Training report

EPITECH INTERNSHIP FROM MARS TO AUGUST 2015.

Written by Théophane RUPIN (rupin_t, promotion 2015)

Subject: development of the company's mobile application (iOS and Android) and its RESTful APIs.

Part 1

This part consists of writing a reference guide for newcomers at elCurator

Context

During my internship I have been in charge of the development of a RESTful API and the mobiles applications at elCurator (iOS/Android).

I also participated to develop some of the web features, installed the continuous integration server, written several blog posts, gave an internal conference on a technical topic.

Since elCurator is a start-up project, I had to be autonomous on many points, and I also needed to get involved in the project life.

In this document, I try to give all the necessary informations to quickly be able develop on the web, Android and iOS applications, but also to understand the used methodologies and work with the elCurator's team.

ElCurator - The developer's beginning guide

Summary

What is this document for?

If you are reading this, you probably just arrived in the elCurator's developers team. First of all, welcome, and congratulation.

Because we are aware there is a lot of informations to absorb during your first days, we try to keep this document up to date in order to give you a point of reference concerning our team, our product and our methodologies. You also will get a good idea of what we think a good developer is.

This document is obviously targeting the developers, but we put all our efforts to make it understandable by everybody. If you are curious about what we are doing at elCurator, and even if you are not a developer, this document should still be interesting, and we hope it will statisfy your curiosity.

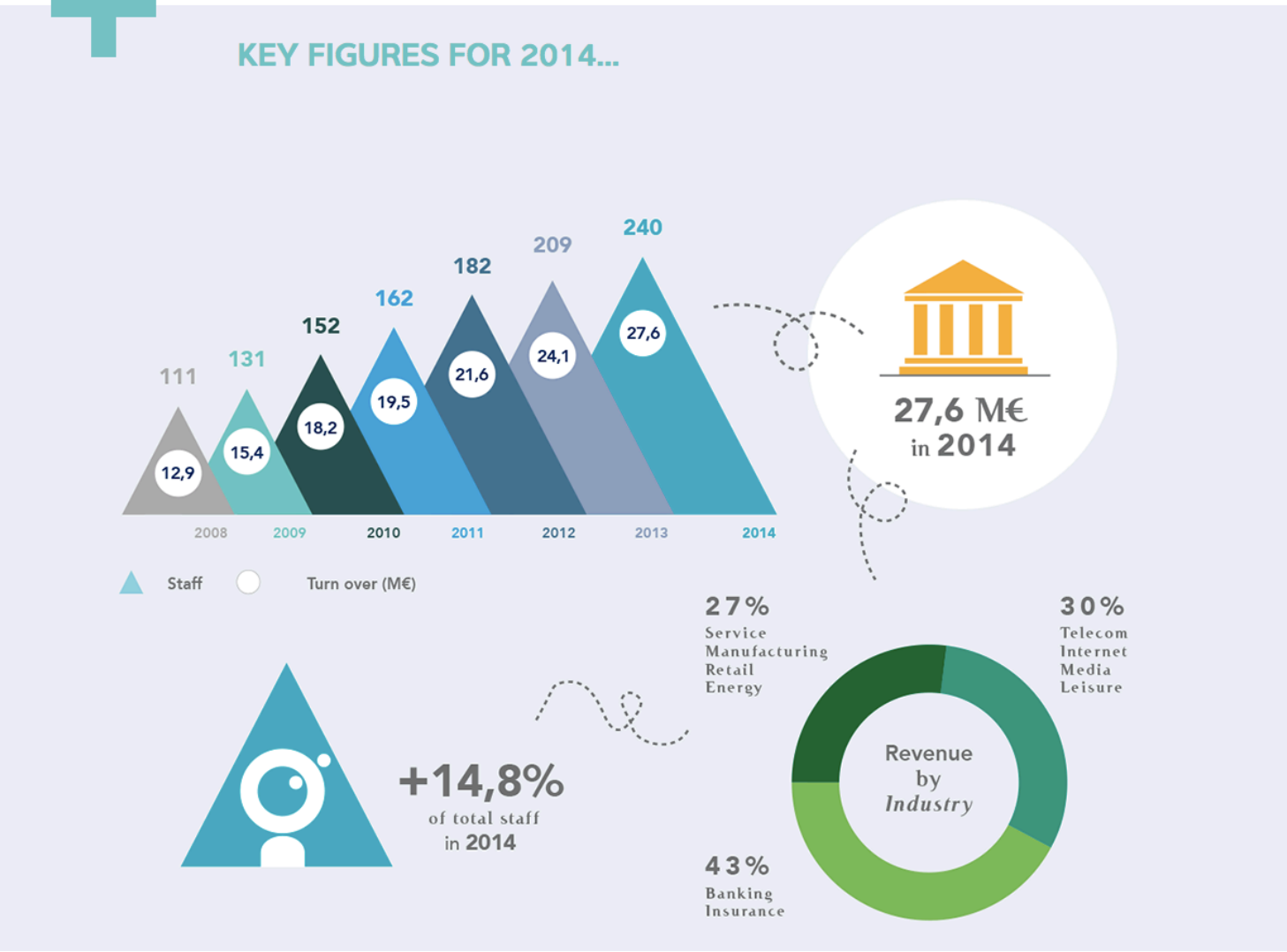
A startup inside OCTO Technology

It is actually quite complex to explain what is our working environment. What is OCTO Technology and how is it related to us? Are we an independant company or not? Who is investing?

We try to clarify these points in this chapter.

What is OCTO Technology?

First of all, we should explain what is exactly **OCTO Technology**. You can find all the informations you need about this company on its website. Even though, we define OCTO Technology as an **IT consulting, design and implementation company**. Founded in 1998, OCTO is now employing 230 people in 5 coutries: France, Brazil, Switzerland, Morocco and Autralia. The OCTO community is mainly made up of IT consultants, and the main activity of OCTO is to guide and help realize its clients' projects.



There is a way of doing things at OCTO which pushes its employees to think about innovative concepts and develop them from inside the company. **That's how two products were born in the company**:

- Appaloosa which is a private mobile application store.
- elCurator which is a collaborative curation platform, but we will explain what it is in the next parts of this document.

From the one day hacking project, to the affiliated company

ElCurator is a product which was **initiated by two consultants of OCTO Technology** in 2012, during a particular day called the OCTO day. This event is happening once a year, and is aimed to let every employees work on whatever they want, as long as it is useful to the company.

That day, **Christopher Parola** - now CEO of the company - started the project with several workmates, and in one day of work, they tried to make a prototype. This was a failure since it

was not working as expected, and Christopher continued to work on it on his free time. **Jeremy Venezia** - now CTO of the company - joined him several weeks after that, and helped him in his task. The product began to work, and a few consultants at OCTO started using it. With time, more and more people were using it, and Christopher and Jeremy decided to **deploy it to the whole company**. Maintaining the project was taking to much time, and the two consultants asked to be full-time working on it. This is how elCurator began to live, as a project funded by OCTO Technology. In June 2014, the project has been **publicly released** and OCTO Technology started to communicate on it in order to sell it to other companies (mainly to its own clients). In January 2015, the project has been **affiliated**, and it became a company named **elCurator SAS**.

Are we a startup?

A startup is an entrepreneurial venture or a new business in the form of a company, a partnership or temporary organization designed to search for a repeatable and scalable business model.

This correspond pretty well to our situation since we are still searching our place on the market by trying to reach as much clients as possible.

Are we an intrapreneurship?

Intrapreneurship is the act of behaving like an entrepreneur while working within a large organization.

We can say we are an intrapreneurship as well, since elCurator has been created and directed by two OCTO consultants from its begining.

And Juridically?

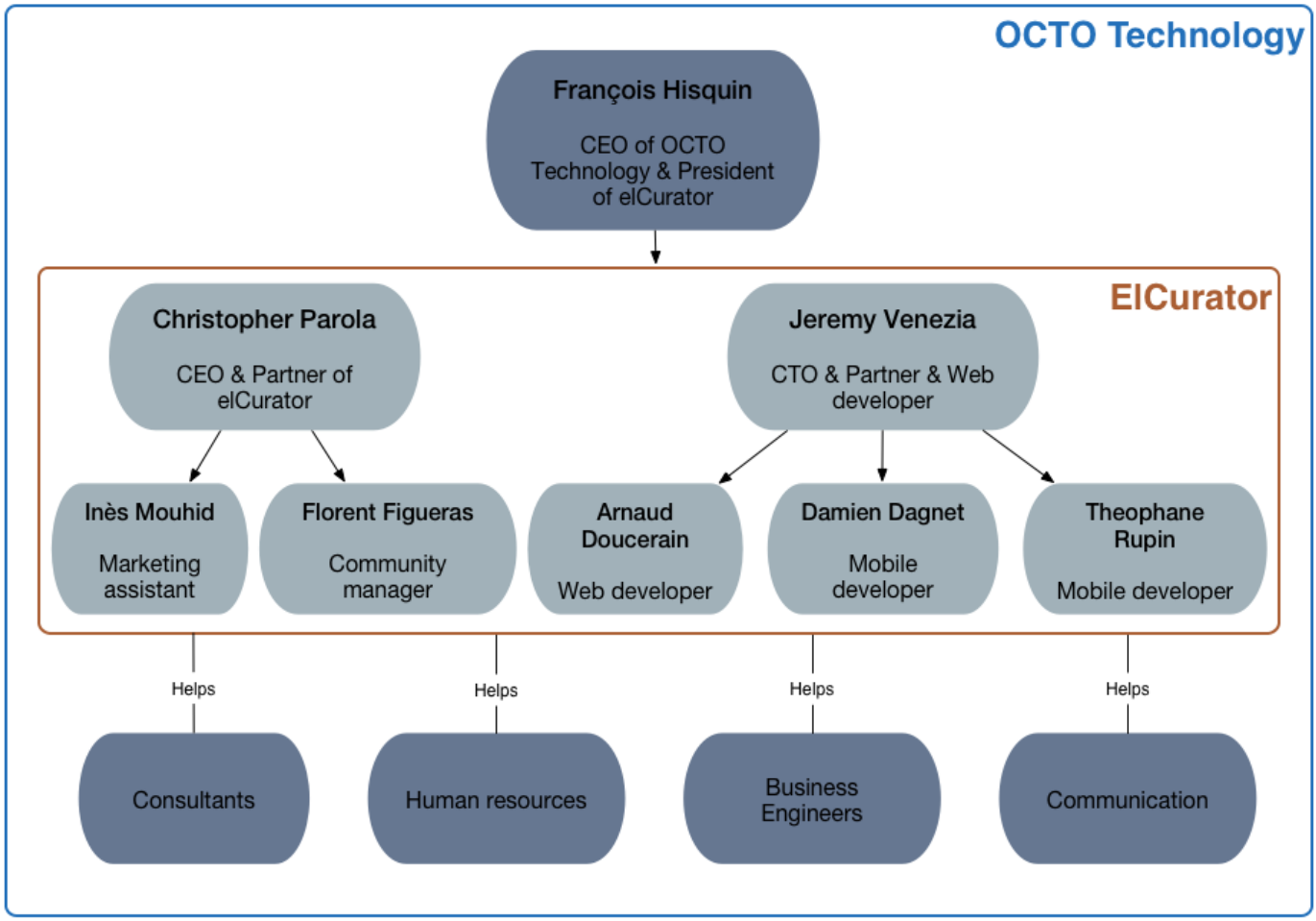
We are a **simplified limited liability company**, which is what *SAS* is actually meaning in french. We are an **affiliate** of OCTO Technology as well, since it owns more than 50% of the our capital.

Our legacy

As we just said, elCurator started living inside OCTO Technology. Its creators are two former consultants. We are all the time in relationship with other consultants of the company. This is actually a good thing since OCTO has a very resourceful community made up of experts on

many topics (the OCTO blog demonstrates it pretty well). This is obviously influencing our way of working together, our methologies, etc... It is a very resourceful legacy to us.

The team



With time, we developed three main activities around our product:

- Software developement
- Communication / Community management
- Sales

Since we are an affiliate of OCTO Technology, we beneficiate of some extra services as:

- Consulting on IT topics
- Human resources

- Business engineering
- Communication

This extra help is very important because it provides us some kinds of frameworks on which we can rely when we don't have the necessary skills. For example, when Christopher had to write a contract from scratch for our first client, he has been able to get some help by OCTO. We also get some help when we need to hire new people. And many more examples...

The product

We talked about the company's situation, but we have not explained the activity of elCurator. What are we actually doing here? As you must have understood, elCurator SAS is a software company developing and selling a product named elCurator. In this chapter, you will find the informations you need about the product to easily understand and start manipulating it. We think each members of our team should be capable of explaining the product and talking about it outside our walls.

The concept

ElCurator is a **collaborative curation platform**. In other words, our mission is to encourage workmates to share high quality contents in their company, and highlight the best shared contents.

The concept is defined by 3 main steps:

1. **Select** the best contents.
2. **Add** a short description to justify why the content is relevant.
3. **Share** it to a community.

Why our clients need us?

Since June 2014, when we publicly launched the product, we started recognizing our clients. Obviously, we are targetting medium and large businesses. Our strategy is mainly a B2B (business to business) strategy. Some functionalities are B2C (business to client) oriented, but the goal is always to be more relevant to large companies by being more famous on the market.

So far, we identified that our product is a good fit for medium digital and consulting agencies, education organizations, and large banks. For each kind of client, we noticed several needs that we are able to fit:

- **Stop infobesity**. It is nowadays a fact that we receive too many informations by mail at work. If your workmates are sending you their curated content by mail, there are good chances that you ignore it because you have plenty of more important mails to treat. We believe there should be a time for treating mails, and a time for curation in a work day, and this is exactly what elCurator permits to do.
- **Identify and highlight experts**. It can be difficult for managers to see who is expert on what topic. Thanks to elCurator, it is very easy to identify them by checking what they are sharing.
- **Collective knowledge capitalization**. The content of the company is saved and the users can peform searches and follow the platform's recommendations to navigate between articles.
- **Continuous formation**. Every workmates can easily reach expert contents and quickly start developing new skills.

The main features

If you are going to work at elCurator, you need to know what are the product main features, so you realize what are its main use cases.

Elcurator is made of several tools:

- A **website**, which is the most used platform by our users so far.
- Two **mobile applications**; Android and iOS.

User story: the web platform

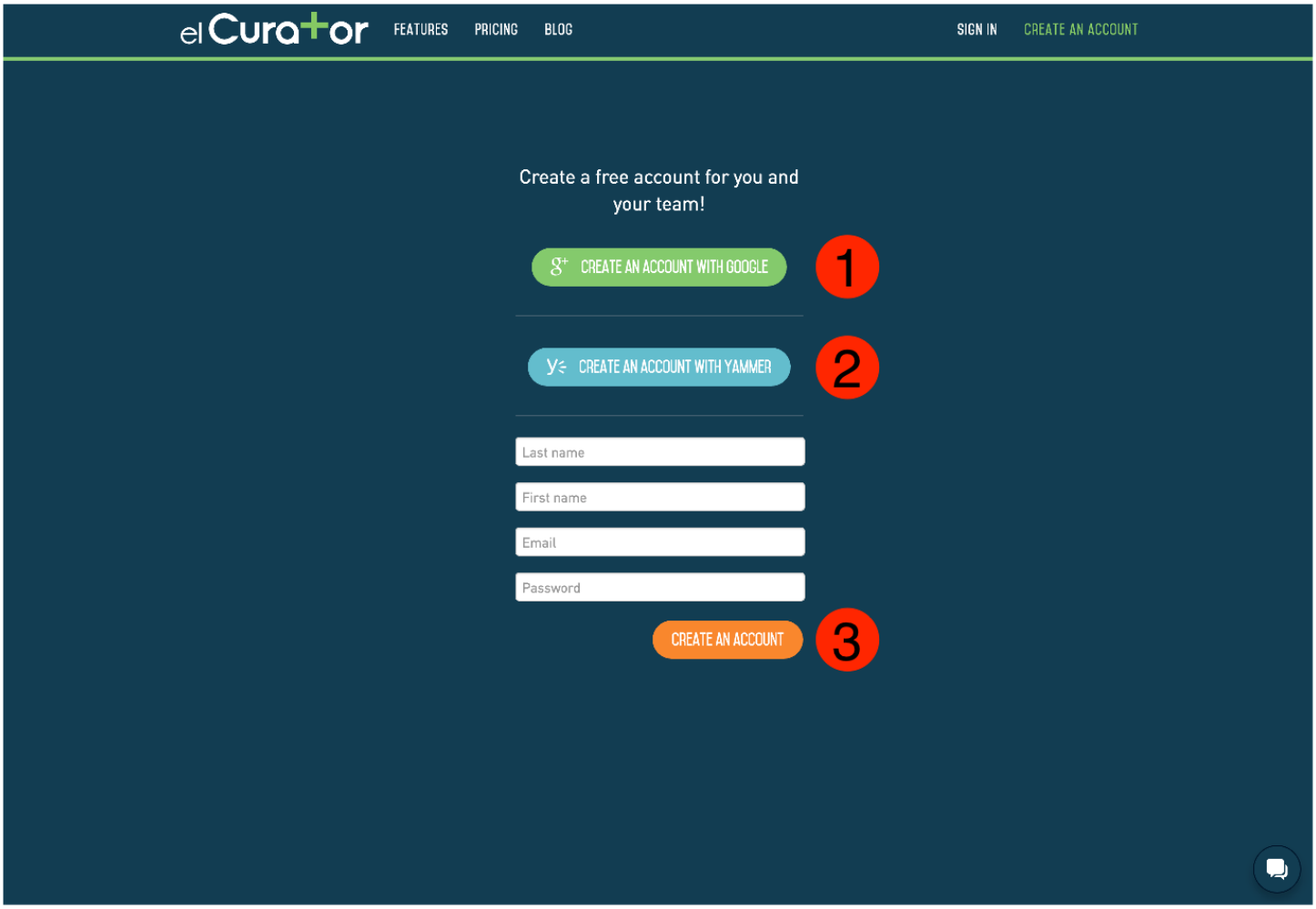
Because a features list would not be very attractive to read, let's tell a user story.

Manu is a project manager. He knows its workmates read many articles every day, but keep it for them. He is desperate because his team as a pretty high turnover, and each time someone is leaving, he knows that all his expertise is leaving with him.

Manu needs a solution. He needs to convince his workmates to **share** their knowledge together. Most of all, he needs to **store** this knowledge, so the new recruits can take advantage of it when they start working in the team.

Searching on the web, Manu types *best curation tool* on Google, and come to the elCurator's landing page. It is said; **free for one organization made of less than 20 users**. Eureka! Manu clicks on the '*create an account*'.

Create an account

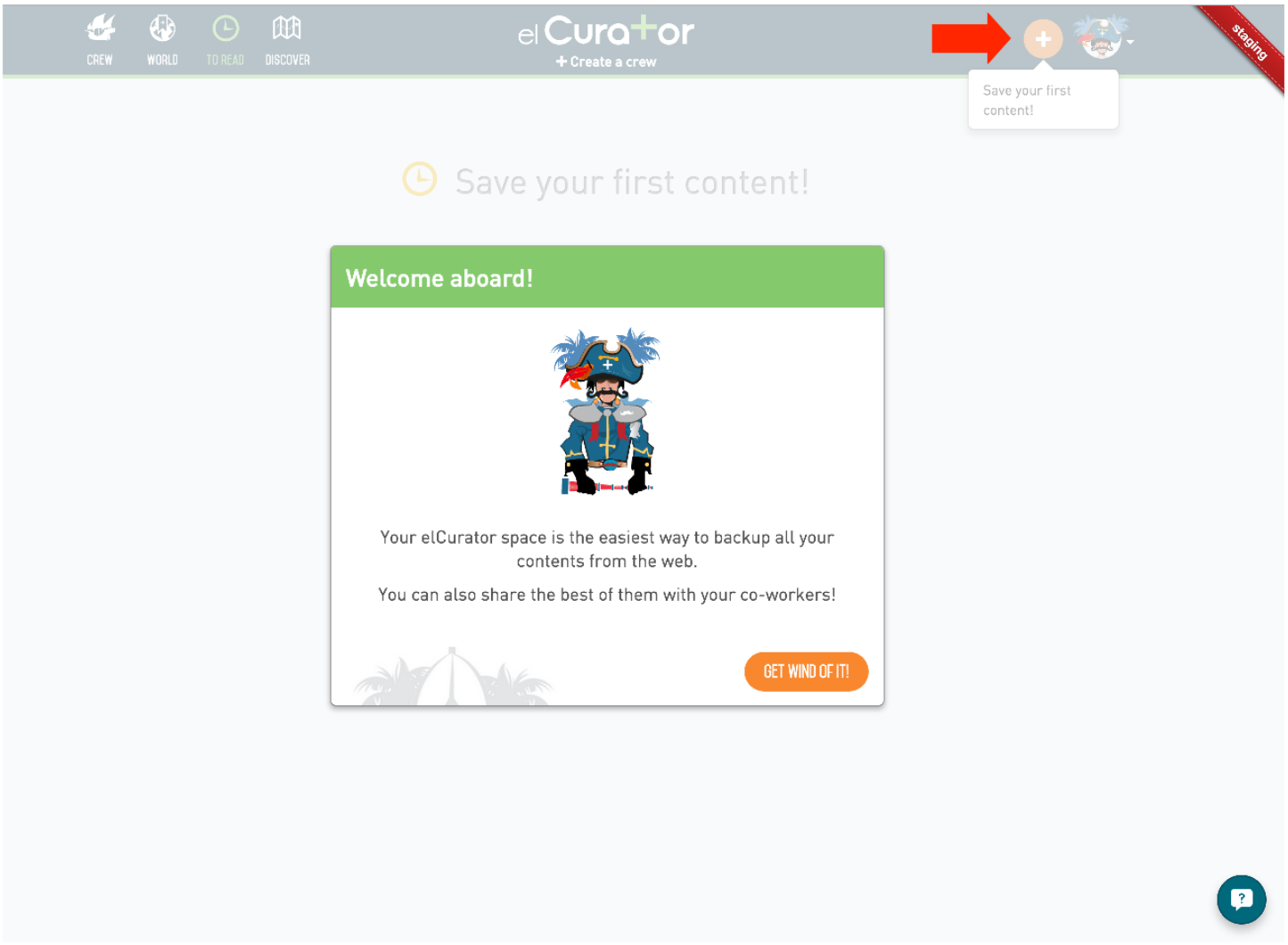


There are 3 ways of openning an account:

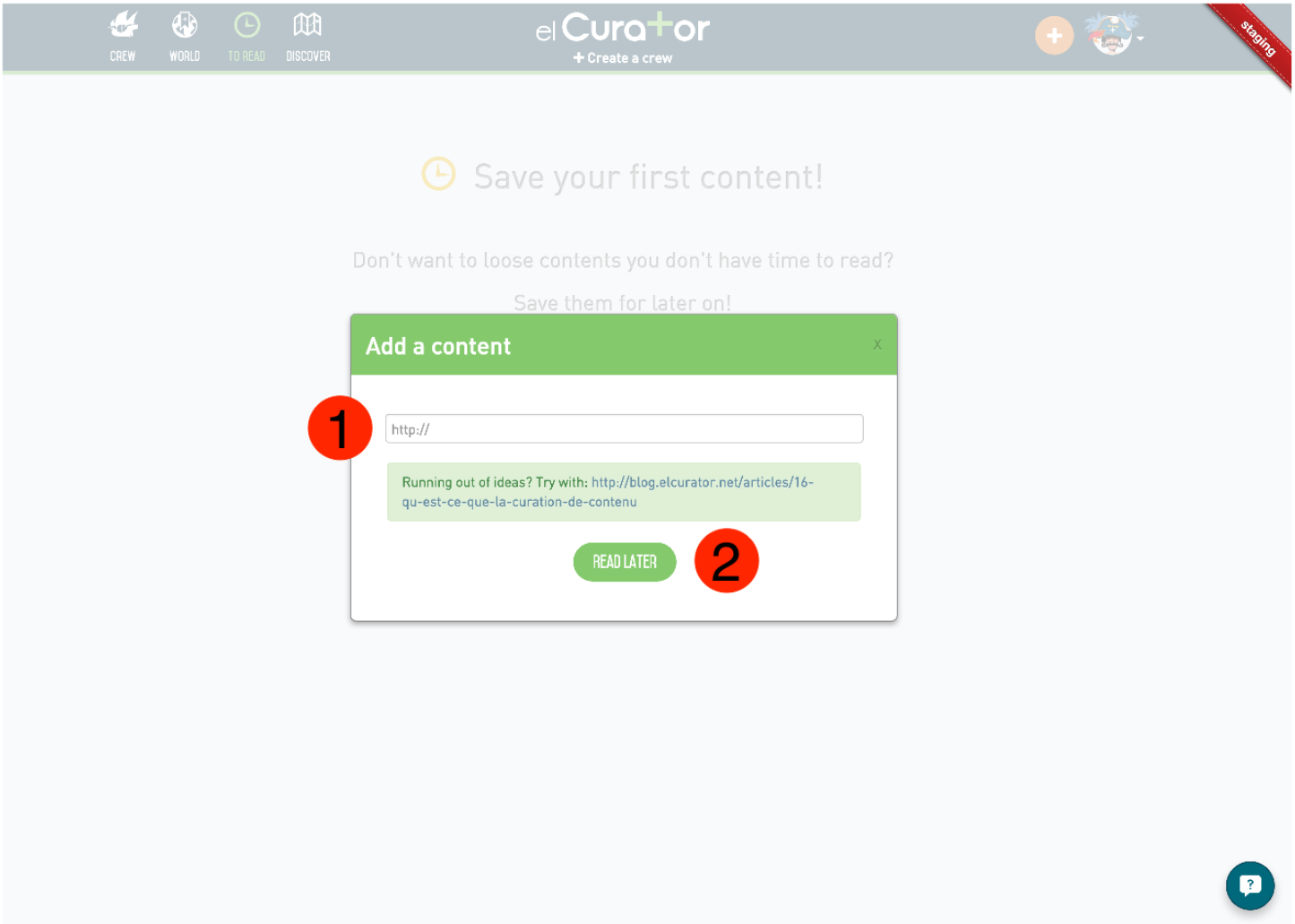
1. Using an existing Google account.
2. Using an existing Yammer account.
3. By filling a form.

Manu choose to create an account with Google.

First content



Manu then click on the + button to add his first content on the plateform.

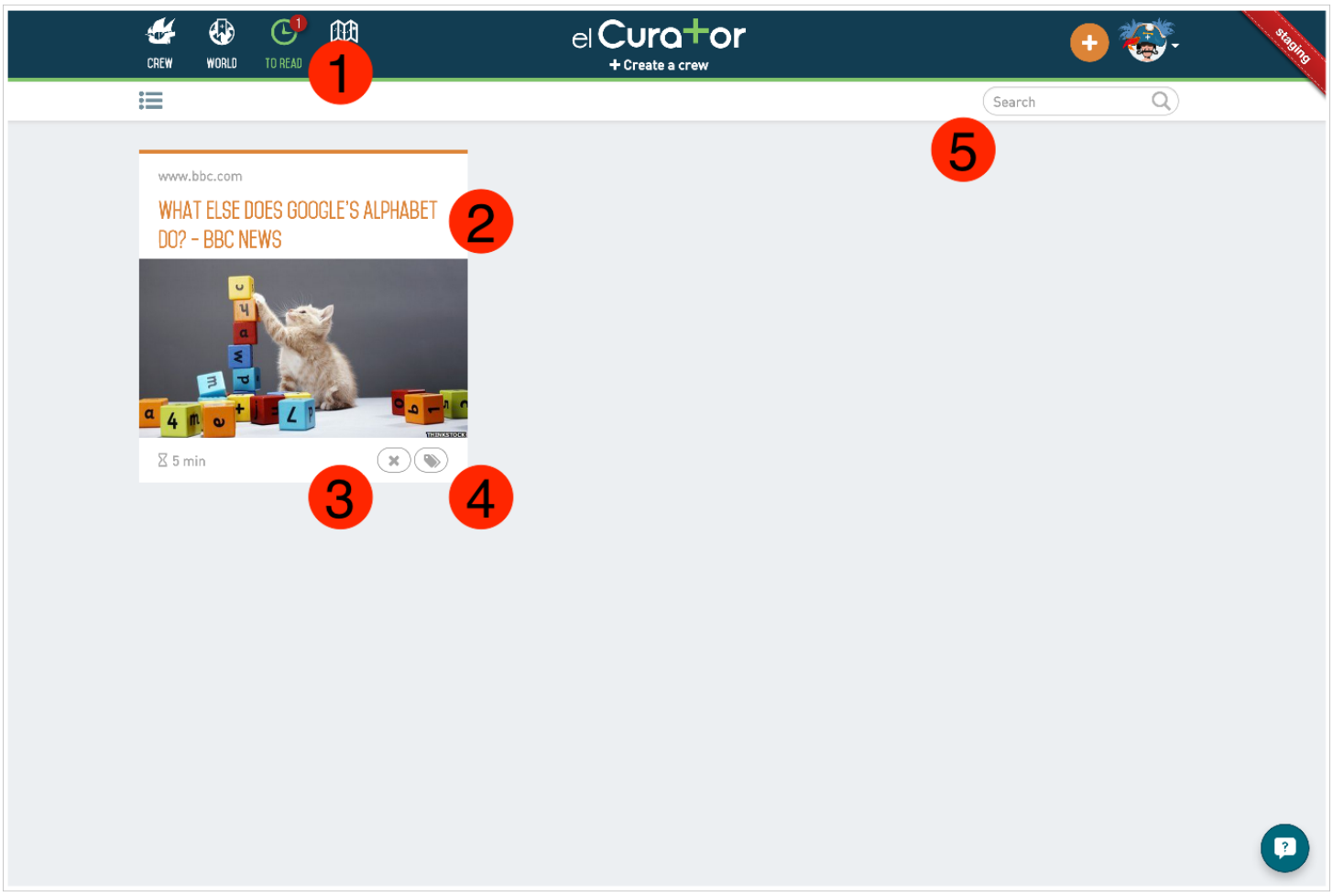


On this form, Manu needs to:

1. Put the url of the content he wants to share.
2. Click on the *read later* button to validate and add its articles into the *to read list*

The to read list

The *to read list* is a place where Manu can store his personal articles.



From here, he can:

1. See how many articles he has left to read.
2. Click on an article card to read it.
3. Delete an article from the list. It could be not good enough after all.
4. Tag an article.
5. When he will have too many articles in the list, he will be able to search into its article contents.

This list is designed to be like a todo list. The goal is to make the users read the articles they stored, then trash it, or classify it.

The reading page

Manu wants to read his article. He clicks on the card, and here is what he sees.

less than 5 seconds ago

WHAT ELSE DOES GOOGLE'S ALPHABET DO? - BBC NEWS

5 min

www.bbc.com

Google's new parent company Alphabet Inc has a vast portfolio.

When the word Google entered the Oxford English Dictionary in 2006 it was widely seen as proof that the chirpy US tech firm, with its primary-coloured logo and "do no evil" mantra, had officially captured the zeitgeist of the internet age.

On this page, he finds the article content.

Notice that elCurator applied his own design to improve the readability.

He also can:

1. Customize the font class and size.
2. Share his content by email, or on Twitter, Google+, Pocket, etc...

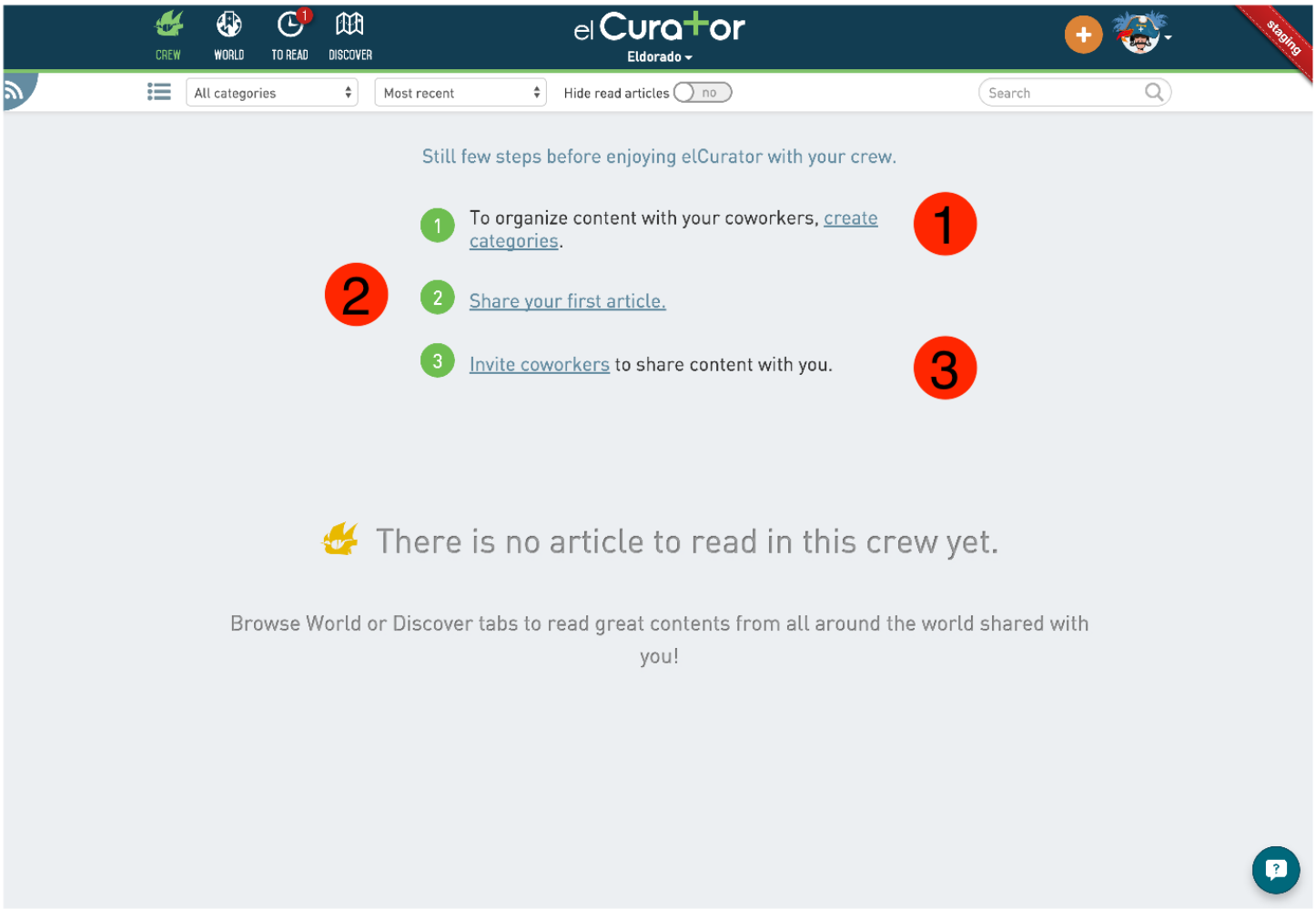
Create a crew

Manu can easily store and consult his personal articles, but now he would like to know how he will convince his workmates to share their knowledge through elCurator.

No worries, we are coming to it!

To be able to share content on the platform, Manu needs to invite his workmates. But to what? Manu finds the *create a crew* button just below the elCurator's logo in the navigation bar. What we call a crew is actually a group of collaborators. Manu creates a crew named *Eldorado*. He is now **administrator** of his own crew.

Right after creating his crew, he is redirected to the page below.



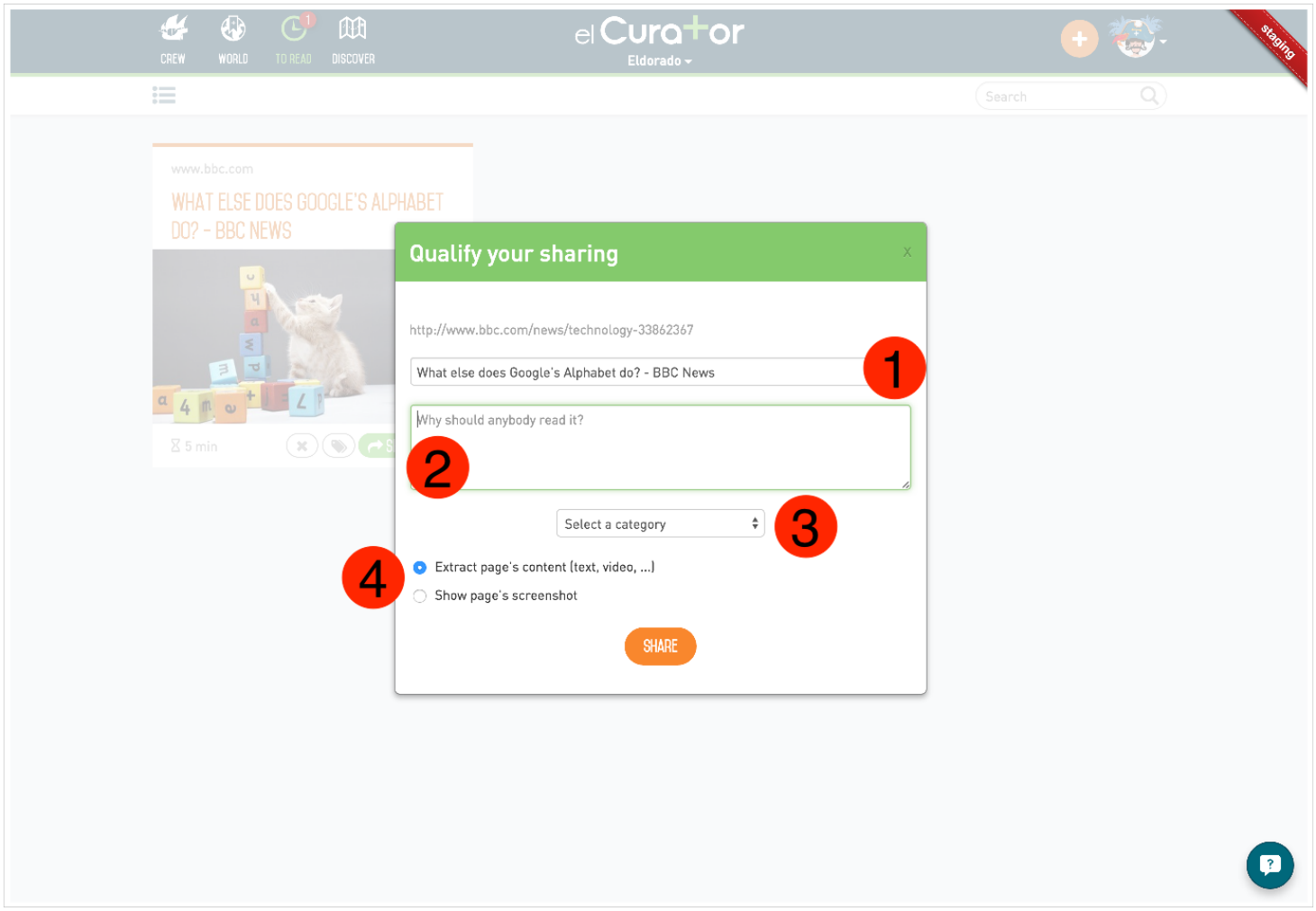
Here he discovers what he can do with his crew:

1. Create **categories**. Remember that we want to be able to easily find relevant contents shared by people who won't necessary tag their content the same way. That's why we force the administrator to create categories in order to oblige his crew members to follow an editorial policy.
2. **Share an article** with his crew members.
3. **Invite his workmates** to do the same.

Share an article to a crew

Manu creates several categories and comes back to his article to read. Now he has created a crew, he can click on the *share* button to share his article to the *Eldorado* crew.

Notice he could have done the same by clicking on the + button on the right of the navigation bar.

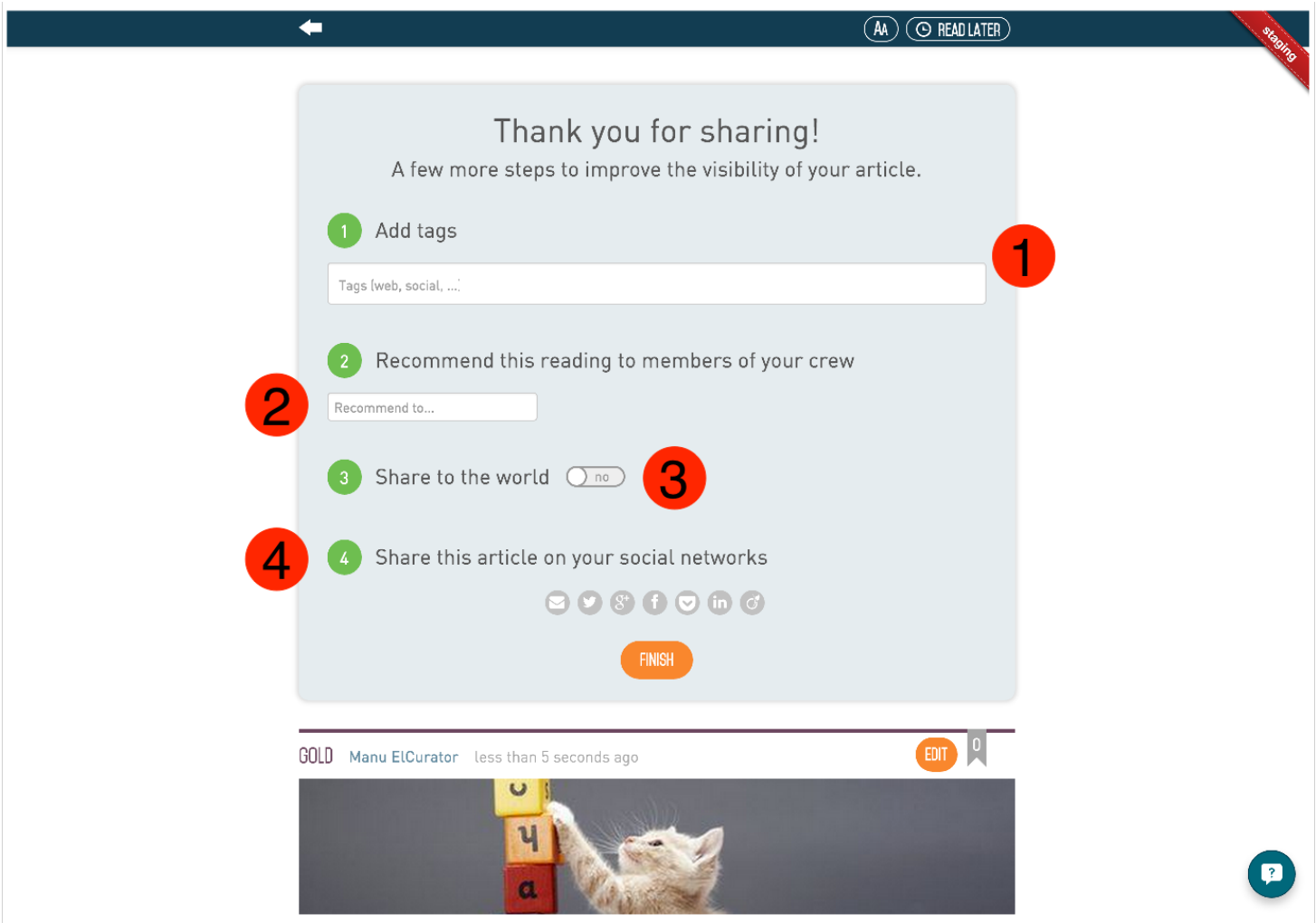


Manu has to fill several fields:

1. The article title.
2. The article description.
3. The article category.
4. Finally, he can choose to ask elCurator to extract the content as a text, or to take a screenshot of the webpage. The screenshot is usually used when you want to share a graphical content which cannot be properly extracted.

Improve the article's visibility

After having shared his article, Manu is redirected to the shared article page, where he finds the readable content plus an optional form to fill in order to improve the visibility of the arricle on the platform.



Here he can:

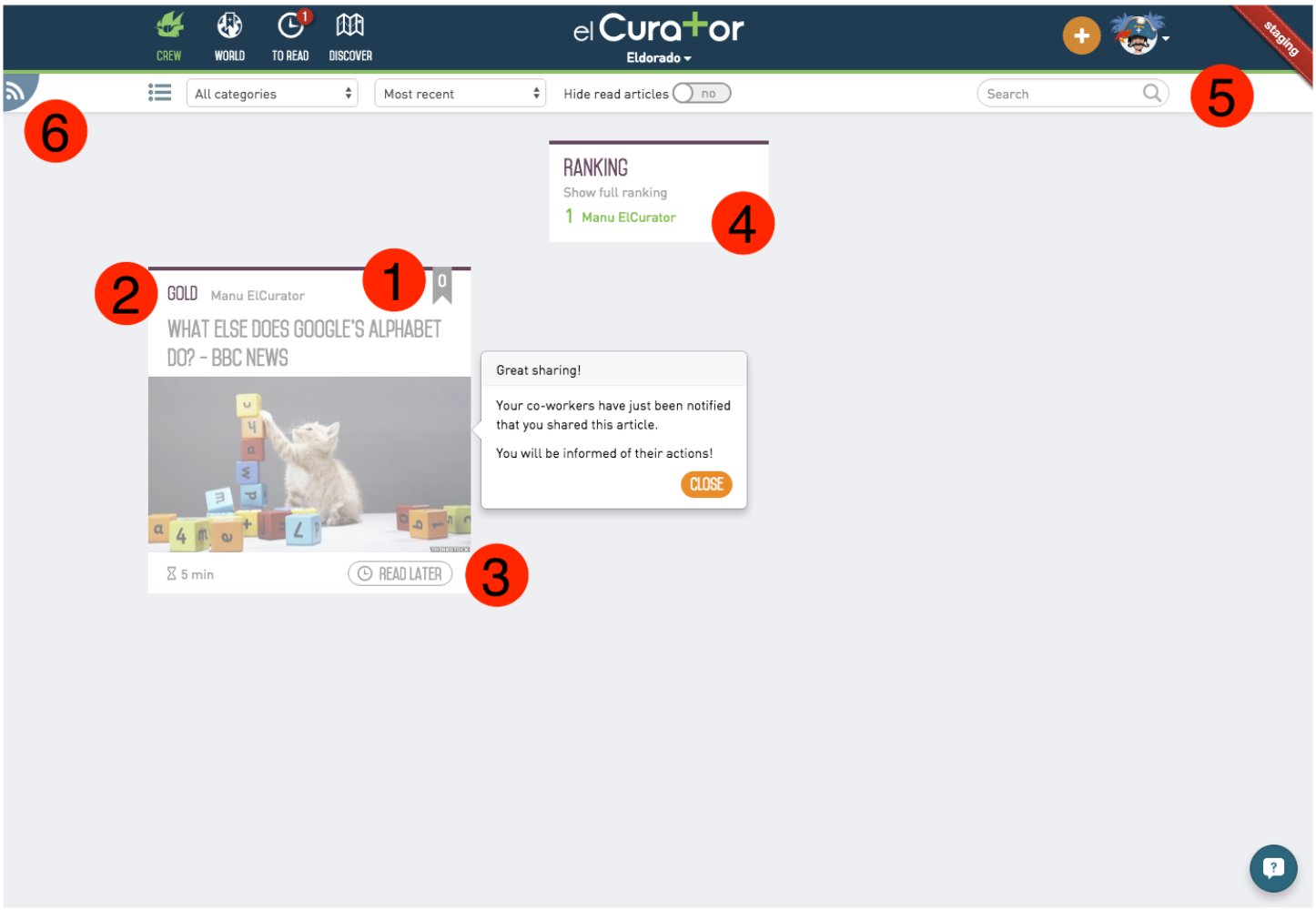
1. Add tags.
2. Recommend it to a member of his crew.
3. Share it to what we call the *world*. It is a public list of articles. When this option is enabled,

the article is still shared to the Manu's crew, but it is also visible in the *world list* by to all the elCurator's users.

4. Share it by email, Twitter, Google+, etc...

The crew's list

Manu clicks on the *back* button in the navigation bar and goes to its *crew's list*.



This page is one of the most important of the application. Here he can find back the article he just shared to his crew.

On this page, Manu notices few things:

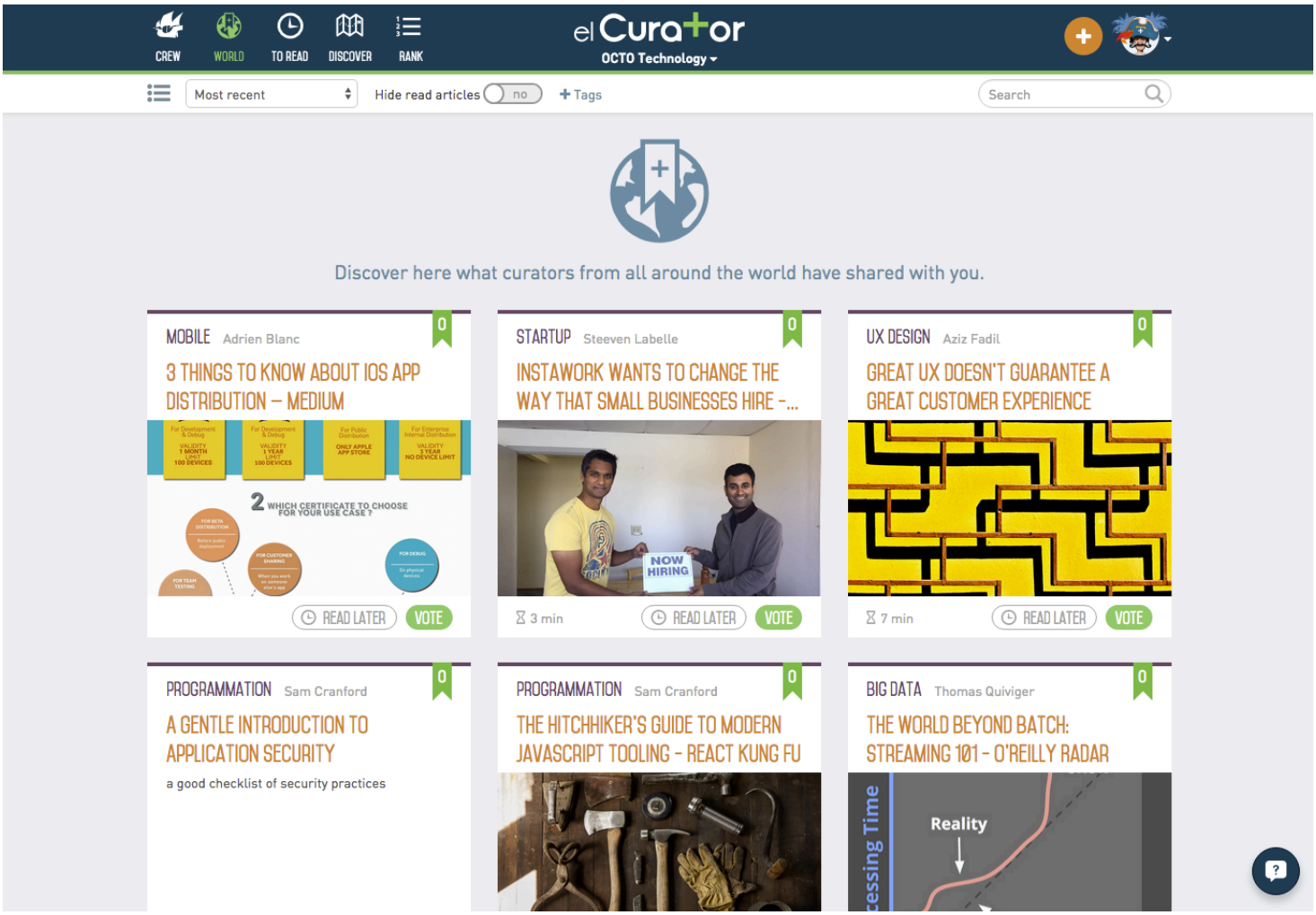
1. The *rate count*. When the members of his crew are going to read the article, they will be able to give it a point if they liked it. The number of *likes* is shown here. When the flag is green, it means the current user can rate the article, but when it is grey, it indicates he is the sharing's author, which means he cannot give a point to himself.
2. The *category*. As we said, every article must be categorized. The article's category is shown here. Each categories has a specific color.

3. The *read later* button permits to the members of Manu's crew to put it in their *to read list*. We think our users have rarely the time to read immediately the content of an article. The to *read later* button is here to encourage them not to forget it, and read it later.
4. A new kind of card which represents his rank among his crew's users. This rank is calculated with the number of rates and readers he had so far.
5. The *search bar* on which he finds a way to filter by category and keywords. He can also hide or show the read articles. And he can order the list by date, read count or popularity.
6. Manu is also using another tool to read his news feeds. He can do it by clicking on the *RSS* button which is a representation of the list using the RSS format. That way, Manu can import all his crew's content in his favorite RSS reader tool.

The world's list

Manu invited his workmates to his crew, but waiting for us to share some content, he wants to discover what the world is doing on elCurator.

He clicks on the *world* button in the navigation bar.



This list is the same as the crew's list except it only contains the publicly shared articles, and

there is a way of filtering by tags instead of categories.

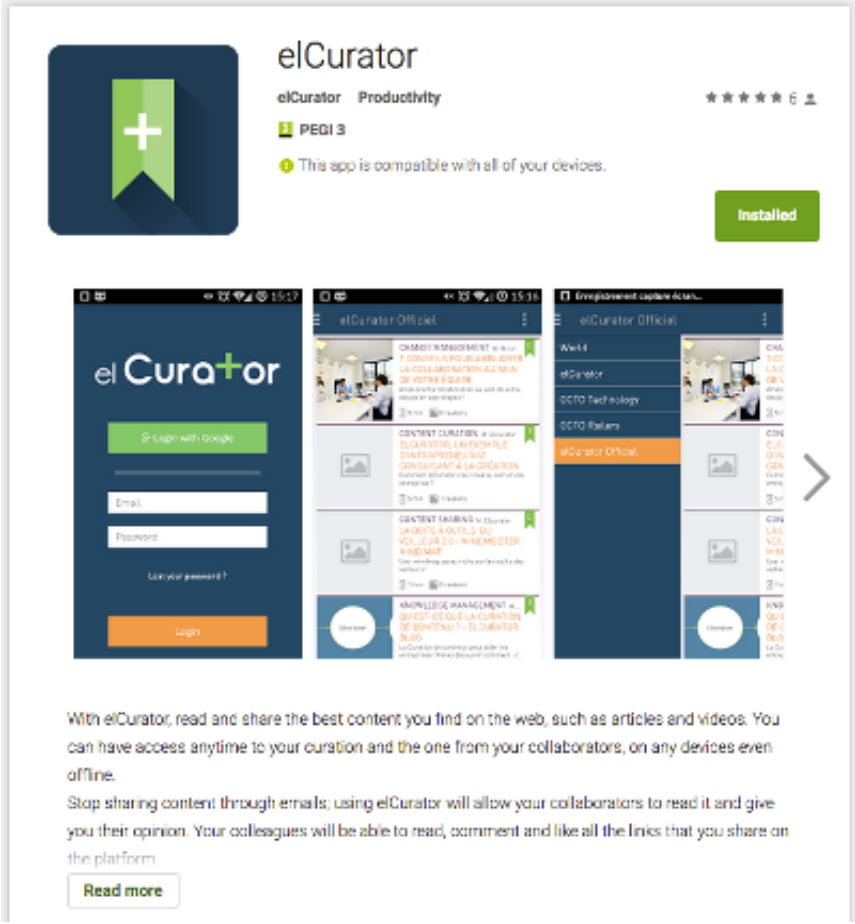
User story: the mobile applications

Note: we wont describe here the iOS's features since they are equivalent as what is implemented on Android.

Installation

Now that Manu knows how to use the platform, we wonders; what if I want to be notified on my mobile when I receive an article? What if I want to read some content in the subway?

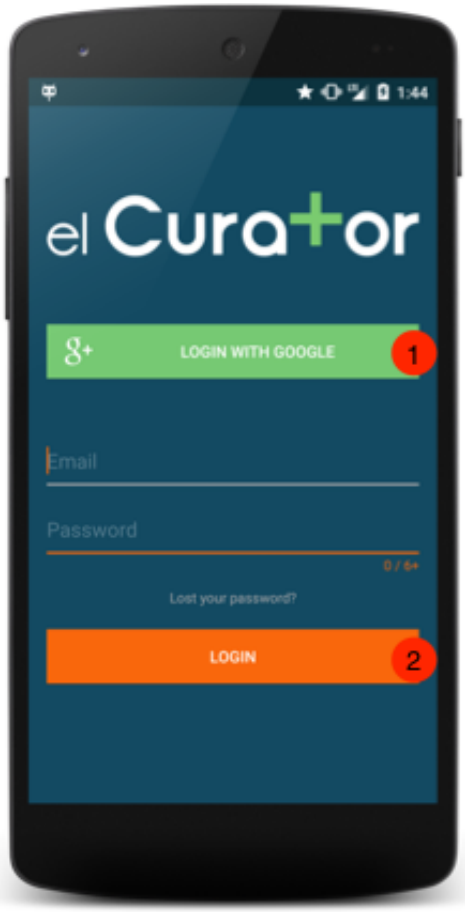
Since he has an Android phone, he goes to the Play Store and searches elCurator.



He installs the elCurator's application on his smartphone.

Login

When launching the application for the first time, Manu gets the login page.

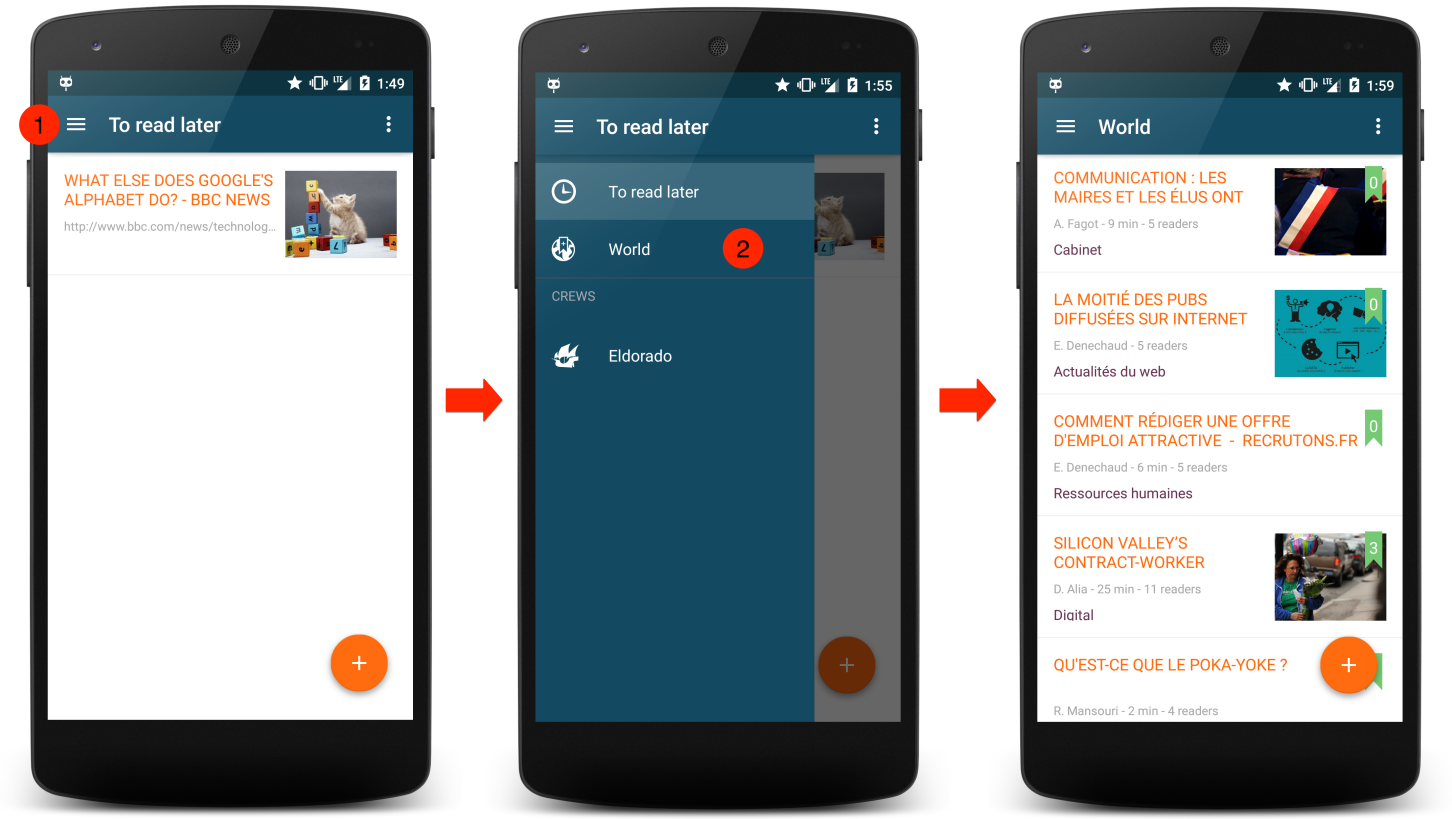


Just like on the website, Manu can login using:

1. His Google account.
2. His regular credentials.

Navigation

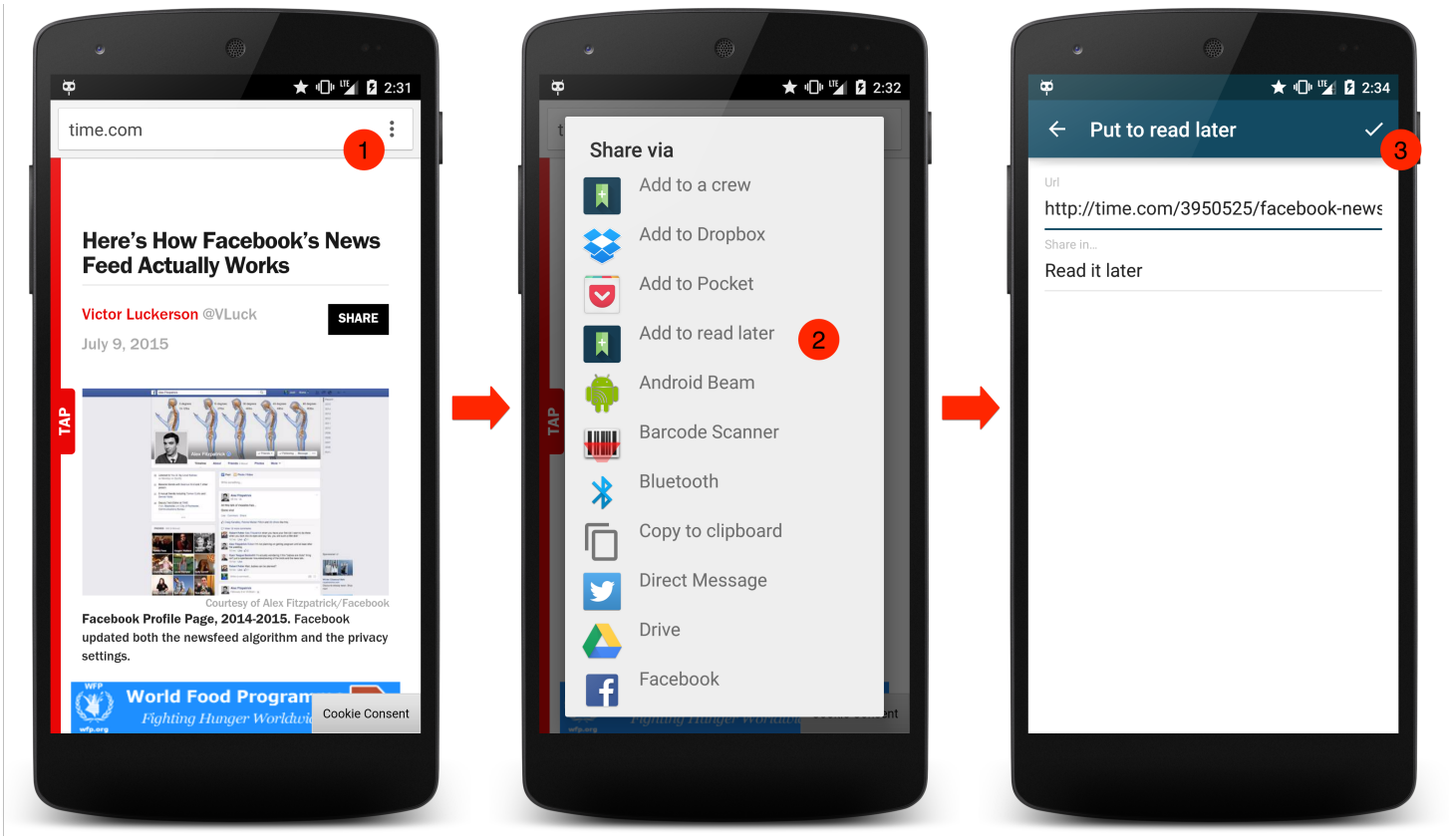
Manu signs in the application via his Google account and arrives on the *to read* list. He finds almost the same graphics elements than on the website.



Has we can see above, the application permits him to navigate to the *world list*, the *to read list* and his *crew's list*.

Add an article

Manu is now leaving his workplace. He starts reading an article on his phone in the elevator, but doesn't have much time to finish. He can put his article in his *to read list* like below.



Note: Manu could also have used the + floating button in the application to access the same form. He also could have chosen to share to his crew instead.

Offline access

Manu is now in the subway. He wants to continue to read his article in elCurator. Since all the content shown in the application is immediately accessible offline, he can find back his article in his *to read list* and continue to read without even noticing he has actually lost his network access.

Note: almost every features are also accessible offline in the mobile application.

User story: the end

Manu is now an happy manager since he has setup an engaging sharing policy in his workteam. His workmates are now able to share their knowledge together, and Manu can even see the statistics of his crew in order to measure if elCurator is efficient or not.

Our methodologies

Developing a product with many use cases like elCurator is not an easy task. We need to be focused on development quality, client support, our marketing image, finding clients, etc.

There are many topics to study, many metrics to observe, many skills to have, and we would be lost in less than a week if we were not applying few methodologies.

In this chapter, you will learn our way of getting things done at elCurator.

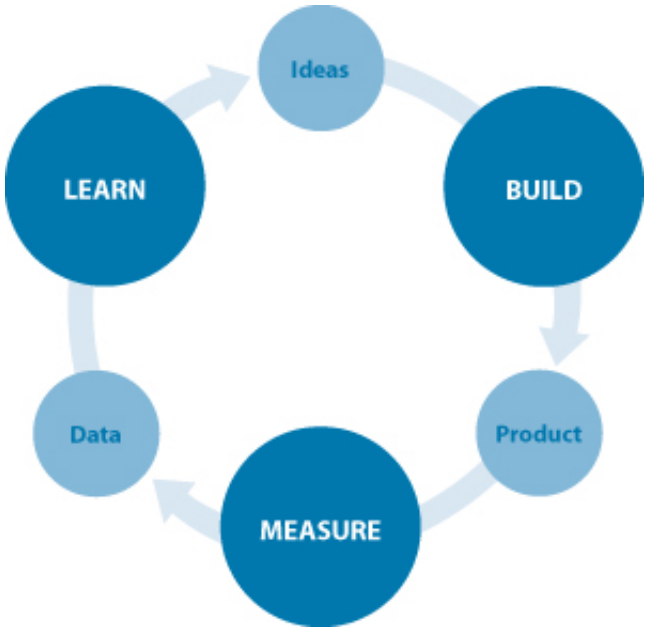
Lean startup

The lean startup is a method for developing businesses and products developped by Eric Ries in 2008. He actually based his work on The Four Step to the Epiphany, written by Steve Blank, which describes the lifecycle of a startup from the begining to success.

How we do it

We are trying to apply the following terminologies as much as possible:

- **Minimum viable product**. It is the version of a new product which allows a team to collect the maximum amount of validated learning about customers with the least effort. ElCurator actually began by a MVP which has been then improved step by step.
- **Continuous deployment**. It is a process whereby all code that is written for an application is immediately deployed into production. We are using specifics workflows and tools in order to be able to continuously deploy the code. This will be described later in this document.
- **Actionable metrics**. These are metrics specifically put by our developers in the code. For example, a metric to know how many times our users clicked on a button which gives access to a specific feature. Day by day, we can then follow these kind of metrics to observe how our product is really used.
- **Build-Measure-Learn**. It is a learning cycle of turning ideas into products, measuring customers' reactions and behaviors against built products, and then deciding whether to persevere or pivot the idea; this process repeats as many times as necessary.



To simplify things, we like to say:

- *Lean = continuous learning*
- *I don't think, I measure*

Why do we need to be lean?

We are a little team. We do not have many resources. We work on a high risk project, since we still don't know if we will be able to make monney with it. For all these reasons, we need to learn from our users to find our place on the market. We are convinced lean startup is a very efficient methodology to do it.

Agilility

Agile software development is a group of software development methods in which solutions evolve through collaboration between self-organizing and cross-functional teams. It promotes adaptive planning, evolutionary development, early delivery, continuous improvement, and encourages rapid and flexible response to change.

How we do it

At elCurator, we are trying to apply the following principles as much as possible:

- **Iterative**, **incremental** and **evolutionary** planning. To apply this, we use a *Kanban*.
- Efficient and **face-to-face communication**. At elCurator, we are used to do several recurrent meetings:

    - **Standup**. Every morning at 10am, we stand up and each teammate explains what he did yesterday and what he will be doing today.
    - **Restrospective**. Every two weeks, we meet in a room in order to observe what is going great and what is going wrong in the team. We also try to find a way of resolving issues.
    - **Team building**. For example, we assisted to a workshop where our goal was to design our own desks.
    - **Brown bag lunch (BBL)**. When we have an extra topic to discuss, we do it on our lunch time. This permits not to be only focused on the project all the time. Sometimes it feels good to see what is happening around us.

  We are all working in the same open space and everybody is almost continuously available.

- Very **short feedback loop** and adaptation cycle.

- **Quality** focus. We believe that quality code means quality product. All the code our developers are writing is reviewed by at least one confirmed developer. If the code quality related to a task is not good enough, then it cannot be considered as done.

Why do we need to be agile?

Agile methodologies permits to produce quality code, to continuousy improve our process, and always adapt our organization to the problematics. All of this is essential to discover and learn about our product ecosystem. In fact, agile and lean start-up are complementary.

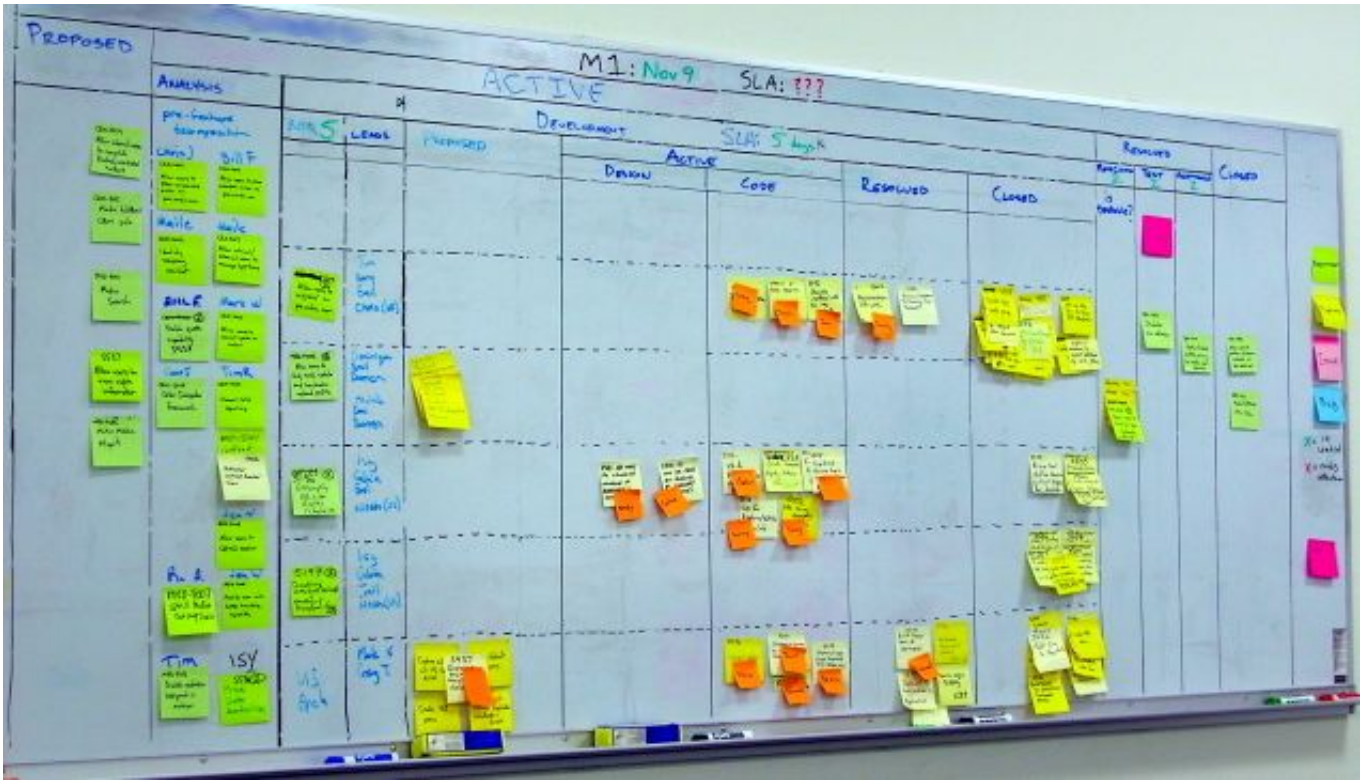
Production workflow

Kanban

Kanban is a system to control the logistical chain from a production point of view, and is an inventory control system. Kanban was developed by Taiichi Ohno, an industrial engineer at Toyota, as a system to improve and maintain a high level of production.

Columns

The Kanban is read from left to right. Each column represent a step of our process. The higher is a task in a column, the higher is its priority.



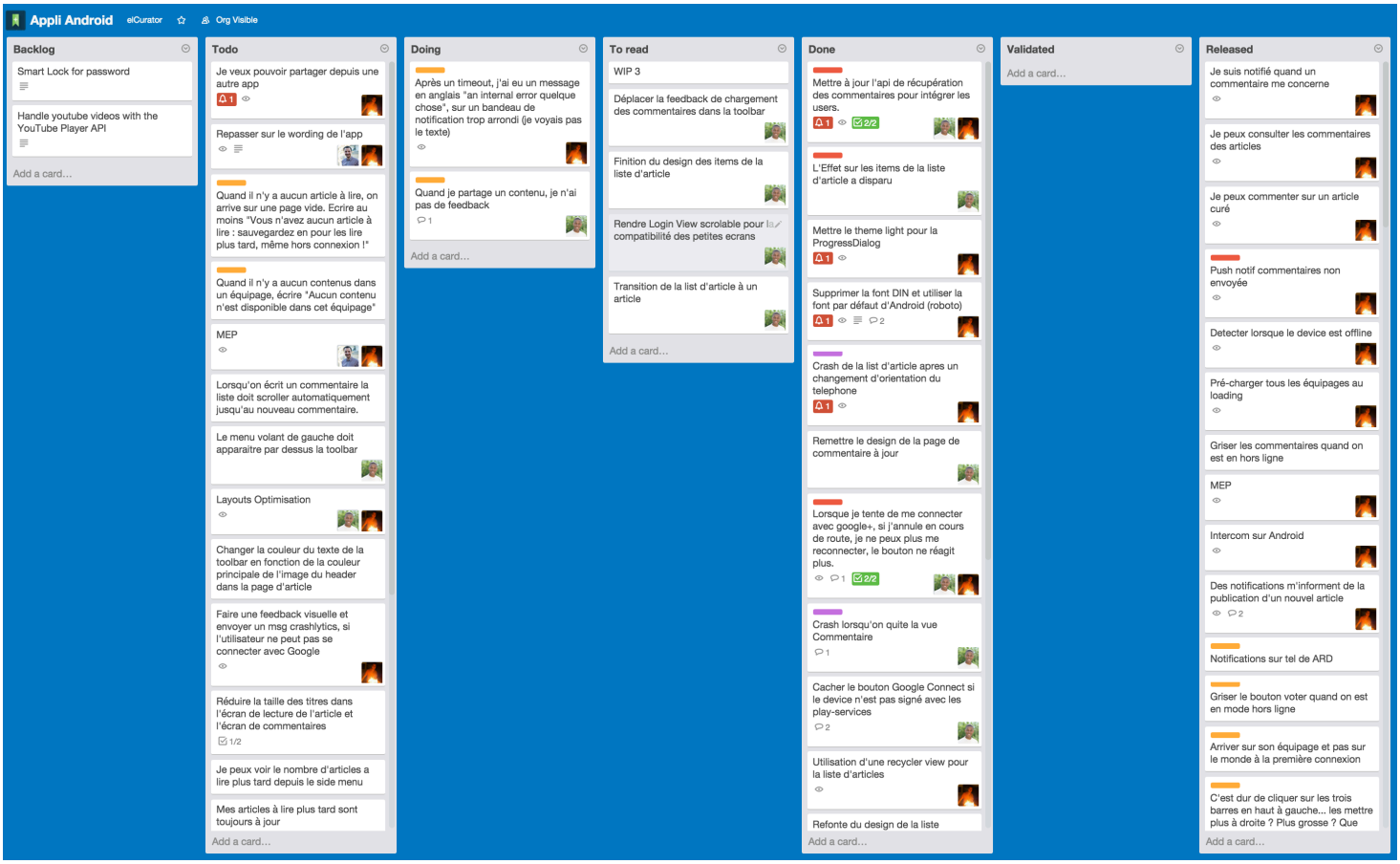
Tasks

Each task corresponds to the most minimalist feature we can describe. If we observed that a done task is producing an abnormal behaviour on the application, we put a bug or crash label on it.

After a task has been created and put into the Kanban, someone can be assigned to it. If someone does not have any assigned task, he can assign himself to the higher one in the column corresponding to its role.

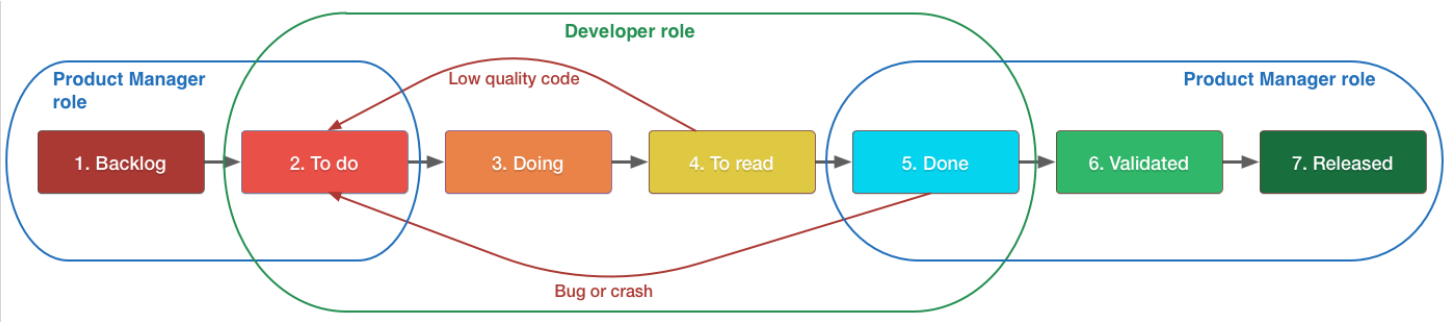
Trello, a virtual Kanban

For practical reasons, we are using a tool called Trello which permits to handle tasks the same way we would do it on a physical kanban.



From the ideas to production ready features

The chart below explains how we actually use our Kanban.



1. **Backlog**. When an idea emerges, Christopher describes it as a feature and put it in the *backlog* at the right priority level.

2. **To do**. As soon as we starts to implement a feature, Christopher takes the highest task from the *backlog*, and puts it in the *to do* column of the Kanban. If the task is too eavy, it is split into unitary tasks so it can be easily processed by the development team.

3. **Doing**. When a developer is free, he takes the highest task from the *to do* column and puts it in the *doing* column. This column references all the tasks on which at least one developer is currently working. Notice there should never be more than one task per developer here.

4. **To read**. As soon as a task is considered done by its developer, it is put in the *to read* column. This column references the code to be reviewed by at least one confirmed developer before it can be considered fully done. This step is very important since it permits to observe and regulate the quality of the project's code.

   If the code is not good enough, the task is put back in the *to do* column, and the concerned developer will have to fix it.

5. **Done**. If the code is good enough, the task goes into the *done* column. Christopher is in charge of validating the features of this column. If it is all good, it goes to the next step, otherwise, it comes back to the *to do* column.

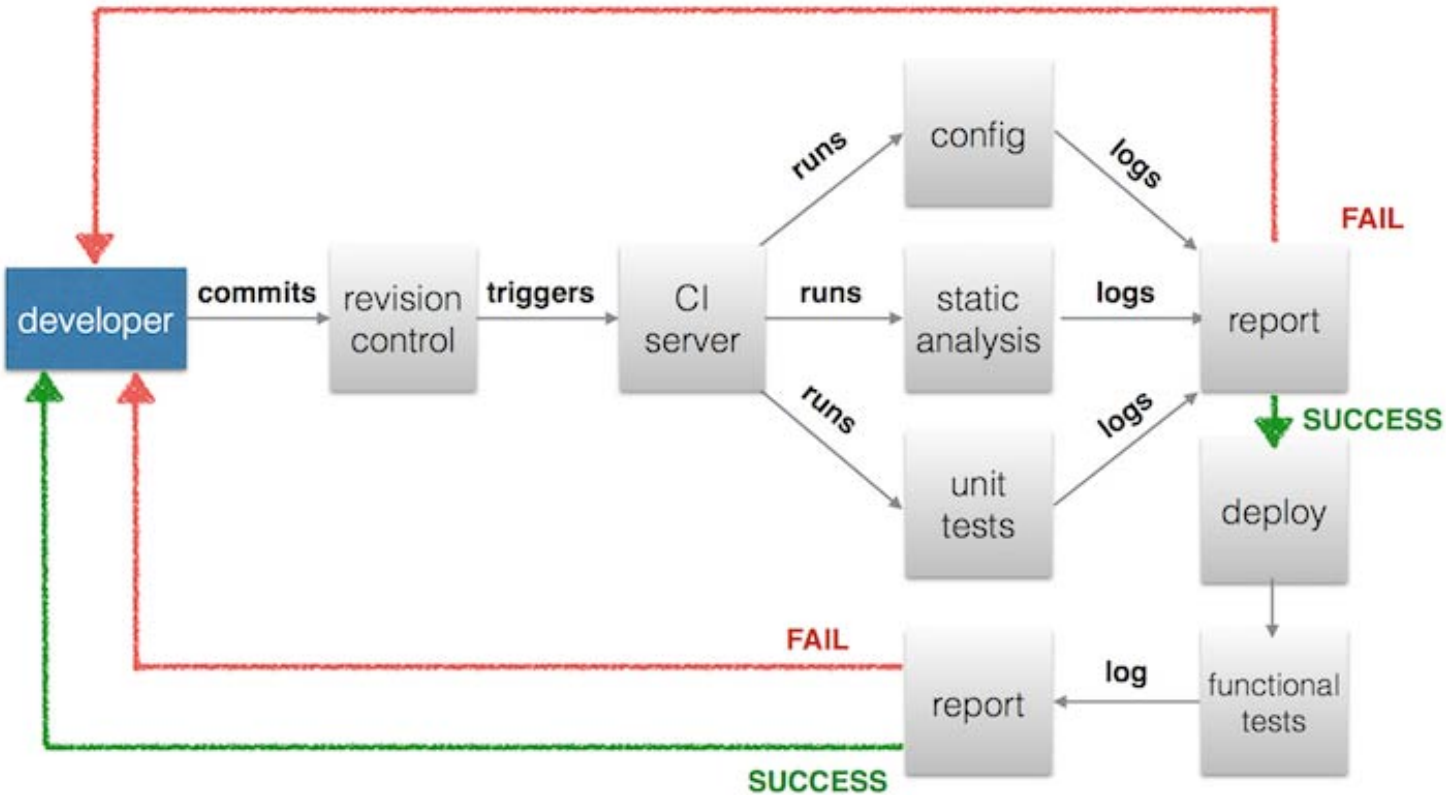
6. **Validated**. This column is a buffer where all the production ready features are stacked, waiting to be released. When a release is done, all these features are going to the *released* column.

7. **Released**. This is an historic of the features which have been released and for which the code is currently in production.

Notice how we arranged this workflow to make it as agile and lean as possible. The releases are continuous, which means we try to push our code in production as soon as a feature is ready. The validated buffer is only here in case the release is technically impossible, so the process is not stopped. Right after a problem is detected, it is immediately communicated and put backward in the process so it can be fixed as soon as possible. Of course, the workflow was not like this at the begining of the project. We improved it little by little to obtain the process we have today, and you can be sure it will evolve again in the future.

Continuous integration server

We saw that our production workflow is lean, and necessitates to be continuous. To continuously be able to push our code in production, we need to always be aware of the build state of the project. This is what the continuous integration server is doing for us.



Let's explain this chart a little:

1. The developer commits his code and pushes it on the revision control server. At elCurator, we use Git, which is the most popular distributed revision control system.

2. For each revision, Git triggers the continuous integration (CI) server. At elCurator, our CI server is a TeamCity server installed on a local mac mini.

3. The CI server is in charge of running several tasks:

    - **Configuration builds**, in order to check if they are correctly building.
    - A **static code analysis**, to check the code quality. For example, if a developer let an obvious error in his code, the code analyser is able to see it.
    - The **unit tests**, which permits to know if the project core functions are still fit for use.

4. All these tasks outputs are stored in a report. If this report contains one or more failures, the CI server send an email to our development team and stops there. The mail contains the report so the developers can investigate. At elCurator, we use a plugin directly integrated in our code editor in order to be always aware of the build state, and to interact with the logs more easily.

5. If everything is running without error, the CI server deploys the project. At elCurator, we have two deployment methods:

    - **Staging server**. Our staging (pre-production) web server is hosted on Heroku. This service permits us to easily deploy to the server with a single command line. It was easy to make our CI server deploys the project on the staging web server that way.
    - **Staging mobile packaged application**. The CI server is also configured to package our mobile application and deploy it on Appaloosa, which is a private application store provider.

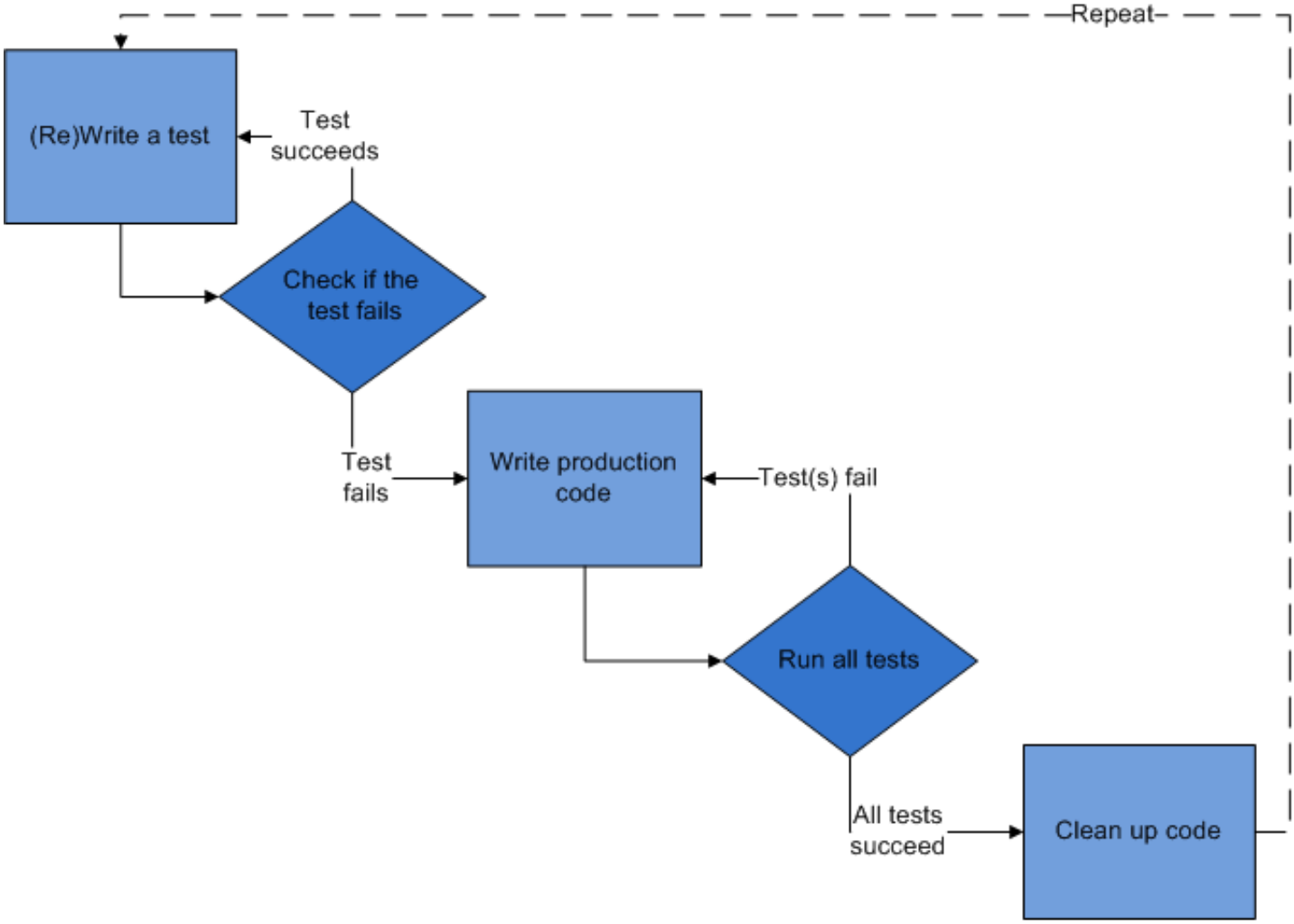
6. At this step, all the project is ready for functional tests. For bigger projects, automated functional tests are developed, so the CI server can run them. At elCurator we still do this part manually. It actually corresponds to the **validation step** of our production workflow.

Note: The staging web server and staging mobile applications are not only used for validation. They are also very useful to observe the project's state at any time. It is indeed very stimulating to be able to see what our team is producing, at least to be proud of our work, but also to be able to easily share our opinion about it.

Test-driven development

The continuous integration methodology is very important to us because this is our only way to know if the project is ready for production or not. And you must have noticed that this methodology cannot be efficient if we do not write automated tests.

In order to be sure our developers are writing tests, we encourage them to use a development methodology called test-driven development (TDD).



As we can see above, it is a process that relies on the repetition of a very short development cycle:

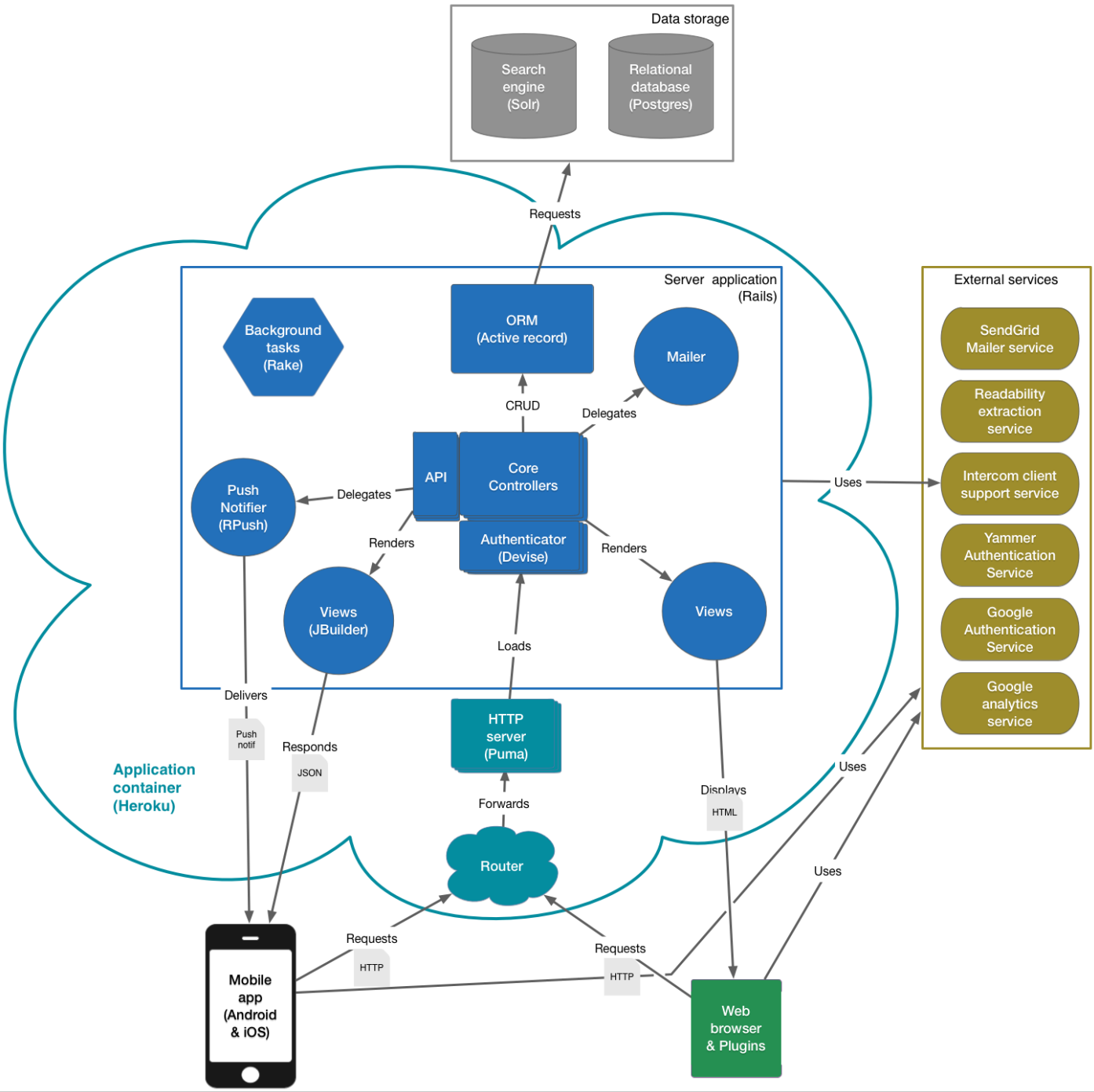
1. The developer writes an (initialy failing) automated test case that defines a desired improvement or new function.
2. Then he produces the minimum amount of code to pass that test.
3. And finally refactors the new code to acceptable standards.
4. Repeat.

At elCurator, we think automated tests are extremely important. Most of the time, if you commit some untested code, it will be rejected at the *code review* step, and an experimented developer will ask you to implement them. Of course, we are aware that it is not an easy task, and you will find the necessary resources in our team or in OCTO Technology to learn how to efficiently test your code.

Application architecture

In this chapter, we will describe the layers of components or services that are used to provide our website, our API and our mobile applications.

Server application



Used technologies

The server application architecture is based on one of the most famous web framework, *Rails*. It is a MVC framework, providing default structures for a database, a web service, and web pages.

We use PostgreSQL for the relational data storage and Solr for the search and recommendation requests.

The HTTP server is Puma, which is multithreaded and much more efficient than the default Rails HTTP server.

One of the best advantages of Rails is that it is very well integrated with Heroku which is a hosting service we use for the staging and production server. Heroku is very powerful because it permits to plug many other services to a server application and takes care of all the installations. It also permits to package a server application and make it stateless so it can be scaled on plural workers at any time.

The core logic uses several external services:

- SendGrid which is a service for sending and manage mails.
- Readability which is used to extract the content of html pages.
- Intercom which is a client support plateform.
- Yammer which is a private social network. We use it for authentication purposes only.
- Google which you must know already. We also use it for authentication purposes only.
- Google analytics which is the analytics engine of google. We use it to collect anonymous informations about our users.

RESTful API

As you can see in the diagram, our server application is not only a webserver, but also a **RESTful API**.

REST stands for Representational State Transfer. It is an alternative to web services, such as SOAP and WSDL. It relies in the HTTP protocol for all the CRUD operations: create, read, update and delete. RESTful web services are appropriated when the web services are completely stateless, limited bandwidth, when the data is not generated dynamically so it could be cached to improve performance and when there is a mutual understanding between the service producer and the consumer.

This kind of API is a perfect fit for providing the application data to ou mobile applications.

The API controllers are living in the same environment than the web controllers. The principal difference is that an API controller renders JSON views with JBuilder where a web controller renders regular views.

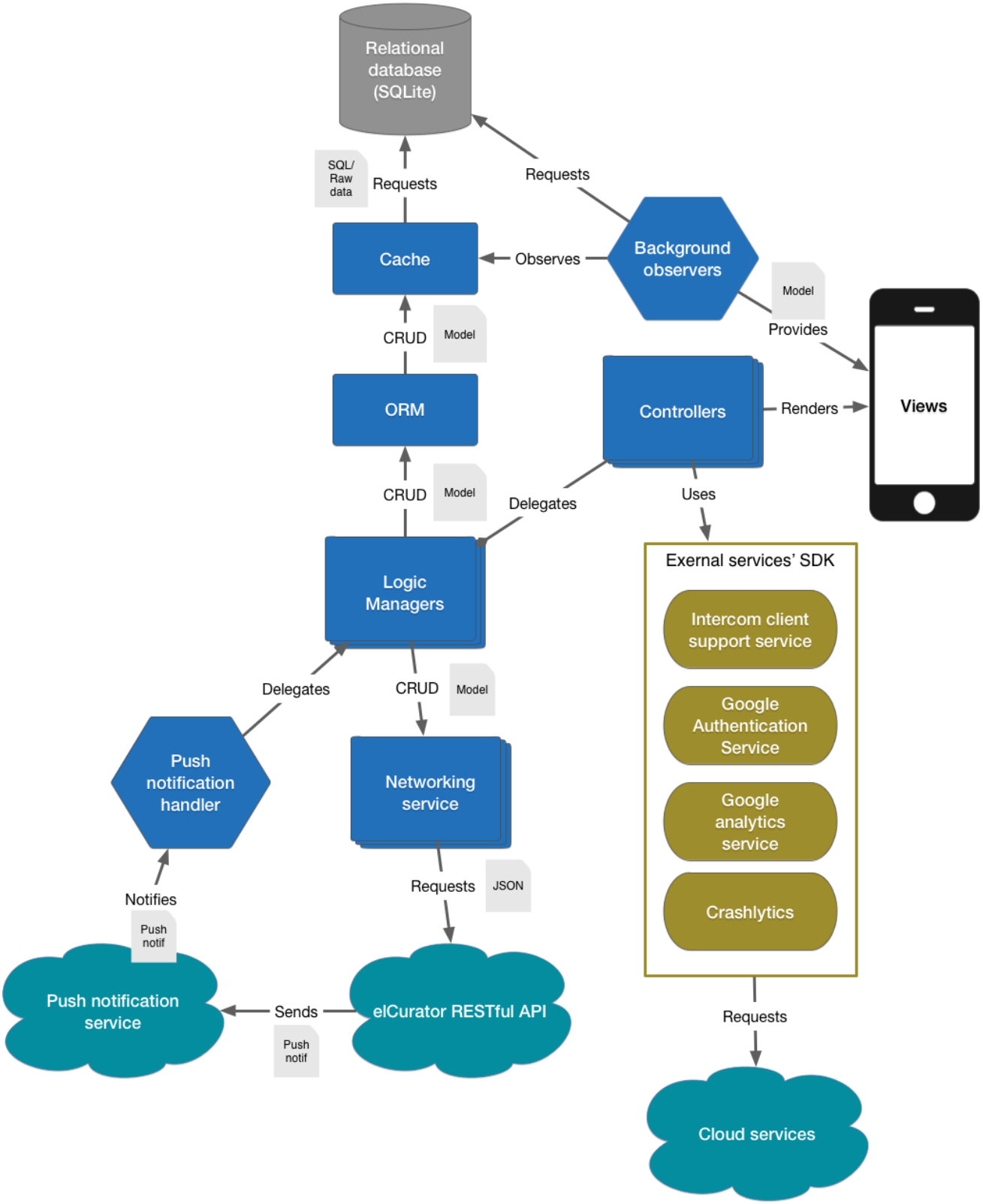
To avoid breaking our clients when updating the API, we used a version control method which consists to put the version number into the path of the API routes. For example, the articles route of the first version of our API is `/api/v1/shared_articles`.

Mobile applications

Architecture constrainsts

Let's explain the mobile application needs:

1. The user interactions should not be limited by the network avalaibilty. In other words, almost everything we can do online, we should be able to do it offline.
2. The user interactions should never be blocked because of a synchrone processing. In other words, our data processing should always be asynchronous.
3. The important application data should always be synchronized with the server, even if the user is not currently using the application.



The chart above represents how the different logical modules are interacting with each other.

We designed our Android and iOS application by following the same constrainsts, which means this chart is applicable for both projects.

Our solutions

1. **Offline mode**. In order to have the same logic between the online and offline mode, we need a local database. You must wonder why the network cache is not enough. It is because we need to apply logic on our data. We need a consistent relational storage in order to execute queries and do offline updates on it. This is not possible with a simple network cache because there is no way to know if it is consistent or not, it does not handle relations, and we cannot do offline updates on it.

   We embedded an SQlite database in our application. On top of it, we put an object relational mapping (ORM). It permits to deal with plain model objects instead of dynamic data structures like dictionaries for example. The ORM also permits to do simple create/read/update/delete (CRUD) operations on our database, which really simplify our code logic when dealing with models. For iOS, we use Coredata, and for Android, ActiveAndroid.

   We implemented a networking service, which is requesting our server in a CRUD way (since our API is RESTful). The server responds with JSON data, which is not very easy to deal with, since it is a dynamic data structure. That's why our service is using a JSON mapping in order to be able to build a model from JSON data. For iOS, the networking service is implemented using AFNetworking and the JSON mapping using JSONModel. For Android, we respectively use RoboSpice and Jackson.

   Since we can handle data coming from our API and our database the same way, we made logic managers, which are basically requesting fresh data from the server, storing it into the database, and finally doing logic with it.

2. **Asynchronous data flow**. In order to never block the UI, we needed to focus on asynchronousity. To be clear, we needed all our interface implementations to be asynchronous. The main issue with asynchronicity is data concurrency. To solve this, we used the promise pattern. This means that each service and manager calls are returning a promise of result while pushing tasks on background threads. To do this, we use the Bolts framework, which is avalaible on both iOS and Android. For more details about this pattern, this presentation explains it pretty well.

   Bolts is actually not solving all our issues. The issue which is not solved here is; how can

we be sure to have consistent data shown on the UI? To solve this, we only show the data coming from our database. Querying the databse every X milliseconds is not an option since it would be very bad for peformances. Instead, we observe the database cache, and only show the data coming from these observers. An observer isn't eavy since it is performing requests from a background thread and it does it only when the cache is udpating. For iOS, we implemented it using the NSFectchedResultsController class. For Android, we used a custom content provider.

3. **Data synchronization between the server and the mobile applications**. We implemented the swipe to refresh pattern so the data can be refreshed on demand on the user. This if great when the user is using the applicatoin, but when he lets it running on the background, it still needs the last shared articles, comments, etc... to be up to date. To perform this, we use the push notification handler, which is implemented on both iOS and Android. For example, when an article is shared, the server silently notifies the mobile applications that there is a new article to request, giving its remote id. The push notification handler then use the article manager to fetch the article data from the server and store it in the database.

Getting started

Server application

In this chapter your will find all the informations you need to getting started with the server application.

If you are not a developer, you might prefer to skip this part since it focuses on very technical aspects of the projects.

1. Clone the elCurator repository.

```
$ git clone git@gitlab.octo.com:elcurator/web.git
$ cd web
```

2. Install ruby using **rvm** or **rbenv**. See `Gemfile` file to know witch ruby version is needed.

```
$ rvm install 2.2.2
```

3. Install a postgres database.We recommend using the graphical installer. Use the username **postgres** and the password **postgres**.

4. Install gems with bundle. You may have to install bundler gem before.

```
$ gem install bundler
$ bundle install
```

5. Create the `config/database.yml` file from `config/datbase.yml.example` .

```
$ cp config/database.yml.example config/database.yml
```

6. Run database migration scripts.

```
$ rake db:create
$ rake db:migrate
```

7. You need to be allowed to the elCurator heroku app before continuing. You also need the heroku toolbelt.

```
$ heroku login
```

8. Import staging dataset in the local database.

```
$ rake db:drop
$ PGUSER=postgres PGPASSWORD=postgres heroku pg:pull DATABASE_URL elcurator
```

9. Create the `config/sunspot.yml` from `config/sunspot.yml.example` .

```
$ cp config/sunspot.yml.example config/sunspot.yml
```

10. Start a server and reindex sunspot objects

```
$ foreman start
$ rake sunspot:solr:reindex
```

11. Configure Git to prepend commits with branch name

```
$ curl https://gist.githubusercontent.com/jvenezia/57673140506ae9e330c2/raw/
$ chmod +x .git/hooks/prepare-commit-msg
```

Starting a local server

elCurator uses foreman to handle the needed processes.

The following command will run a **puma web server**, a **worker**, a **rpush** notification worker, and a **sunspot solr server**.

```
$ foreman start
```

`Procfile` file describes how foreman runs needed processes.

In development mode, forman users `.env` file to setup environment variables.

Starting the application in console mode

```
$ rails console
```

Running specs

elCurator uses Rspec testing framework.

To run all tests :

```
$ rake
```

Running specs on multiple threads

elCurator uses parallel_tests gem to run specs on multiple threads.

```
$ rake parallel:create
$ rake db:migrate
$ rake parallel:prepare
```

```
$ rake parallel:spec
```

Transferring heroku databases

To import an heroku database in your local database:

```
$ rake db:drop
$ PGUSER=postgres PGPASSWORD=postgres heroku pg:pull DATABASE_URL elcurator --ap
$ rake db:migrate
```

To push your local database in an heroku database (**never do this in the production environment**):

```
$ heroku maintenance:on -a elcurator-staging
$ heroku pg:reset -a elcurator-staging
$ heroku pg:push elcurator DATABASE_URL -a elcurator-staging
$ heroku run rake db:migrate -a elcurator-staging
$ heroku restart -a elcurator-staging
$ heroku maintenance:off -a elcurator-staging
```

To migrate an heroku database to another, pull the **source heroku database** in your **local database**, then push it to the **target heroku database**.

Known problems at installation

Nokogiri

If the nokogiri gem installation fails with bundler, install it manually with the following command before bundling again. Check if the needed version is correct.

```
$ gem install nokogiri -v '1.6.3.1' -- --use-system-libraries
```

Sunspot

If you encounter some troubles with sunspot and solr:

- The problem may be related with a sunspot_rails issue. Regenerate the solr files with the following commands:

```
$ spring stop
$ rm -fr solr
$ rake sunspot:solr:start
$ rake sunspot:solr:stop
$ git checkout solr/conf/schema.xml
```

- If you encounter `RSolr::Error` / `OutOfMemoryError` , add the following lines under `development:solr:` in `config/sunspot.yml` , and increase their values if needed :

```
min_memory: 512M
max_memory: 1G
```

- Double check by grep-ing your process list (`ps aux | grep solr`) to see if two instances are running and then shut down (`kill -9 PID`) that pid, that is not referenced by solr.pid

Android application

In this chapter your will find all the informations you need to getting started with the Android application.

Run the application

1. Clone the elCurator android repository

```
$ git clone --recursive git@gitlab.octo.com:elcurator/android.git
$ cd android
```

> **We are using submodules, which is why you need to put the `--recursive` option when cloning.**

2. Download and install Android Studio and the Android SDK Tools.
3. Download and install Genymotion. This tool will permits to setup and start using the Android simulator much more easily and efficiently.
4. Run Android Studio and click on *import project*. This should find the *Gradle* configurations files and setup a working development environment for you.

5. Start a Genymotion Android simulator or connect your Android device.

   Don't forget to turn on the developer settings of your device before connecting it.

6. Click on *build variants* in the left menu bar of Android Studio. In the left side bar, you should now see a dropdown button permitting to choose the currently used build variant.

   A **build variant** is a variant of your project's gradle configuration. If you look into the `app/build.gradle` file, you will see something like this:

```
productFlavors {
    production {
        applicationId "net.elcurator.android.production"

        def host = "www.elcurator.net"
        buildConfigField "String", "BASE_URL", "\"https://$host\""
        resValue "string", "HOST", host

        ...
    }choose
    staging {
        applicationId "net.elcurator.android.staging"

        def host = "elcurator-staging.herokuapp.com"
        buildConfigField "String", "BASE_URL", "\"http://$host\""
        resValue "string", "HOST", host

        ...
    }
    dev {
        applicationId "net.elcurator.android.dev"

        def host = "10.42.12.218:3000"
        buildConfigField "String", "BASE_URL", "\"http://$host\""
        resValue "string", "HOST", host

        ...
    }
}
```

   The elCurator application has 3 build variants: - development for which the API host is a

local address - staging for which the API host is the staging server address - production, for which the API host is the production server address.

So if you are running the elCurator server application on your local network, you can specify its address in the development build variant, otherwise, your can choose the staging build variant.

7. You can now run the application.

Run the unit tests

1. Go to the left menu in Android Studio and click on *build variants*. You should see a *test artifacts* drop down button in the left side bar. There you can choose between Android *instrumentation tests* and *unit tests*. Choose *unit tests*.

   For testing our application, we are using Robolectric. It is a library permitting to run Android unit tests directly in a JVM, which is far more efficient than running it in a simulator.

2. In the file manager of Android Studio, right click on the `net.elcurator.android` package from the `test` directory and select *run tests in* `net.elcurator.android`.

Release the application

Since the CI server is building a release package for each code version, you should never have to do it manually. Nevertheless, we prefer document how it is working under the hood so you are able to do it yourself in case the CI sever is down.

1. Go to the *build* menu in the top menu bar of Android Studio and click on *generate signed APK*.

2. In the popup window which has just been shown, choose the `app` module and click on *next*.

3. Then enter this *keystore path*; `app/release.keystore`.

4. The *key alias* is `elcuratorkey`. For the *keystore password* and *key password* ask it to your manager. Click on *next*.

5. Android Studio will generate a signed apk which you will be able to publish to the *Play store*.

Install the staging application from the private elCurator Store.

1. From your device, go to `https://1986-elcurator-store.appaloosa-store.com/1986-elcurator-store/mobile_applications?locale=fr` with Chrome.
2. Sign-in with your OCTO email address. If you haven't received the password yet, ask to your manager to register an account for you in Appaloosa Store.
3. From here, you should be able to download and install the `elCurator Staging-debug` APK on your device.

You don't need to publish this APK yourself since the CI server is already doing it for each code version.

Useful resources

You will find here a non exaustive list of resourceful documents. It is very important to us that you quickly get a preview of the topics we are dealing with here at elCurator, and we think these documents can help.

If you find that a resource is not relevant, or you think you know one you could add, don't hesitate to let us know. Your feedbacks on this document will always be appreciated.

Methodologies

- Large scale agile tips by Hervé Lourdin (@USI by OCTO)
- Slides about lean startup by Christopher Parola (@OCTOAcademy)
- Lean startup feedbacks by Christopher Parola (@Lean startup experience)
- Clean code: a book about how to write good looking code

Server application development

- Rails architecture
- RESTful API best practices

Mobile application development

Android

- Google material design specifications
- Official android training courses
- Learning guide

- Best practices for the beginner
- Awesome ui libraries list
- Another awesome libraries list
- How Facebook improved performances on Android with flatbuffers
- Android project template for testing

iOS

- Learning guide
- Best practices for the beginner
- NSHipster: last iOS developer news

Thanks for reading

We hope this document has been useful. We are very excited to have new people in our team. The fact that you read this entire document demonstrates you are motivated. It will probably be a good experience to have you with us then.

As you must have read before, don't hesitate to give us your feedbacks about this document, so we can improve it.

Part 2

This part consists of writing an email addressed to my training supervisor in order to convince him to integrate me in a new project team.

Context

The project I am asking to work on is called *CarValley*. It is a carpool application willing to be used by people working at Palo Alto, but living elsewhere in the bay area. This project is special because it will be initiated with a team already working in San Francisco. They already took the market temparature and they are very enthusiastic about the product value.

Email

Me to David Alia
08/20/2015

Candidature, mobile deloper, CarValley

Hello David,

I hope everything is going well in California.

I have just read the mail you sent on our mailing list in which you ask for motivated developers to work on the *CarValley* mobile application.

As you already know, I am very interested by this idea from its begining and I think it is wonderful that the OCTO's administration finally decided to start its production.

I am currently finishing my internship on the *elCurator* project where I initiated their mobile applications production. I know pretty well Android/iOS and their environment, which means I could make myself very useful to help developing *CarValley*.

My end of study project was to develop an application backend with high data concurrency problematics to solve. I think my backend skills could be appreciated for a project like *CarValley* as well.

We are using lean startup and agile methodologies at *elCurator* to develop the product, which means I know the basics on this topics too. I suppose the product will quickly have to evolve depending on the first users' feedbacks, and I believe these methods can be very helpful to do so.

I really loved to be working on a challenging product and I would like to continue on this way. Being a developer on a project like *elCurator* or *CarValley* is not only technical. It needs the whole team to be involved in the project life. This is what I find the most stimulating.

I also lived in California for 8 months during my studies, and I know pretty well the bay area. By the way, some people working there told me they would use a tool like *CarValley* if one were existing. English is not an issue to me. I actually like to communicate in english. And of course, I would really enjoy discovering this beautiful area a little deeper.

Even I don't have so much professional background, I still have a large range of skills and I think you will need fully operational people like me to quickly get results on *CarValley*.

I am fully available to talk with you about my candidature.

Best regards

Théophane Rupin - Mobile Developer
OCTO Technology

50, Champs-Elysées Ave
75008 Paris, France
tel. +33 6 43 28 37 65

Part 3.

This part constists of writing an email addressed to a high placed manager of the company I am working for in order to convince him to give me a project's full responsibility.

Context

This time, I tries to convince the CEO of OCTO Technology to give me a budget in order to produce and start the *CarValley* project in the San Francisco bay area.

Few weeks before I send this email, we had a brown bag lunch (BBL) about intrapreneurship. A BBL is a meeting organized at lunch time in order to discuss about a topic. At this occasion, I talked with him about the affiliate OCTO started at San Francisco. He explained to me it was complicated to find interesting clients there because they are used to invest in promising startup teams, and they do not believe in IT service/consulting that much.

Email

```
Me to François Hisquin
08/20/2015
```

And what if we started a startup owned by OCTO in San Francisco?

Hello François,

I hope you are well.

I do not know if you remember the dicussion we had at our last BBL. Anyway, it was very interesting, and I thank you for that.

You talked about OCTO Technology's difficulties to find interesting clients in the San Francisco bay area. You also mentioned that it might be because big companies there are more interested in young promising start-ups than IT service and consulting company.

I agree with you on that point, IT consulting is probably not the best way of doing business in the bay area. They are certainly already convinced (and they are probably true) they know how to build profitable products.

I have been thinking a lot about this topic then, and I thought of a probably better business model for our local affiliate. Why not producing highly valuable products by using the local ecosystem? When I studied there, something catched my attention. Everybody there is interested in the last hightech services getting on the market. I am not saying people are easily convinced, but they are a lot more open minded and communicative about hightech products than in Paris.

I would not write this without a nice product idea to propose. When I was in San Francisco, I also noticed that many people were working at Palo Alto (where all the big IT companies are) but living in the bay area. Most of them are forced to take their car every day to go to work, which is long and annoying when the trafic is dense.

A friend and I developed at the time an application called *CarValley* to provide to these people an easy way to organize themself in order to go to work together. We thought of a sort of *BlaBlaCar*, designed for people working in Palo Alto. Unfortunately, we had to come back in

France before launching the product.

If you let me go there, I would be able to sell this product by directly soliciting the big companies like Facebook or Google who love this kind of ideas. It would of course be an intrapreneurship, like OCTO already did for several products here in Paris. That way I could profit of the OCTO's structure and human resources to create a strong team. In compensation, I would give you results.

I am convinced this project would be a nice move for our affiliate. Even if we consider that we do not get enough results to continue the product in one, or two years, we will have created a strong business network around it, and I really think this is how people do business there. Talented profiles start a project, talk and show around them what they can do, then, even if the project is not making enough money, they still have a powerful professional network permitting them to start another business with each time more chances on their side.

I am fully available to talk with you about this proposal.

Best regards

Théophane Rupin - Mobile Developer
OCTO Technology

50, Champs-Elysées Ave
75008 Paris, France
tel. +33 6 43 28 37 65