is mostly to have a component-based application architecture where we use Angular directives as components.

Regarding the back-end, we planed to use Node.js[6] with Express[7] in order to have Javascript for the full stack, which is a common and good practice. However, we let the experience drive the product, not vice versa. Therefore, to make a good experience we prioritized user testing over implementing the back-end as the given time frame was not allowing us to do both. User testing also seemed more important as it would validate or invalidate our assumptions regarding the interaction of the user with the platform. We estimated that it would bring us valuable feedback to improve the front-end. Moreover, it is an established best practice to not having a back-end in order to ease the iterative process of updating the front-end.

Nevertheless, it also implies some restrictions. First, it means that once the AngularJS 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

[3] `https://angularjs.org`
[4] `https://material.angularjs.org`
[5] `https://github.com/johnpapa/angular-styleguide`
[6] `https://nodejs.org`
[7] `http://expressjs.com`

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. Hence the user can still use and evolve in the app. In figure 6, a schema illustrate the data flow, where directives, controllers and service are concepts coming from AngularJS.

We were not trying to build a fully working application without back-end. 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.

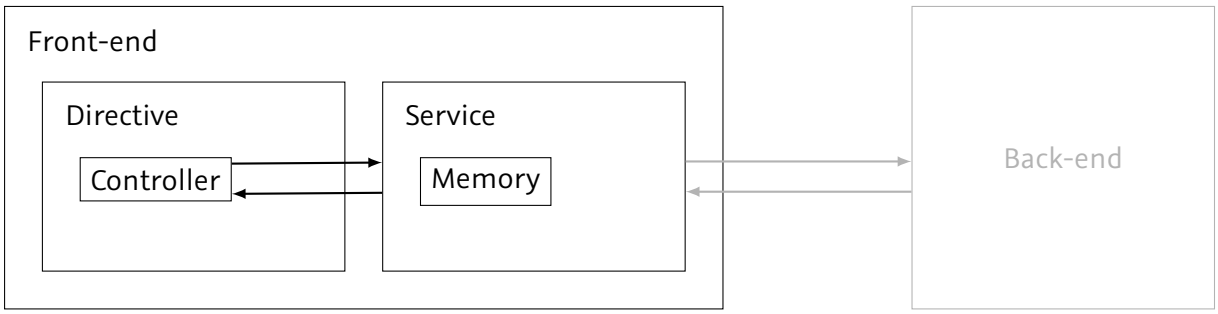


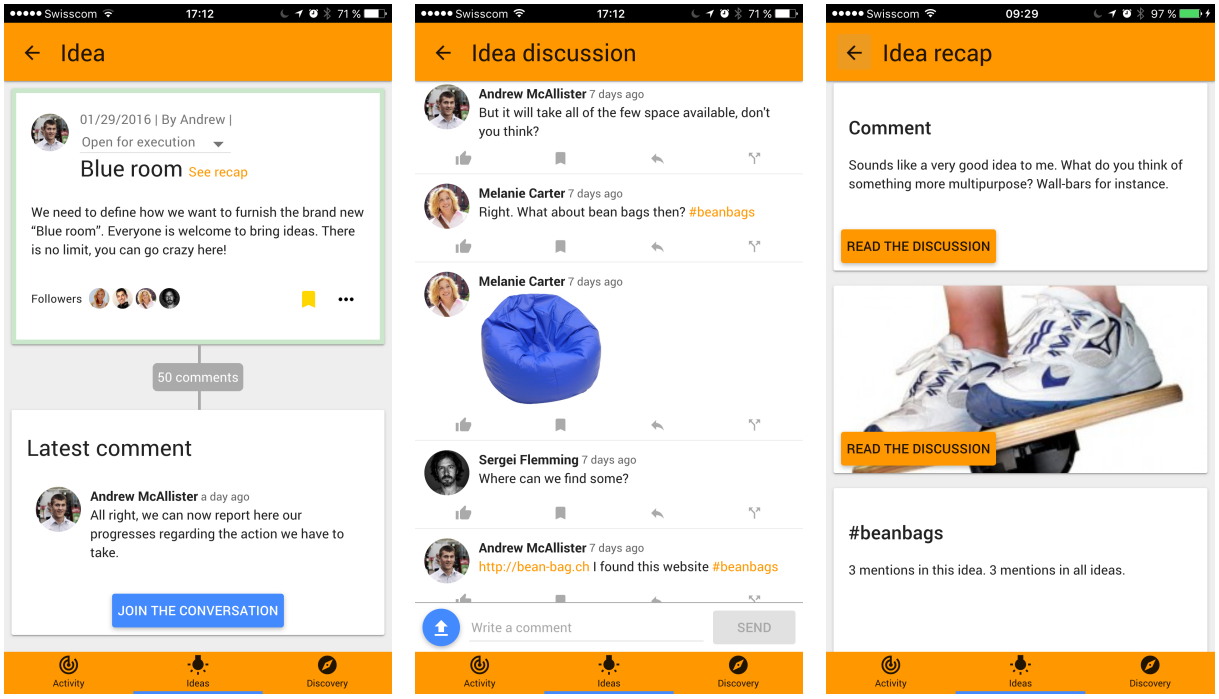
Figure 6: Schema illustrating data flow for Flok. In the front-end, the controllers of directives ask for data to services. Those services should ask the back-end for the information stored in the database. But as there is no back-end, services get the information from their own memory.

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 reflection 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 (figure 7a) gives the general information about the idea: author, creation date, description, number of comments, state (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 branching (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 (figure 7b) where all the comments (section 7.2.3) related to the idea can be found. This is also where comments can be added to the discussion. Finally there is the recap view (figure 7c and section 7.2.5) which highlight the most important comments of the discussion. The user flow to add an idea is illustrated in appendix A.1 and the one to discover and follow an idea in appendix A.5.



(a) Main view (b) Discussion view (c) Recap view

Figure 7: Screenshots of Flok front-end featuring the different views of an idea

7.2.2 Ideas state

Here there is actually not much to say regarding the implementation itself. We introduced these states in order to easily visualize the development situation of an idea. Initially, to update the state, it has either to be promoted or demoted, respectively to the next or previous state. 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 state (see figure 8). 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. Instead, we went for a drastic simplification. No more poll, no more promotion or demotion. We simply let any follower update the state from a dropdown menu (see figure 9). The change is instantly applied. Now there might be disagreement within the followers regarding the state an idea should have, but they are all in the same team and they can discuss that. Moreover as it is so simple to change the state, they can easily reversed their decisions. Having simplified this action also enabled us to implement an interactive way for the user to update the state from desktop, by dragging-and-dropping ideas in *Kanban*[8] columns (see figure 10). You will also notice that these columns are color-coded in order to make it even more clearer for the users to understand what can be executed.

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

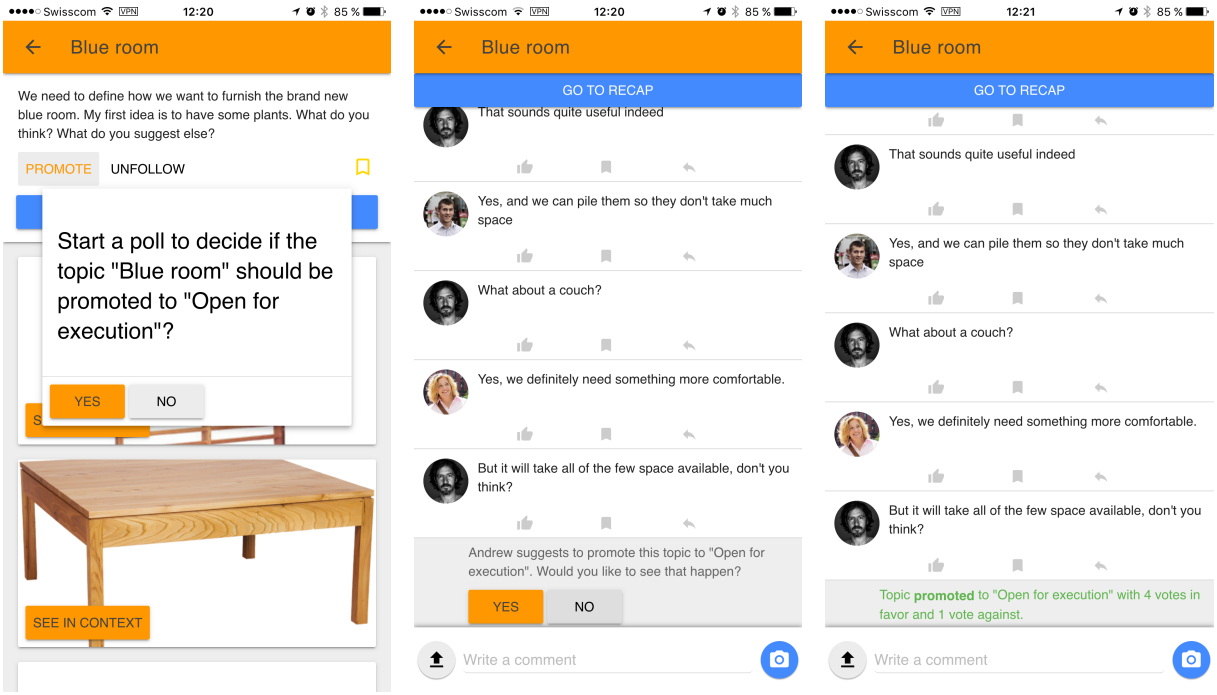


Figure 8: Screenshots of Flok prototype featuring the promotion poll

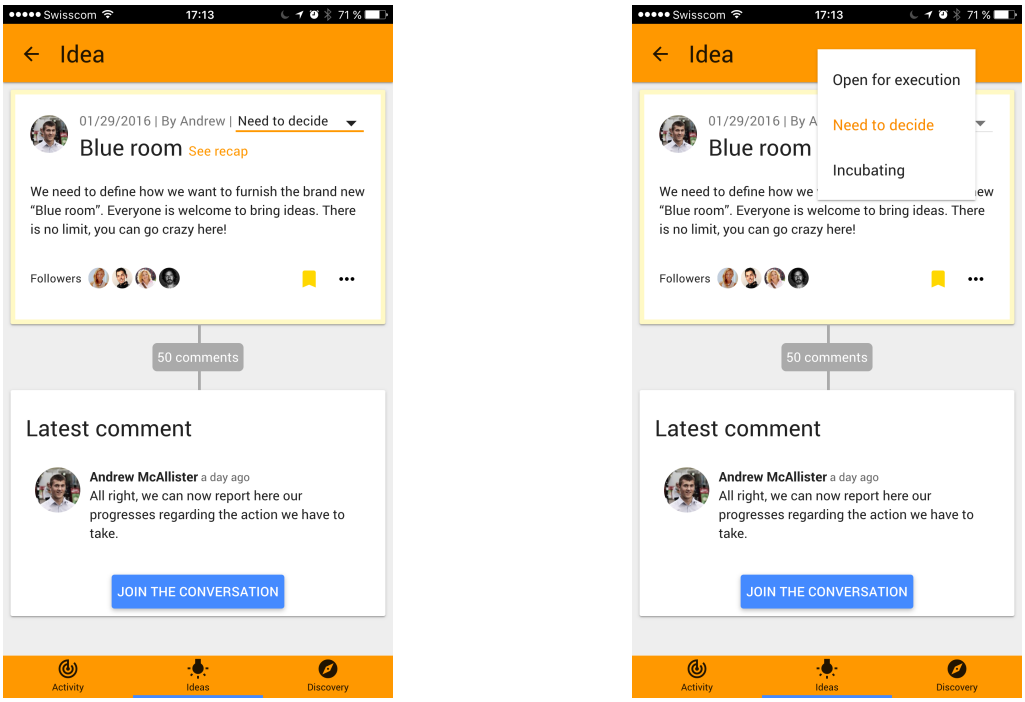


Figure 9: Screenshots of Flok front-end featuring the simple state update of an idea

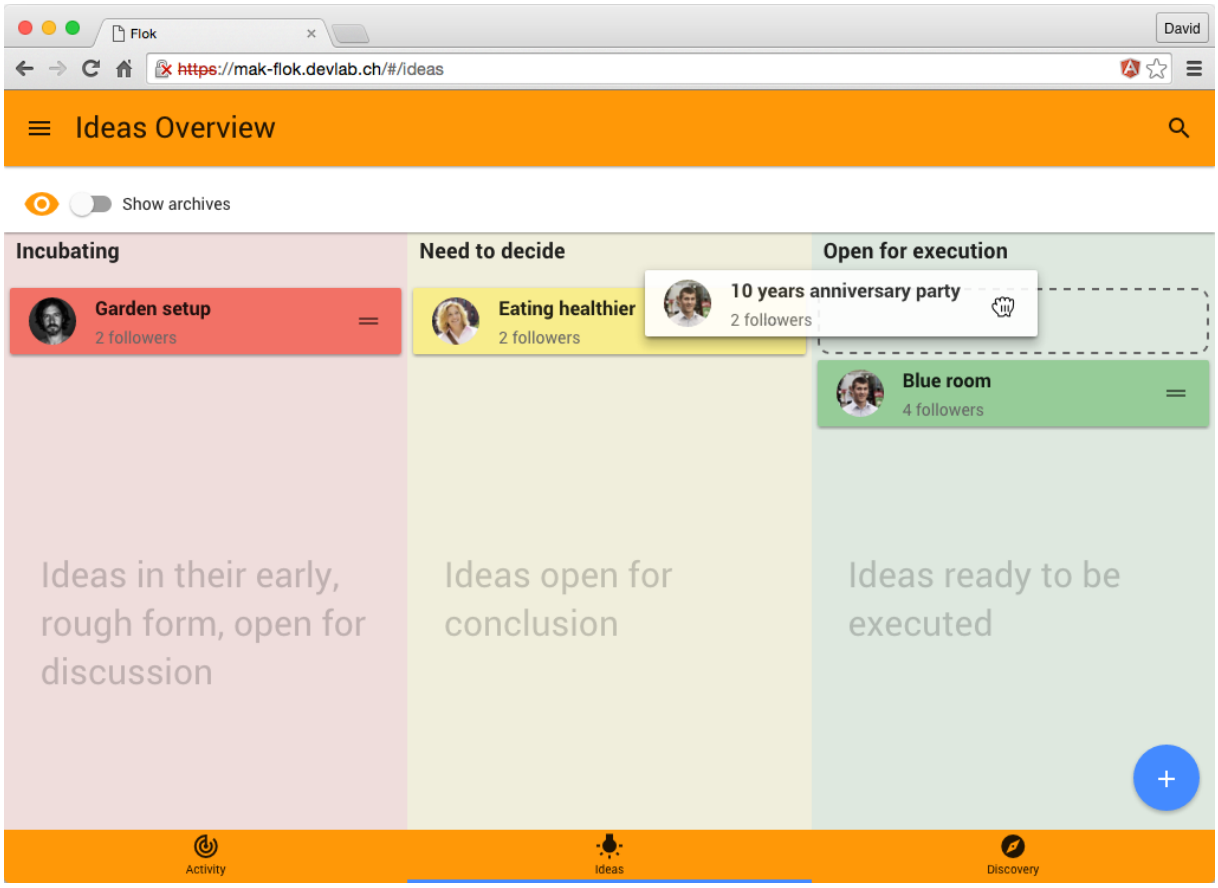


Figure 10: Screenshots of Flok front-end featuring the drag-and-drop of an idea from a state to another

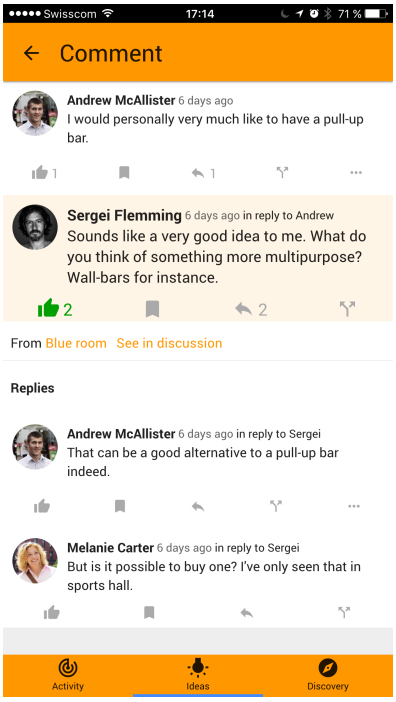
7.2.3 Comments

Comments are one of the central components of Flok. They have one author, they are part of one idea and they are either a note (a text comment), an image, a document, a system information or a deleted comment. When adding a comment to a discussion, it can be the reply to another comment and thus help to establish relations among comments (see figure 11a). For more details on the properties of each of these type and on the relations of comments with other entities of Flok, we send you back to the entity-relationship diagram in figure 3. The user flow to add a comment is illustrated in appendix A.4.

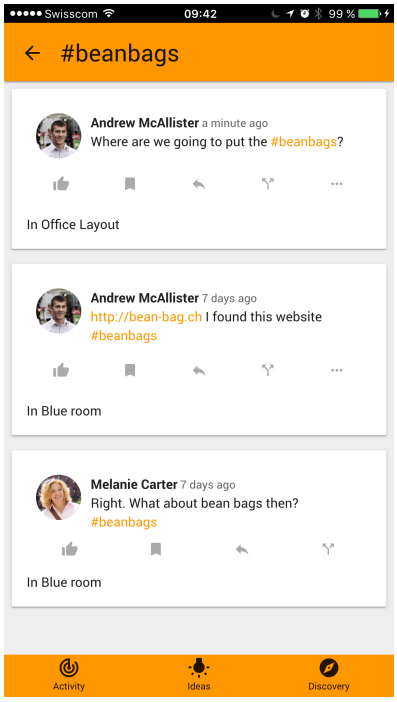
We would still like to mention the support of hashtags[9] in comments. When clicking on one, we are brought to the hashtag view, which list all occurrences of the hashtag in all ideas (see figure 11b).

Below each comment, a list of actions is offered to the users. They can like the comment, bookmark it, reply to it, or branch it. Liking a comment is a visual indication showing support, agreement or sympathy. Bookmarking a comment adds it in the bookmark list of the users who can access it from their profile. It is a way to save comments they want to easily access later. Comment branching is covered in section 7.2.4. And if we are the author of a comment, we have an additional *three dots* button toggling a menu to either edit or delete a comment.

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



(a) The comment is highlighted, above we have the comment it is replying to and below we have the replies.



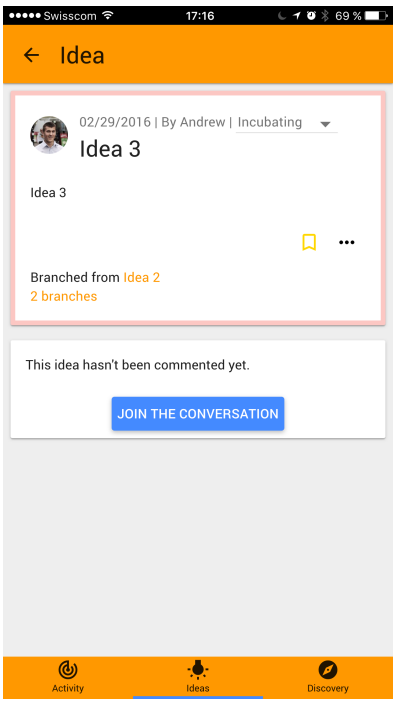
(b) The hashtag view list all the occurrences of a hashtag throughout all the ideas.

Figure 11: Screenshots of Flok front-end featuring comments and hashtags.

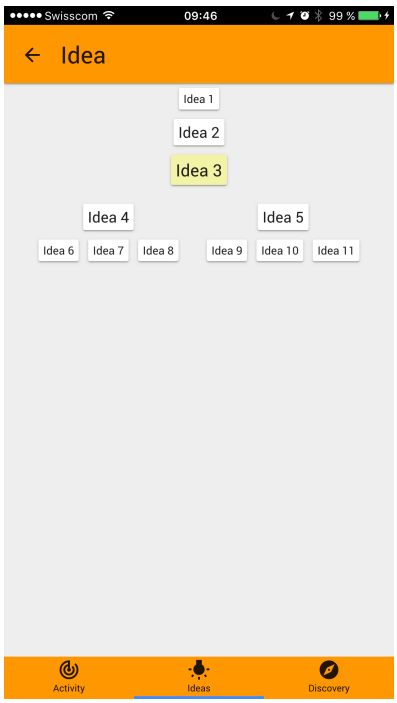
[9] https://wikipedia.org/wiki/Hashtag

7.2.4 Branching

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 *branching* feature. Basically, you can create an idea by branching another idea or a comment. The user flow to branch ideas or comments is illustrated in appendix A.6. The main goal is to create the relation between the initial idea or comment 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 with a standard new idea 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 branches relations (see figure 12). Clicking on the number of branches from the view of an idea will bring to the idea tree navigation.



(a) In the idea view, it is indicated from which idea an idea has been branched from, as well as how many branches it is the source of.



(b) The idea tree navigation shows up to two parents or children branch level.

Figure 12: Screenshots of Flok front-end featuring the branching mechanism.

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 is 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. You already saw how it looks like on mobile in figure 7c. This same layout on a

desktop would be quite inefficient, therefore we adapted the view to give a better experience when showing the recap on a large screen, as you can see in figure 13. The user flow through the recap view is illustrated in appendix A.7.

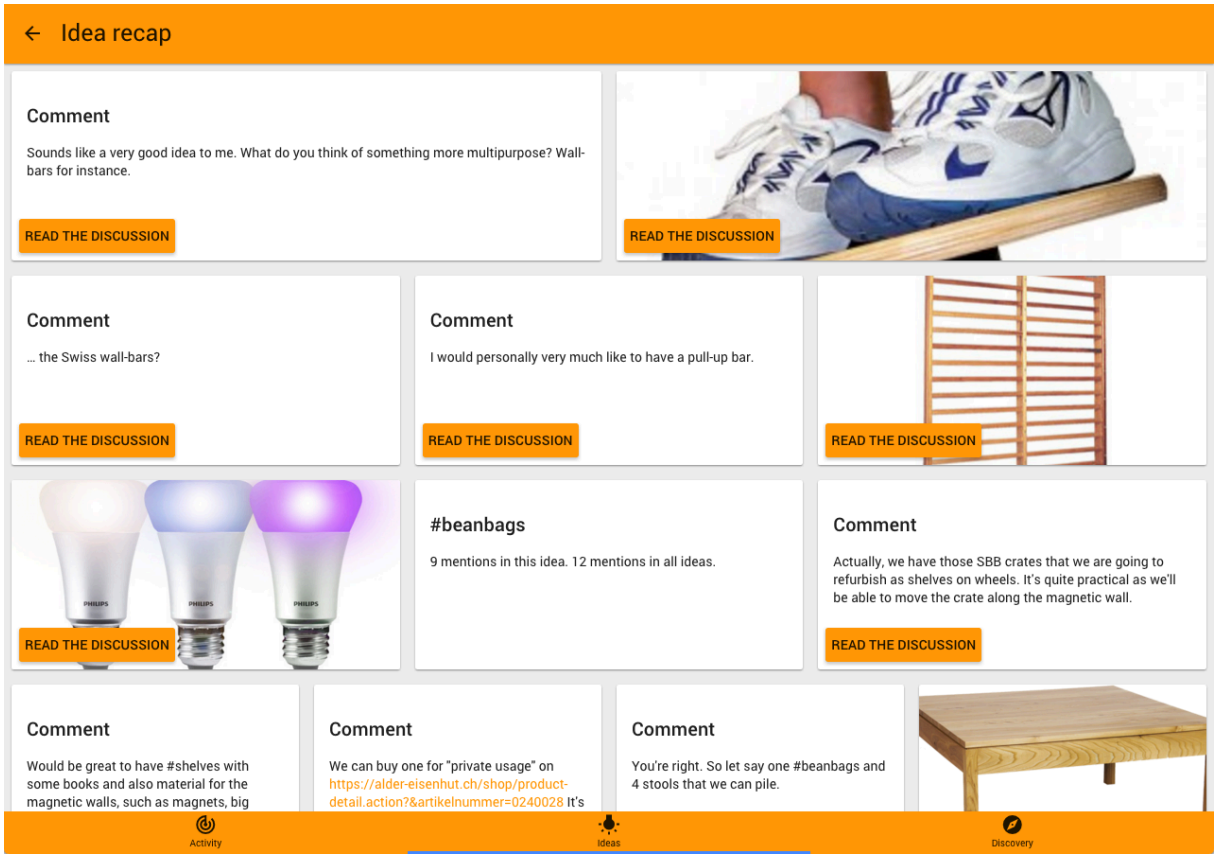


Figure 13: Screenshot of Flok front-end featuring the recap view from a large screen

To select the comments and show them in a specific order, it runs an algorithm which returns the most relevant comments 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 do not have any back-end. That is why we implemented the recap algorithm in the front-end.

Initially we wanted to take into consideration the text content of the comments 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 comment of the idea's discussion based on the following properties.

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

The weight of comments w_m is then computed as follows.

$$w_m = 4r + 3b + 2l + f + i + d + \frac{t}{250} \tag{1}$$

The minimum weight for a comment 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 initial content in the recap. As the comments in the recap are ordered by weight, if images or documents do not get more weight, they will simply stay at the bottom.

You will notice in equation 1 that some of the comment properties are weighted with different factors. Replies have the most importance with a factor 4. Indeed, replying to a comment means that this comment generated engagement and new content in the idea. If users bookmark a comment, it means they want to have it saved for themselves. It affects their bookmark list and they will therefore be careful regarding what they bookmark. This shows a strong interest for the comment, thus bookmarking being the second most important property with a factor 3. The likes however, are only affecting the comment 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. Branching a comment means creating a new idea. Therefore, we could say that a comment that has been branched is of great importance. It certainly is for the idea ensued from the branch, but for the idea containing this comment, it does not necessarily have that much importance. If it does, then it would get replies, bookmarks or likes that are going to make it appear in the recap. This is why branches only have a factor 1 in the computation of a comment weight. The length of a text comment also influence the weight of the comment. If it has enough content, i.e. 250 words, 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.

In addition to comments, the recap view also features hashtags. To show them among the comments, they also have a weight w_h based on the number of mentions m.

$$w_h = 0.5m \tag{2}$$

Hence, a hashtag appears in the recap if it has at least two mentions in the discussion.

Of course, for this simple algorithm, the given 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 to the user the recent events occurring in ideas she follows (see figure 14). This allow her to have a quick overview of everything that happened in the ideas she is interested in. The actions that are logged in the activity are the following:

- creating an idea
- branching an idea
- following an idea
- moving an idea from a state to another
- renaming an idea
- adding a comment (note, image or file)
- liking a comment

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.

This part of Flok did not receive as much improvement as other parts. Indeed, it is interesting to look at the activity in the platform to see what others are doing. But as interaction between users is not possible yet without a back-end, it was difficult to see the activity grow and therefore notice if the way it has been implemented is adapted.

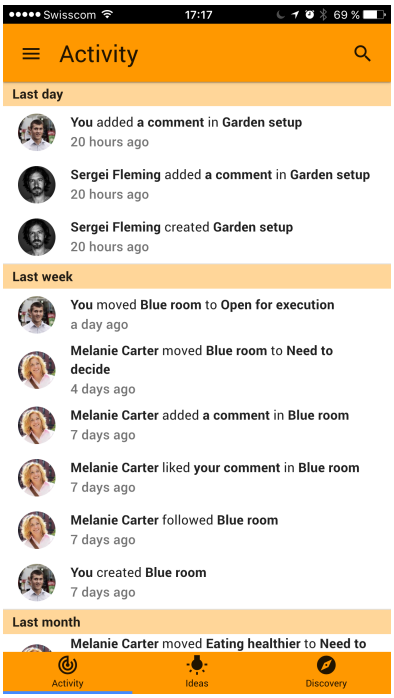


Figure 14: Screenshot of Flok front-end featuring the activity of the platform

8 User testing

In order to check the assumptions made during the user-centered design process, we made three user testing sessions. We used the service TestingTime[10] to find test users. The tests were made through Skype. The test users shared their screen with us. We spread the tests over one week, on Monday, Wednesday and Friday. The time between two tests allowed us to make some adjustments to the front-end before the next test and directly see if it already improved the user experience.

When explaining the test and giving instructions, we did not give information about how the app works. We wanted to see if people can understand the usage by themselves. We asked the test users to describe us what they do and what they expect to happen when interacting with the app. We also asked them to tell if they understand or not what is happening and if there is anything they like or dislike when using Flok. Moreover, as the test session have been done with desktop computers, the app was tested with a desktop viewport.

The test sessions were divided in two parts. The first part was about testing the app without any initial content. The goal is to see how someone discover and learn to use the app. As there

[10] `http://testingtime.com`

cannot be two people communicating through Flok yet (see section 7.1), the test users cannot have interaction with other users or see how the app could be used to collaborate. That is were the second part of the test session come in. There, the test users start using the app with some initial mock content. They can see discussions between several people about one idea, which could help them understand other aspects of the app.

In the following subsections for each test session, we give the good points, which are the positive aspects we noticed during the test session or that were explicitly given by the test user. Then we explain how various features were improved after having seen the test user misunderstanding or struggling with them.

Unexpectedly, it appeared that the first test user did not have an up to date browser installed on her computer. Older browser compatibility is something that has been put aside for now and therefore the layout of the app was not correct at several places. As the test was limited in time, we did not asked the user to download a more recent browser and tried to test the app with the broken layout. Fortunately, almost all the functionalities were working correctly. From that situation, we learned that we'd better ask in advance the test users to have an up to date browser ready for the coming test.

N.B. The screenshots in this the following subsections have been cropped to highlight a specific part of the user interface.

8.1 First test session

The first test user was a 34 years old woman.

She was clearly not experimented in computer usage, which was a good thing. Indeed, observing her behavior highlighted some bad interface implementation which we did not notice by ourselves as the way it was done was obvious for us.

As already mentioned, she was using an outdated browser which did not render the app layout as it should. Therefore, during the test, there was also some misunderstanding of how some of the features should work.

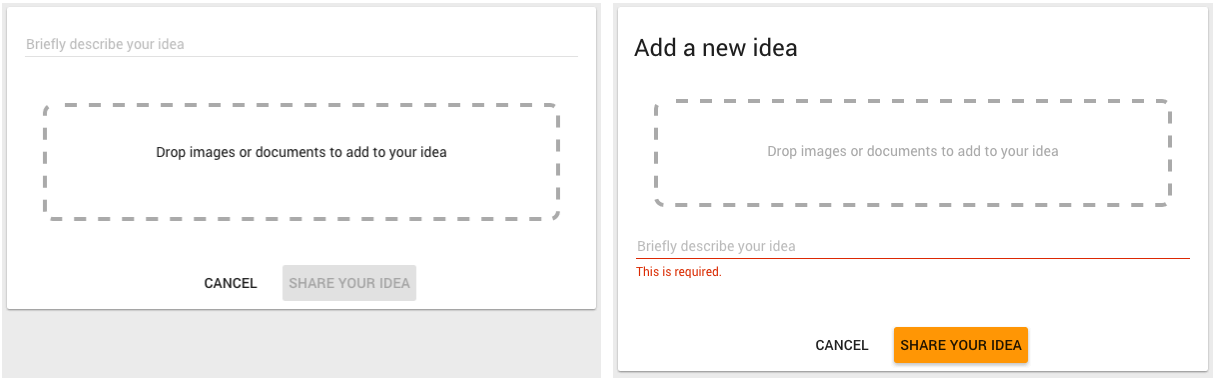
Unfortunately, she did not get to the recap view of the idea tree navigation view. The layout issue might be the reason, hence we waited for the next test to see if it is needed to do something about it.

Good points

- She immediately noticed the plus (+) button to add an idea, which tells us it has a good visibility.
- She understood well the idea state concept.
- She liked having a global view of the ideas and how "everything is linked together".
- She liked to be able to see other person profile and their activity in the platform.
- She mentioned that the app could be useful for companies.

New idea form

In the form to create an idea, the test user first noticed the drop zone for files, where she added an image. Then she tried to click on the *Share your idea* button, but it was disabled as she did not write anything in the idea description, which is mandatory (see figure 15a). She did not understand why it was not working as she did not even noticed the text field. It was obvious for us that you should write a description for your idea and therefore, did not realize that it was necessary to highlight this field more. What we did to improve the user experience is to make the text field error comment visible not only after having clicked once on the field and leaving it empty, but also when clicking on the *Share your idea* button. We also placed the field closer to the button (see figure 15b).



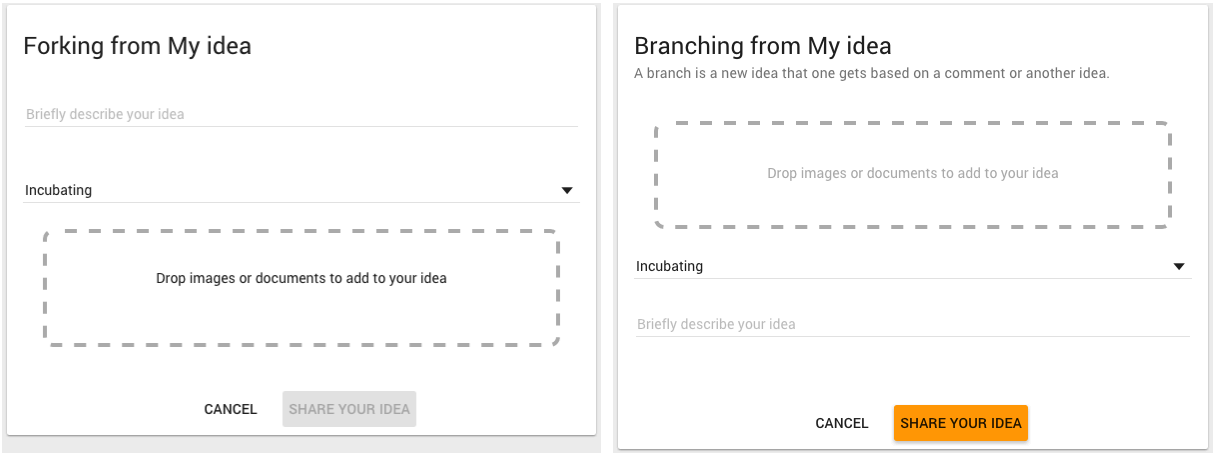
(a) Before user testing. The button to submit the button is disabled when no description is given and if we click on it, nothing happens.

(b) After user testing. Clicking on the button without having filled the description make the error message appear. The description field has also been moved closer to the button.

Figure 15: Screenshots of Flok front-end featuring the form to add an idea.

Forking/Branching

At the time of the test, the branching feature was called *forking*. When the test user forked the idea and noticed the result, she said she was confused as she was rather expecting the text she wrote for the forked idea to be a new comment within the original idea and not another idea. This made us think about the naming of this feature, which induced the new *branch* name. We deduced that this name is more meaningful for any user as they can easily make the analogy with a tree and its branches splitting into more branches. See figure 16 to notice the difference.



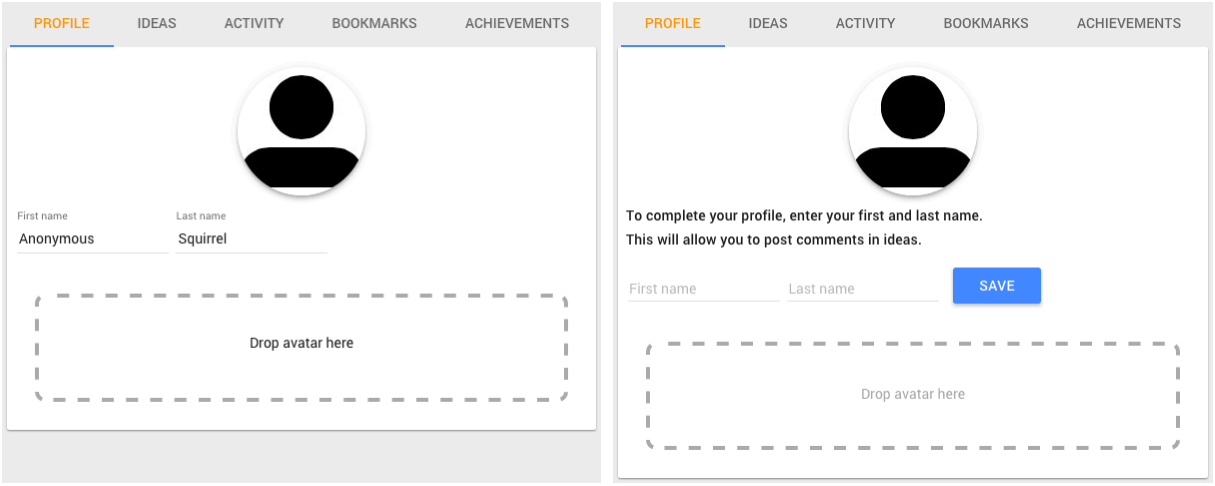
(a) Before user testing. The branching feature was called forking.

(b) After user testing. The feature is called branching instead of forking and has an explanatory text. There are also the same changes as in figure 15.

Figure 16: Screenshots of Flok front-end featuring the idea view.

Profile completion

As the test user tried to add a comment in the discussion, she noticed that she could not when she saw the message saying that she has to complete her profile first. She clicked on the given link to perform the profile completion, but once on the profile tab of the person view, she rather went through all the other tabs rather than changing her first and last name. Therefore it was pretty clear that we should clarify what has to be done for the user. To do so, on the profile tab of the person view, we added a message in bold text stating that "*to complete your profile, enter your first and last name*", and a second line informing of what it will enable the user to do. Also, instead of having the first and last name filled with the placeholder "Anonymous" name, they are now empty. See figure 17 to notice the difference.



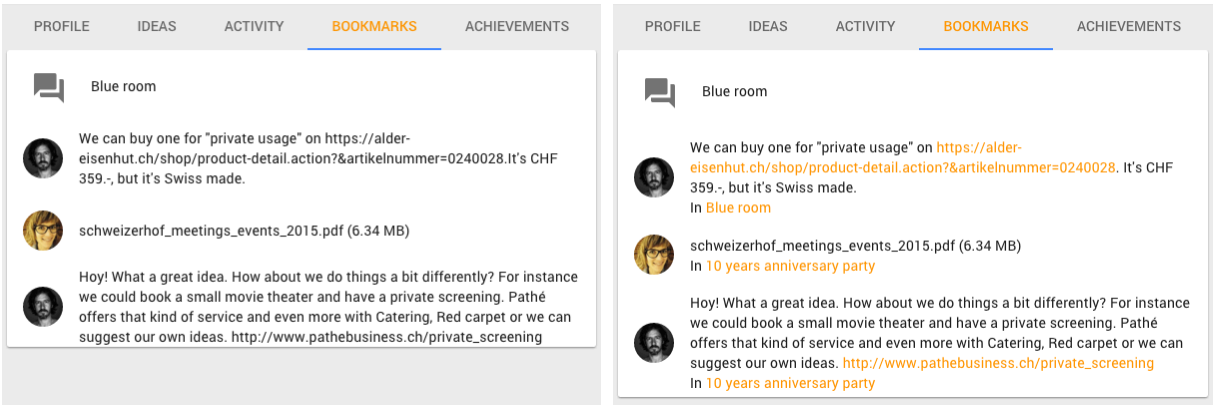
(a) Before user testing. The field were filled with the "Anonymous" placeholder.

(b) After user testing. The fields are empty if users did not provide names themselves and a message tell the user what is needed for the profile to be complete and what it will allow them to do.

Figure 17: Screenshots of Flok front-end featuring the form to edit one own profile.

Bookmarks view

In the bookmark view of her profile, the test user first understood that this was some kind of direct comments between users of Flok. After thinking by herself about it, she finally got that there were ideas or comments from ideas that a user can bookmark to have them in one place. This anyway made us realize that it was not immediately clear. The comments notably, were just consisting of their text content and the author avatar (see figure 18a). We now have changed that so that they also feature the name of the idea they are from (see figure 18b).



(a) Before user testing. It was not clear from where the comment bookmarks are coming.

(b) After user testing. The comment bookmarks show the name of the idea they belong to. We can also notice that the links in the comments are automatically detected.

Figure 18: Screenshots of Flok front-end featuring the bookmarks view.

8.2 Second test session

The second test user was a 42 years old man.

Unfortunately, we noticed at the beginning of the test that we forgot to deploy all the changes made since the first test on the staging server. In the end it was not such an issue. Indeed, as this test user was using an updated browser – not like the first test user – the user experience was already different and there was almost no overlap in the "problems" he faced. Therefore, the test still made us notice quite a few other improvements we could make to the front-end. This can also be due to the fact that his experience in computer usage was significantly more advanced than the first test user and hence, has quite a different perspective when interacting with a user interface.

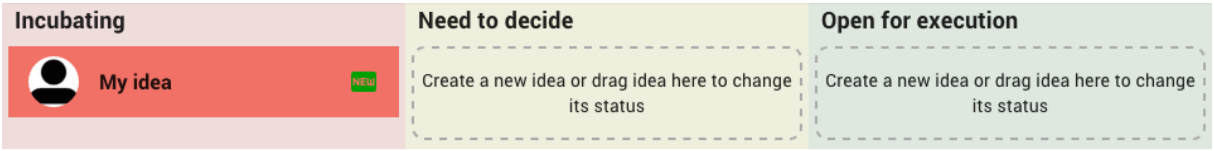
Good points

- He interacted as expected with the profile completion process.
- He interacted well with the branching navigation and found that it was clear.
- He liked that ideas can be reordered to prioritize them.
- He liked the interface, the choice of colors, the clarity, the easy navigation and the fact that it is not cluttered

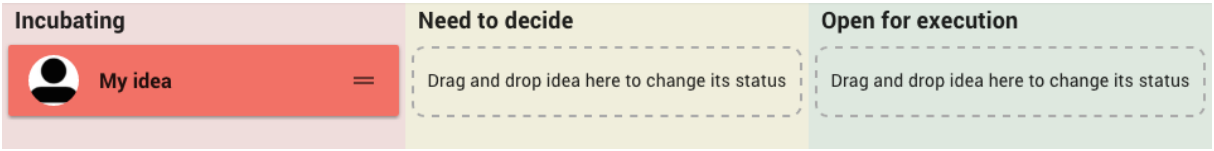
- He found the app was running pretty fast.
- As a person working in the music events domain, he would imagine Flok working very well for the organization of bands or festivals.

Ideas overview

This test user did not immediately realize that he could drag-and-drop ideas, even with the comment "*Create an idea or drag an idea here to change its status*" visible in the empty columns. The beginning of the sentence seemed to be confusing as he tried to click on it to create an idea. We made a few changes to improve this issue. First we re-formulated the sentence to "*Drag and drop idea here to change its status*". We also added a drag handle icon on the ideas in the columns as well as a shadow to increase the feeling that it can be moved. See figure 19 to notice the difference.



(a) Before user testing. It was not clearly visible that ideas could be dragged-and-dropped and the text in the drop placeholders could be misleading.



(b) After user testing. Ideas have a shadow and a drag handle, making it more obvious that they can be moved. The icon indicating that an idea is new has also been removed as it did not bring much value.

Figure 19: Screenshots of Flok front-end featuring the ideas overview.

Bookmark action

The test user guessed that the bookmark button meant bookmarking, but he was not sure at all and clicking on it did not help him understand as it just colored it and incremented the counter. He said that if it is indeed a bookmark, he would still not know where to look for it. This made us add a small notification showing when bookmarking a comment or an idea (see figure 20). It also contains a button that the user can click to go see his bookmarks.

Forking/Branching

When the test user clicked on the fork (previous name of the branch feature) button of a comment, and created a new idea, he was a bit confused as he expected the forked idea to be just inside the comment and not as a whole new idea. He understood by himself afterward, but this comforted us that the changes we brought after the first user test, by renaming the feature and giving more explanations in the form to branch an idea, are indeed needed to make it clear even sooner (see figure 16b).

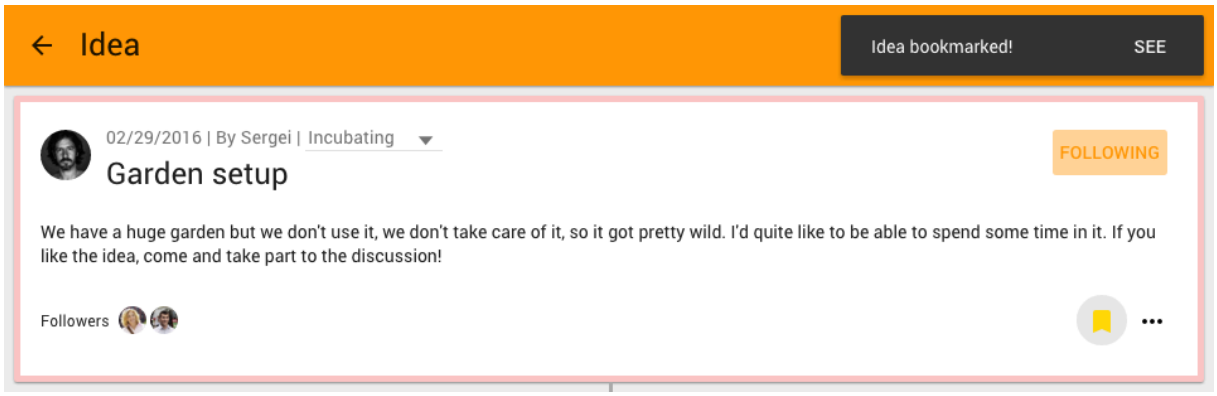
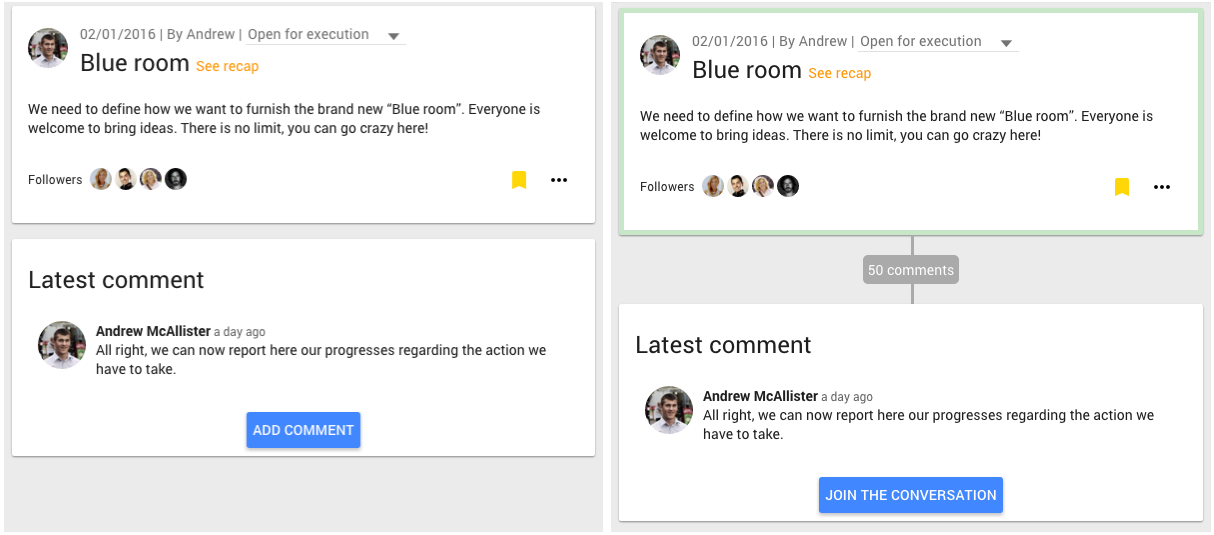


Figure 20: Screenshots of Flok front-end featuring the bookmark notification.

Idea view

When the test user clicked on the button to add a comment in the idea view, he was surprised to see all the comments there was, as nothing on the main view of the idea was telling him how big already was the discussion. To prevent this reaction, we added a comment counter in the idea main view (see figure 21).



(a) Before user testing. It was not clearly whereas the idea had already been discussed a lot or not.

(b) After user testing. A comment counter gives the number of comments posted in the discussion. There is also a lightly colored frame around the idea card indicating its state.

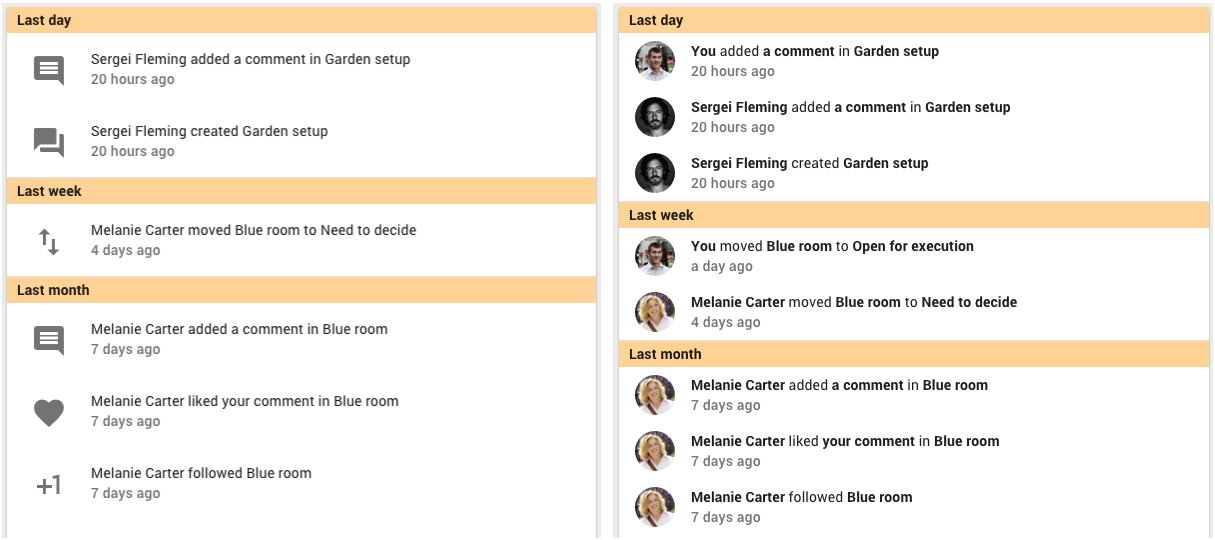
Figure 21: Screenshots of Flok front-end featuring the idea view.

Recap view

Then the test user visited the recap page of an idea and was not really sure of how useful it would be for him. He said he prefers having the comment in chronological order. We did not make any change here as we wanted to see how the last test user would react.

Activity

In the activity view, the test user expected to see his own activity, which was only visible from his profile. Initially we thought that the users of the app would not like to see what they did as they know about it, but it appears that in the end it is rather counter intuitive. Hence we added the user activity to this view as well. See figure 22 to notice the difference.



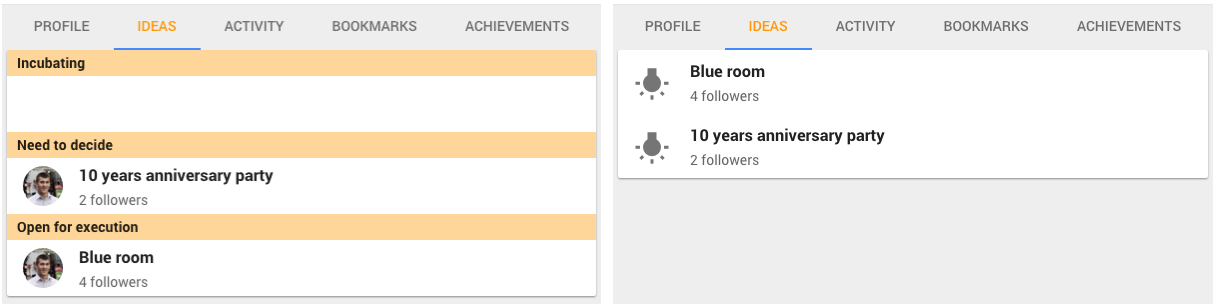
(a) Before user testing. The activity was not featuring the user own activity and the icons represented the type of activity.

(b) After user testing. The activity view also contains the user own activity and highlight the actor of the activity by using its avatar instead of an icon.

Figure 22: Screenshots of Flok front-end featuring the activity view.

Ideas in profile

Finally the test user went to his profile and in his ideas view, he expected to see not only the ideas he created, but also the one he is following. But this would not be so different than the ideas overview. Therefore, what we rather did is to simplify this view by removing the state indication and not showing the author avatar as it is always the user itself in this view. See figure 23 to notice the difference.



(a) Before user testing. The view was still showing the state of an idea.

(b) After user testing. The view has been simplified and does not show the avatar of the user as they are all from the same user.

Figure 23: Screenshots of Flok front-end featuring the user own ideas view.

8.3 Third test session

The third test user was a 26 years old woman.

For this test we did deploy the changes we brought after the last two tests. Therefore, all the changes brought after the two first tests were applied.

Good points

- She understood the bookmark action thanks to the small notification we implemented between the previous test and this one (see figure 20).
- She found the recap view interesting and imagined that it could indeed be quite useful.
- She understood well the branching feature, notably thanks to the explanations we added, after the two first tests, in the form to create a new branch.
- She appreciated the gamification side with the achievements.
- She liked the simplicity of the interface and added that it helps to understand the concepts that are a bit complicated.
- She said that at the beginning she needed some minutes to get into it, but that it gets pretty clear after having used the app.

Profile completion

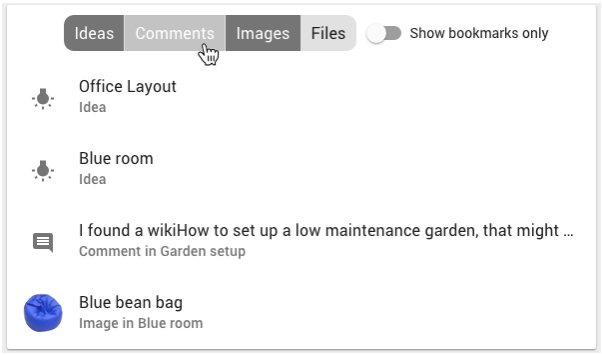
After having added an idea, the test user saw the banner telling her that she can complete her profile, but she did not do it. We realized that the banner suggest to complete the profile to let others know who the user is, but it does not say that it would enable the user to add comments. Therefore we changed the text from "To let others know who came up with this idea, complete your profile." to "To let others know who came up with this idea, *and for you to be able to add comments,* complete your profile.".

Ideas overview

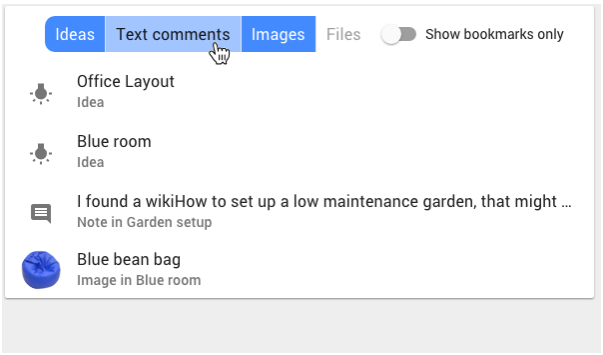
The test user noticed that ideas can be dragged-and-dropped thanks to the visual changes we brought since the previous test (see figure 19). However, she did not get that the drag handle icon was the part that needed to be dragged and tried in vain to move the idea by dragging it from the middle of it. As she is the only test user that had this new interface, we did not make the change to have the whole box draggable again. It would be better to do further user tests to check if the behavior is the same with other people using the new interface.

Search

In the search results view, the test user immediately understood the "*Show bookmarks*" switch toggle. However, she got really confused with the filters to show or hide Ideas, text comments, images or documents. It was not clear for her when they were activated or not (see figure 24a). To fix that, we changed the design by making a filter toggle less visible when deactivated, and colored when it is activated (see figure 24b).



(a) Before user testing. It is not really clear if the visual state of a filter toggle mean that the filter is activated or deactivated.

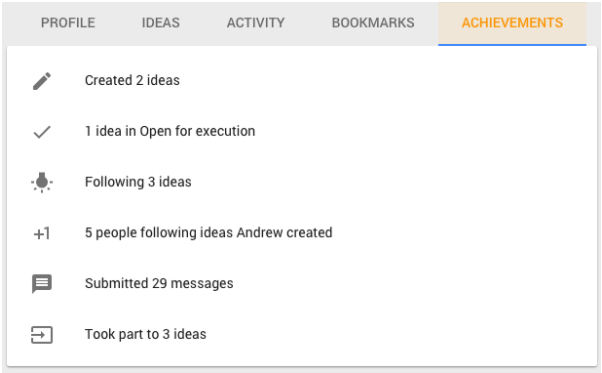


(b) After user testing. The visual state of the toggles is clearer by making the deactivated state lighter and adding color to the activated state.

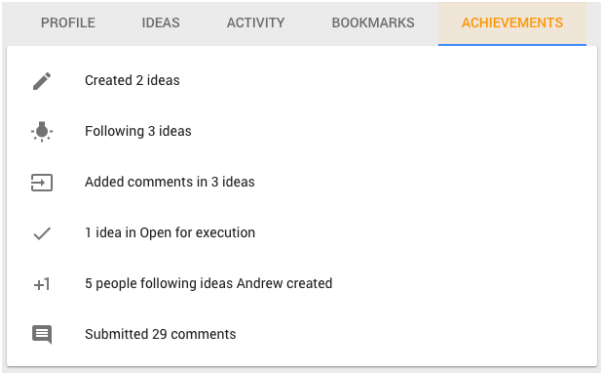
Figure 24: Screenshots of Flok front-end featuring the search results view.

Achievements

The test user was not sure of the difference between the number of ideas followed and the number of ideas she took part in (see figure 25a). The latter counts the ideas where she posted at least one comment. The simple fact that we feel the need to explain it here means that it requires clarification and that is why we simply changed it to the number of ideas in which you *added comments* (see figure 25b).



(a) Before user testing. It is not clear what taking part to an idea mean and could be mingled with the number of following ideas.



(b) After user testing. The confusing achievement now says that it is the ideas where we added comments and has been moved closer to the following ideas to make the user notice they are not the same.

Figure 25: Screenshots of Flok front-end featuring the achievements overview.

9 Conclusion

For this project, we have taken a user-centered approach to test the hypothesis, by focusing first on the user needs. We started by defining personas to enforce ourselves to see the app as if we were its user. We used user story mapping to create descriptions of how users are going

to interact with Flok. We constructed an entity-relationship diagram to represent the information architecture of the product. We built wireframes to quickly have a rough overview of the whole product and detect as early as possible where the user interaction can be improved. We designed a usable visual prototype to have a better idea of the look and feel without having to worry about the logic, letting us to be flexible to frequent and drastic changes. We implemented a functional front-end in the form of a single page application, currently without back-end, but with all the data saved in memory for the duration of a session. We conducted three user testing sessions which allowed us to gather valuable feedback and make significant improvements regarding the user interaction.

At this point of the project, we have a working front-end. Time limitation made us foster user testing at the expense of the back-end implementation. The drawback is that the current lack of back-end does not allow us to test the real-time approach mentioned in the initial hypothesis. It is therefore difficult to rigorously establish how the shared know-how or the overall collaborative spirit of teams has been improved. The advantage is that user testing really allowed us to have a more human experience by using the feedback we got to make the interface simpler and more intuitive. The test sessions did not only reveal what needed improvement, but also what was already well done. For most of the components, the test users interacted with the interface as expected and understood what was going on. Moreover, when we did make improvements to the app between two test sessions, they were proven to be good. The test users also gave us direct feedback about what they think of the app. We mainly retain that they liked the interface and how its simplicity helps to understand the concepts that could be complicated to grasp, such as branching for instance. The second phase of the tests gave the test users a glimpse at what collaboration would look like within Flok. Their feeling was that the app could be useful for companies or organizations.

It is difficult to make strong conclusions based only on the three test sessions. However, these tests enabled us to uncover most of the usability issues. The feedback we have got and the improvements they allowed us to make are really promising and tell us that we are going in the right direction.

That is quite unfortunate that we had to make this hard choice between back-end and user testing. If we could have had both, we could have been able to test real interaction between users and notice how it could have improved collaboration. Maybe if we had taken more time at the beginning to plan a bit on the longer term, we could have been able to do both. However, it would have been at the cost of another part of the project where we would have had less time to accomplish. Then it is a question of balance between the different components of the project and defining where we put the priority. In this case we prioritized the research on user-centered design. Moreover, it would have been a good thing to do more user testing in order to have more feedback to validate or invalidate some of our left assumptions and hence to have more solid results in the end.

It is clear that one of the important next step is to implement the back-end. This will enable us to have a persistent database and to truly make the app collaboration-oriented. For this, it would be important to insure real-time synchronization. That would not only be between different users, but also if a single user change device and want to seamlessly continue where she left.

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/.

A User flows

This appendix illustrate various user flows in Flok. Some may be described or mentioned in the report, but nothing shows exactly how it looks like as it is not a necessity in their context.

It is also possible to try out the latest version of Flok's front-end at the following URL: `https://flok.preview.ch`. It requires to log in with the following credentials:

- Username: flok
- Password: kovokukako-76!

A.1 Adding an idea

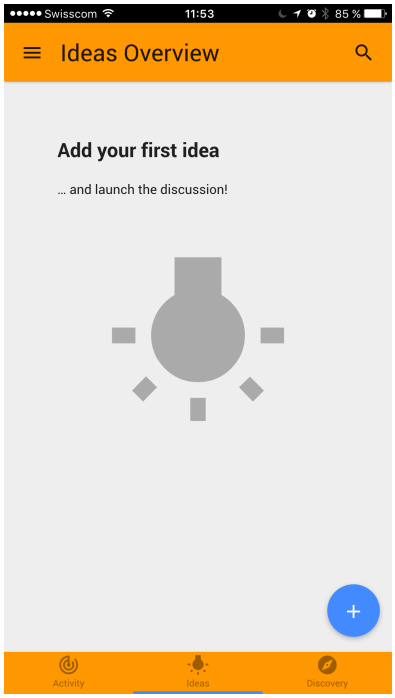


Figure 26: When loading Flok for the first time, the ideas overview is empty as we haven't created any idea and aren't following any neither. To add an idea, we must tap on the plus (+) blue button.

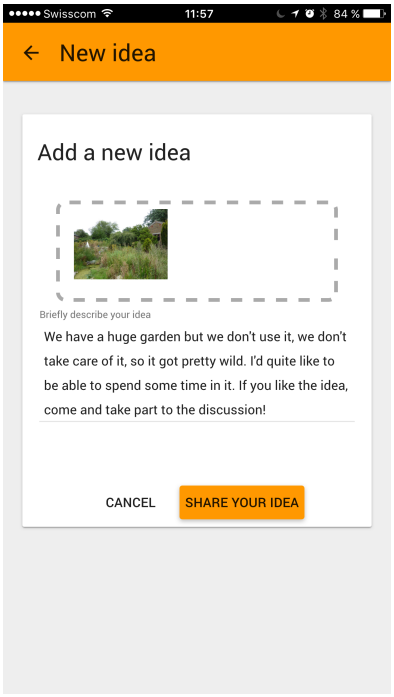


Figure 27: The form to add a new idea, with a descriptive text field and drop zone for files. To enable devices without drag-and-drop abilities, taping on the drop zone will show the file selector.

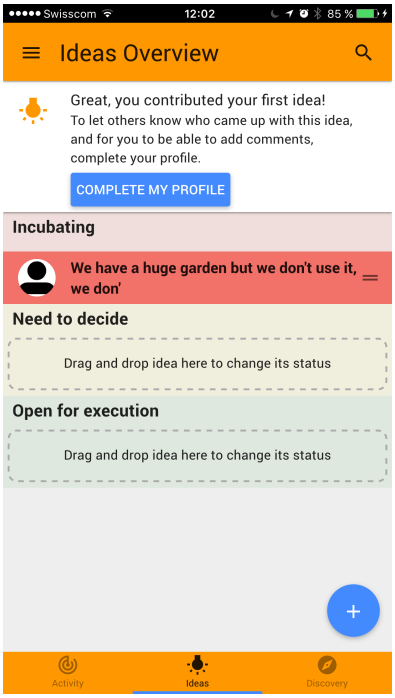


Figure 28: Once the idea added, we are brought to the ideas overview where the newly added idea can be seen. As a new user, we are suggested to complete our profile (see section A.2), notably to be able to comment about any ideas.

A.2 Completing your profile

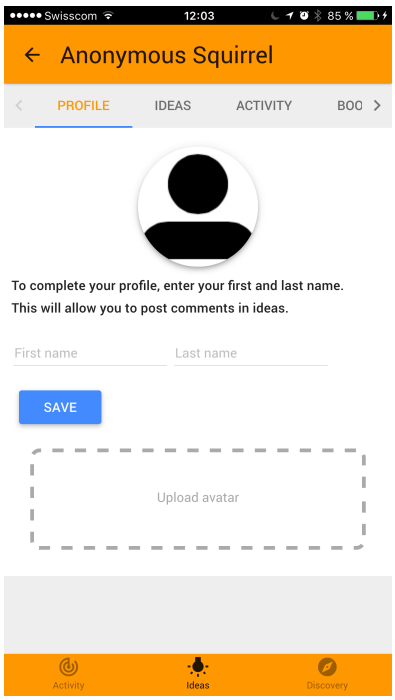


Figure 29: As a new user, after having created our first idea (see section A.1), we are suggested to complete our profile. Clicking on the button to do so bring us to this form.

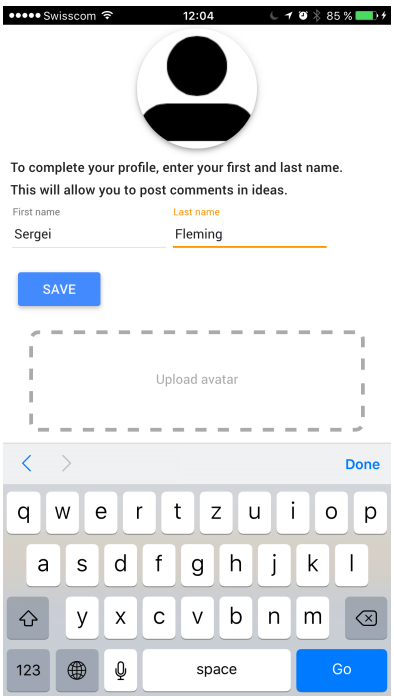


Figure 30: In this form we can give our first and last name and define our avatar.

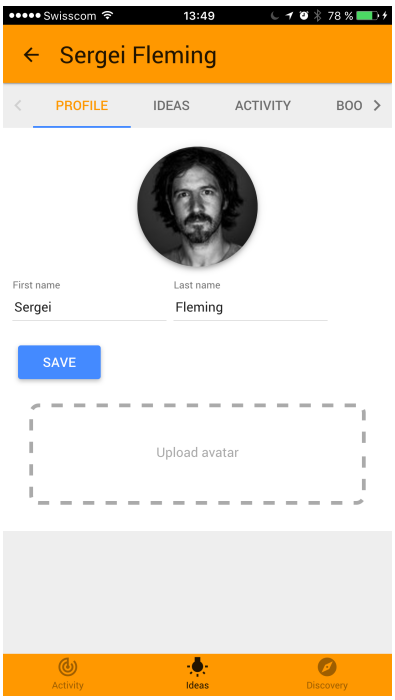


Figure 31: Taping on the save button will immediately update our profile.

A.3 Updating the name of an idea

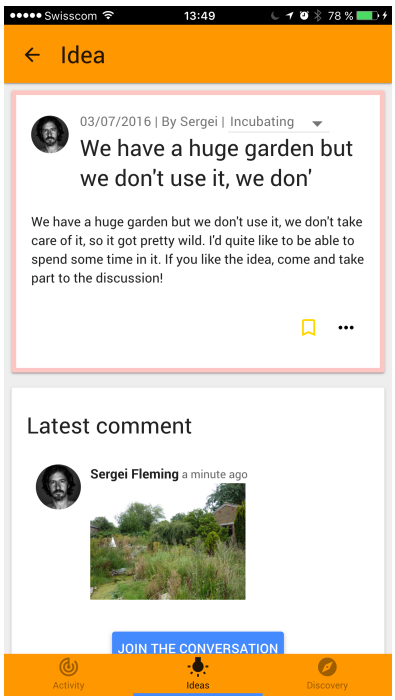


Figure 32: Initially, the name of an idea is an automatically shortened version of the description.

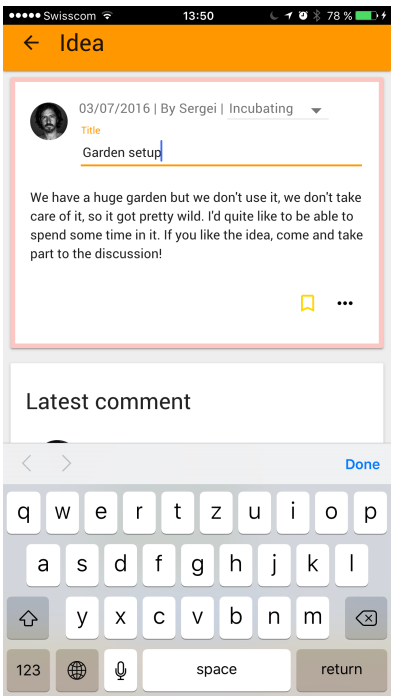


Figure 33: Taping on the name allows us to change it.

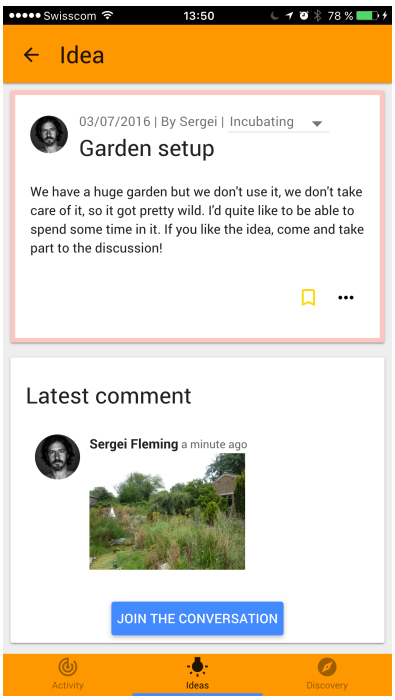


Figure 34: Leaving the text input field immediately validate the change.

A.4 Adding a comment

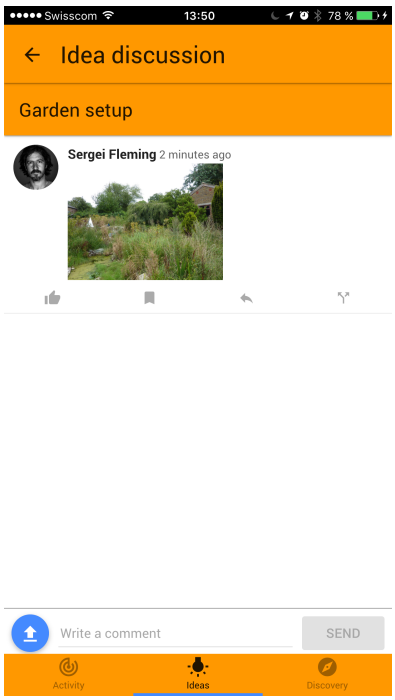


Figure 35: The text field at the bottom of an idea discussion allows to add new comments. The blue upload button let us add images and documents.

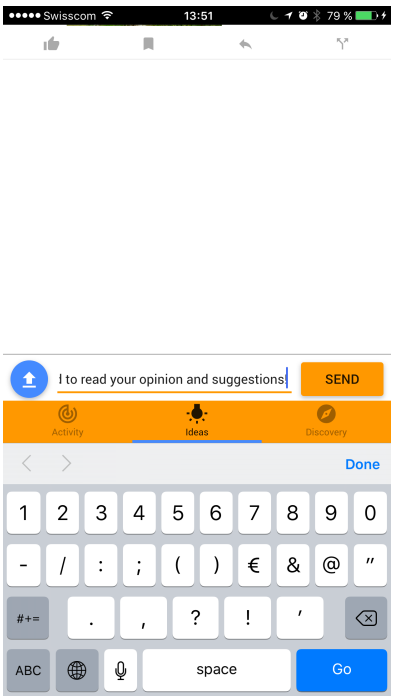


Figure 36: Entering text enables the send button.

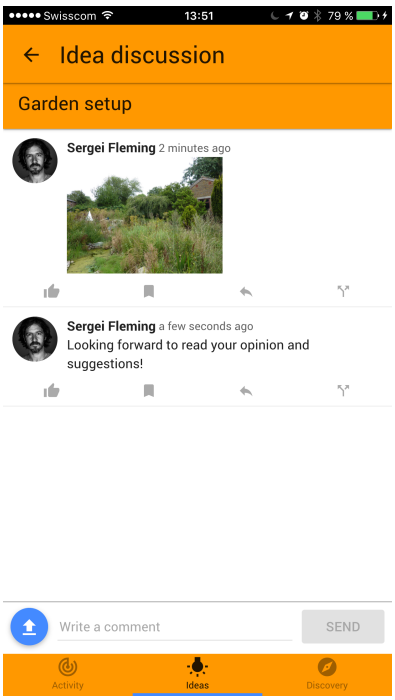


Figure 37: Once submitted, the comment is added at the end of the discussion.

A.5 Discovering and following an idea

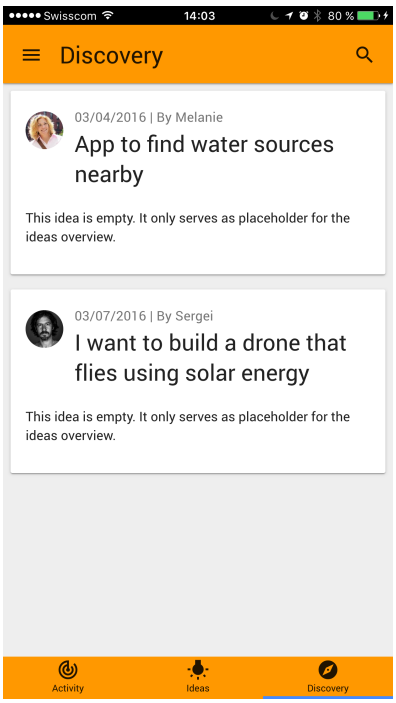


Figure 38: The discovery view showing ideas we are not following.

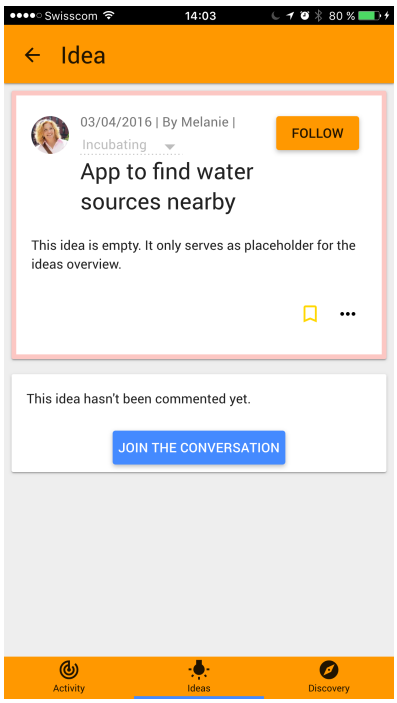


Figure 39: The follow button appears in the idea view of unfollowed ideas.

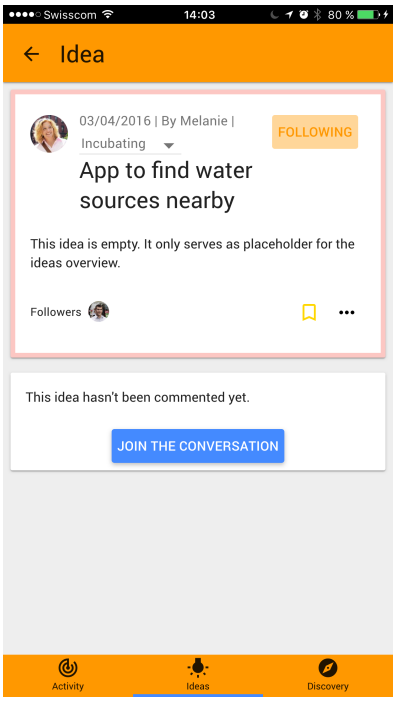


Figure 40: Taping on the follow button moves the idea in our ideas overview and change the follow button a following indicator

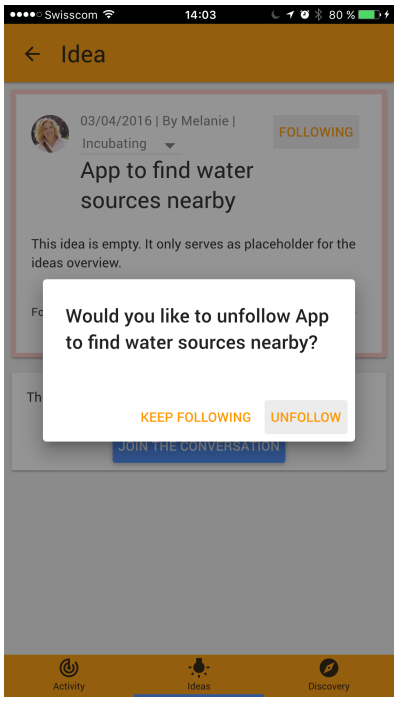


Figure 41: Taping on the following indicator triggers a dialog asking if we want to unfollow the idea.

A.6 Branching an idea

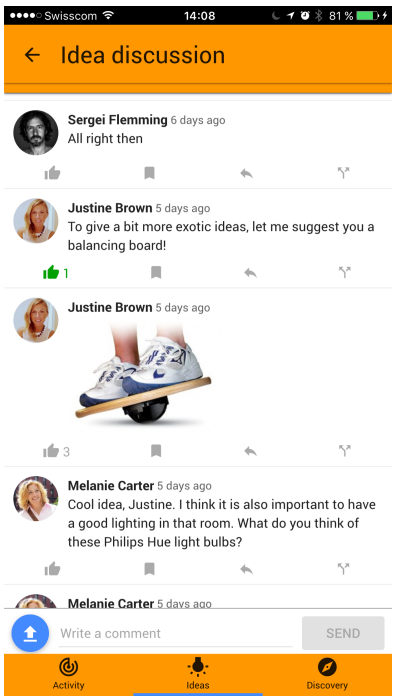


Figure 42: We can branch a comment using the dedicated button at the rightmost of the action buttons.

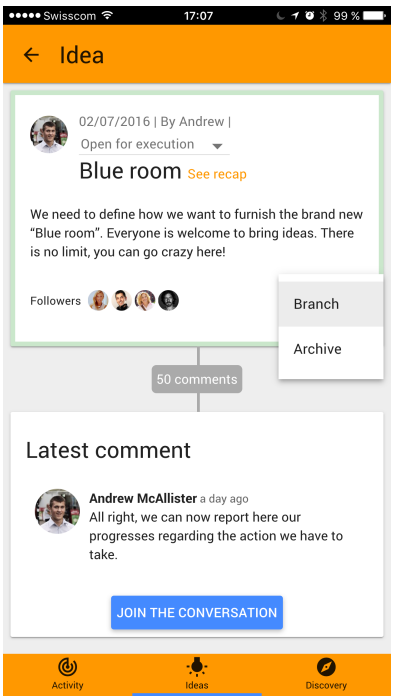


Figure 43: We can branch an idea by tapping the *three dots* button showing a menu with various actions.

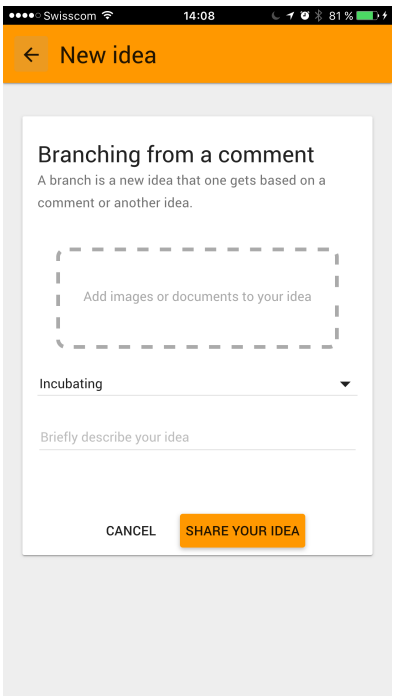


Figure 44: The form to branch an idea is based on the form to create a new idea. The difference is that here we have a description explaining what a branch is doing and we let the user define the state directly.

A.7 Going through a recap

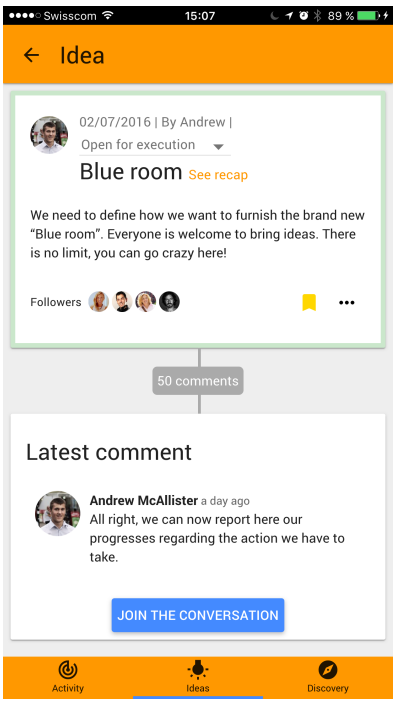


Figure 45: The "See recap" link is visible in the idea view if the recap contains at least two items.

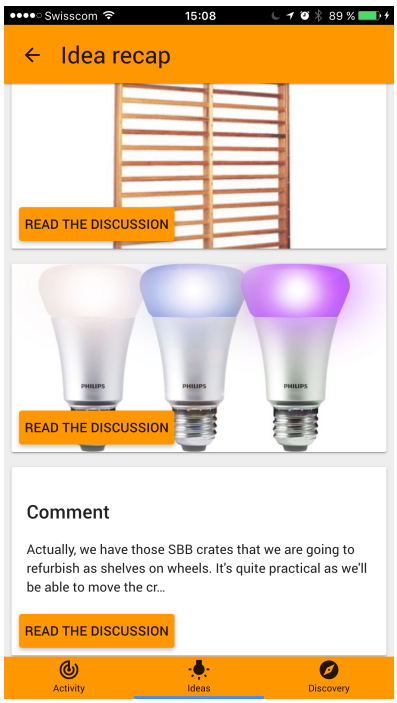


Figure 46: The recap contains images and comments from the discussion, but also popular hashtags.

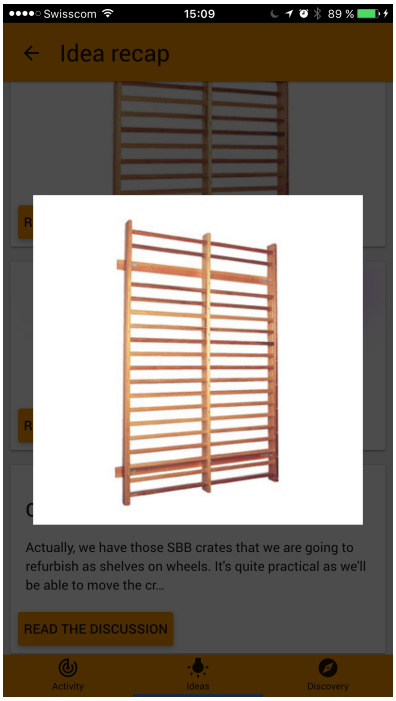


Figure 47: Taping on an image in the recap show it fullscreen. If it is a text comment, we are brought to the comment view.

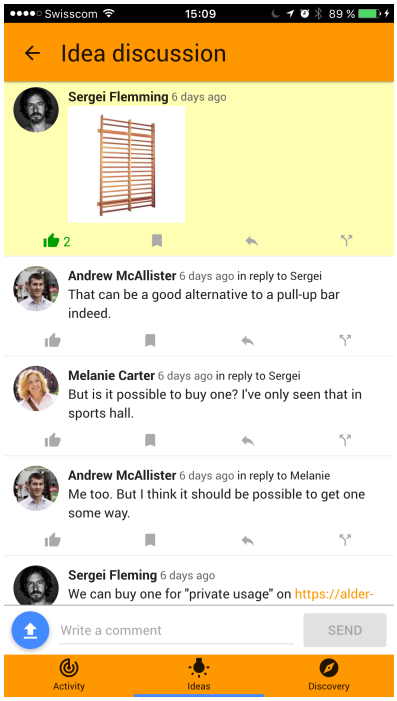


Figure 48: Taping on the "Read the discussion" button brings us in the discussion and highlight the corresponding comment