Workplace Internship Report - INF6972S

Full stack developer and data analyst

DERVAUX Gautier

1971606

May 6th 2019 – August 23rd 2019

Professor in charge: PESANT Gilles – Department of Computer Engineering and Software Engineering

Supervisor: ILSON Julian – Founder and CEO

University: Polytechnique Montréal - 2900 Edouard Montpetit Blvd, Montreal

Company: We3 - 2048 Rue Harvard, Montreal

Table of contents

Table of illustrations

Introduction

As a master degree student, I've had the chance for the last 4 months to be part of the start-up We3 as an intern. Based in Montreal I closely worked with Julian Ilson, CEO and Emanuel Peter, CTO as well as Gleb Zaitsev a full stack developer in Ukraine. From full stack developer to data analyst, I was offered a great opportunity to develop my technical skills in an industrial environment. Moreover, through many discussions on the company, the users, the business, I also learned a lot in terms of product design marker fit and many other business critical notions to take into account when founding and developing a start-up. The core focus of my internship was to improve user's experience. First to try and do that, one needs to fully understand the business. Then I'll explain how I worked towards achieving this objective first as a full stack developer and then as a data analyst. Finally, I'll elaborate deeper on my experience on how most of my objectives were met by the end of my internship

I. The Market and The Company

A. The market

Even though, recent generations are connected almost all the time, meaningful social interactions become harder and scarcer according to (Maija Kappler, 2019). This short documentary, breaks down the problem like so. The first observation is that loneliness rises among young adults. It's harder to have and keep a close ring of friends which can be essential to combat loneliness. The more alarming observations also raised by (Kirkey, 2019) is the fact that loneliness is a health issue. Stress and depression are awfully damaging and lead to health problems. Meeting new people can solve those issues, but people tend to avoid awkward social interactions that could potentially lead to friendship. Even though almost everyone has Facebook friends, Facebook has never been a place to meet new friends to bond with. That's where We3 comes in.

B. We3

We3 is a start-up founded in February 2017 by Julian Ilson, my supervisor and Emanuel Petr. Its main activity is developing a free app that tackles the problem stated above. It enables people to get in touch with nearby person by groups of

three. By the time of this report, the app allows to match only with people from the same gender. The core value that We3 brings is that a user is matched with other compatible users that share the same interests or the same ideas.

To do that, a user goes through eight series of questions called levels, to build an accurate portrait of the user's personality. These, around hundred, focus on key aspects of a person:

- Demographic: relationship status and others
- Goals and Motives: does the user look for people to do exercise with (for example)?
- Personality: Is the user more optimistic for example?
- Beliefs and Values: Religion, beliefs and views on it etc.
- Lifestyle

Those questions sometimes come with choices as well. A question could be: 'Do you want to start to exercise?' Would the user answer yes, a pop up would appear letting the user choose which sport among a list they want to start. Moreover, to make the experience more personalised, some questions trigger other questions to get more personalised questions for the user and a more accurate profile.

They are two ways of meeting people through the app. Once you've answered enough questions and other technical requirements are met such as created an account, gave your location and so on, you can search for a tribe. The search succeeds when two other users are found for your tribe. You can therefore choose to start this tribe or pass it. Starting this tribe cost a certain number of stars that a user gets by answering the questions or buying them through the app. At this point, the two other users are invited through notifications and have twelve hours to respond or be replaced if possible. Invitations, unlike starting a tribe, are free. That's the general idea of the app. Users are, when no problem occur, unaware of the whole system behind. Let's dive into this system.

C. The technical context

The system behind the mobile application is mainly made of two parts: the application on the phone called the front-end, and the back-end. It's an almost identical system as web pages. The user interacts with the app on his phone that send requests to the back-end. The back-end is hosted in a cloud platform called Heroku. The API or web server receives user's request, when necessary, authenticate the users and if needed, interacts with the database also in Heroku as described in Figure 1. Many challenges need to be tackled at every step.

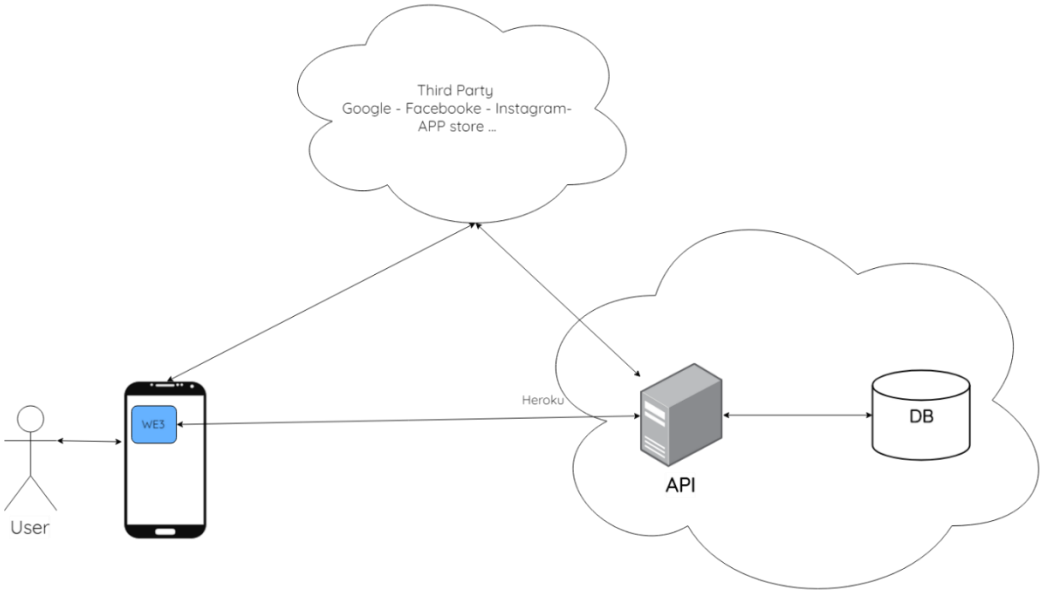

Figure 1. App architecture

Unlike web pages, mobile applications need to account for highly diverse devices. One disparity is the operating system: IOS or Android. Both have very different app ecosystem for example. To launch an app in the IOS app store, one needs to go through a review performed by apple that will authorize or not the app to be available in the app store. Developing the app for both OSs can be a challenge. Nowadays there are two approaches: code the app twice on native languages: Java for Android and Swift for IOS or use a higher-level coding language and use a framework to build the app on both platforms. The first approach is the most efficient in terms of app performance but adds complexity to develop and maintain two different codes that should be identical in terms of user experience. The second one saves a lot of coding time but may reduce app efficiency and doesn't fully abstract both platforms. As a start-up, We3 chose the quicker approach. The app is developed with Ionic and Cordova on top of Angular 6. Only one code needs to be produced and maintained and is fairly easy to code since it uses the same technologies as the web. The remaining challenges are similar to web page development but vary in difficulty.

Indeed, different phones come with different specifications in terms of computation power, users also have very different internet subscriptions that impact the network bandwidth the app can use. The app needs to be optimized for all of those parameters. However, the different screen sizes are the most important for a customer product. All designs need to be adapted to at least the smaller screen size, usually bigger size screens are easier to accommodate to. A future challenge if the app wants to expand is to make it responsive enough to support different languages.

To fetch results or send requests, the app communicates with the back-end. The entry point is the web server. It's a restful API that exposes open and secured endpoints. In the case of We3, this is deployed through Rails in the Ruby programming language. Unlike Angular, Rails does not require compilation which allows to do modifications without interrupting services that run in the server. The API then queries the database. It's an important security pattern to have the database isolated from the users. The database is a PostgreSQL server on Heroku as well. It's a relational database. In the event where the app is widely used and reaches millions of users, this would need to change. Indeed, relational databases rely on a strict schema that accelerates queries. However, it's hard to maintain and hardly scale to databases too large to sit on one server. It remains a good choice for a start-up start given the current number of users.

All those reasons made me choose to do my internship at We3. Indeed, the problem tackled by the app is very interesting and echoes with my experience of finding and keeping close friends even abroad. Moreover, the technical challenges the company tackles are at the core of my academic curriculum. Thanks to the small size of the company, I was able to touch almost every part of the app and be close to the business side. My internship focused on improving the user's experience. To do that, I was given two different roles. First I helped the development of the application by coding the app as well as the back-end. Then I took a more data-oriented approach.

II. My internship

Even though the app is around 2 years old, there is much to be done in order to make it a full profitable app. My first goal was to help the team develop the app as well as help reviewing it through testing and debugging

A. Full stack developer

To do that, one needs first to be familiar with the system. The first phase of my internship consisted in getting to know the team, the app and getting ready to code. I therefore installed everything that was needed to locally develop, test and contribute to the app. Once that was done, I could begin my first mission part of improving the user experience main goal: add visualisations when levels are completed and personalise the user's profile with a radar chart based on his answers.

1. Add visualisations

These two visualisations have different purposes.

As previously explained, the core value of the app is the promise to meet compatible people. The compatibility is based on questions users need to answer. This process of forcing users to answer a certain amount of questions is tedious. In the mobile application market, users have very little patience and tolerance in terms of frustration in applications. Therefore, if there is no visual feedback that indicates how many questions they answered and more importantly how many are left, the company's retention of users will be very low. That's the goal of the circular bar chart I had to implement. Its goal is to give an approximate idea of the progression the user achieved and improve the app's aesthetic.

The radar chat on the other hand has a very different purpose. The main goal is to make users' profile more attractive and personalised so that they can identify with their profile. It's also a way to give an idea of the algorithm behind the matching. When a user sees another profile, he can directly compare their persons by overlapping their radar chart. Without this chart, users have access to very few information on the user to decide whether they want to match with this person or not. Users can add their university, workplace and a small description of themselves but it needs action from the users which is not ideal. This chart comes from a more in-depth analysis of yourself. However, misinterpreted, it could have a negative impact. For example, if a user sees that he has a very low score on religion, it could offend him. That's the reason the indications about the precise categories is kept hidden.

In terms of development process, both charts were done in a similar way. Before I arrived, the team first detected the issues they should solve. Then discussed their precise goal. Finally, Julian made different designs. The first step for me was to get familiar with them. Then I followed this process:

1. Implement a proof of concept in a blank HTML, CSS, JavaScript web page
2. Integrate It into the app with dummy data
3. Modify the back-end to make the necessary data available for the app
4. Interconnect the front-end and back-end

Let us take a closer look at each visualisation.

a. Level complete chart

This chart was fairly simple to implement. For the first step, many blogs have examples of such charts like this one (Orlov, 2017). Once I understood how it worked and could be implemented, the real challenge when integrating it in the app was to make it responsive. Indeed, screen sizes vary greatly and therefore the dimensions of the graph which includes total size, bar width, bar space between them and so on, need to be computed somehow as a ratio of the screen. The graph is drawn in a scalable vector graphic “svg” element thanks to the d3 library for JavaScript. The final challenge was the labels of this graph. The coordinates of the text were easily computed but with a responsive font, its size is hard to anticipate. The solution here is to wait for the initialisation of the page and when it’s ready to retrieve the actual size each label to be able to encase the text into a nice colour patch to fit the design.

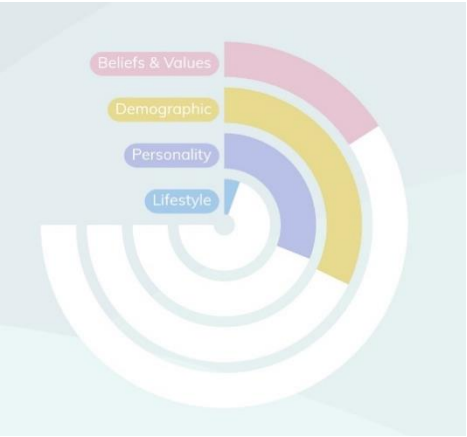

Figure 2. The final result of the level complete graph

In the backend, it requires a small amount of change. The data is fetched through an endpoint that queries the database to get the number of questions answered per category. The trick here is that there is not a fixed amount of questions per user. Indeed, some questions are asked to the user if and only if a previous question was answered in a certain way: yes, no, neutral or even picking certain elements from a question’s follow up can trigger dependencies as we called them which will increase the total number of questions the user will have to answer. To solve this issue, we hard coded the normalisation factors computed from the mean number of questions answered.

b. The radar chart

The radar chart was more of a challenge. It's a more original chart which makes the design more difficult. Thanks to my visualisation background I partly acquired thanks to the INF8808 course, I was able to criticize some aspects of them which led to some modifications of the final design. Figure 3 illustrates one of design Julian made. The problem here is the mix of continuous and discrete data.

Figure 3. An early radar chart design

(Munzer, 2015) tells us that one of the first question in data visualisation is to know what data to represent. Then you can think of how to best represent it. The "what" in this case is the mathematical representation of a user in the app. This comes from the analysis of the answers. Thing is that some data are continuous and some are discrete. The "how" here was the radar chart. This graph can be very efficient for continuous data not discrete.

The specificity of such graph made it impossible to follow a pre made tutorial so to code it, I learned directly from the documentation (MDN web docs, 2019) on how to draw paths in svgs. I was then able to compute the points and draw the patch but I also discovered that one can user Bezier curves to draw a smoother patch. Julian working with limited design tools wasn't able to design it but I tried and it was adopted in the final design.

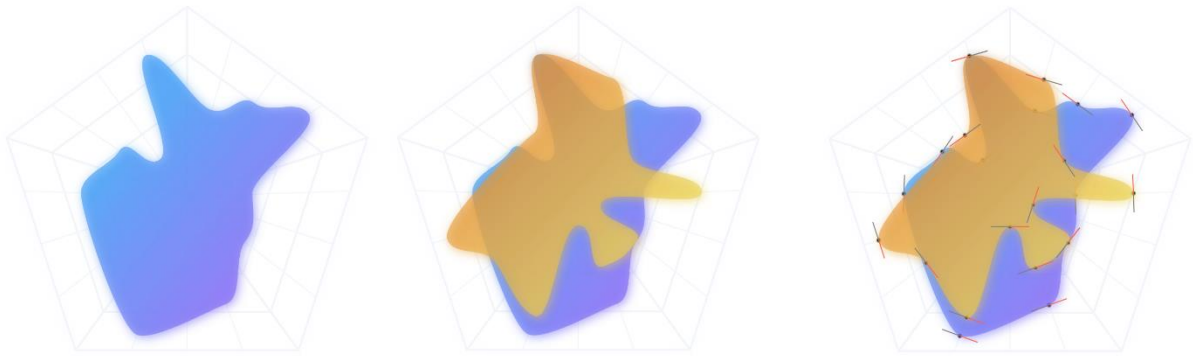

Figure 4. Smooth radar charts with tangents on the right

The most challenging if not interesting aspect of this chart is how and when to fetch the data. The data behind the graph comes from the answers and complex computations that are run asynchronously. Sometime after the users answered a batch of questions a job is scheduled to update or compute the data. To fetch the most recent data, they are two ways. The less optimised one is for the client to poll the server every 'n' seconds to fetch updated data. The best way is the server to notify the user whenever the data is updated. This mechanism is achieved with Pusher™ that uses web sockets between the client and the server. This two-way communication channel is very efficient and reliable.

Figure 5. Evolution of the radar chart- First prototype on the left – Final app version in the middle with the legend on the right

2. Profile Improvements

The profile was lacking content when I started my internship. The radar chart was a first step to fill it but it still needed personalisation and styling improvements. To do so, I was given the task to enable users to link their Instagram account and to style it up according to Julian's design.

a. Instagram integration

The integration of Instagram was tricky. Even though, one can access public profiles and therefore its media without any particular authentication, a third party such as our application has to perform some authentications to be able to use the Instagram api. Every third party requires a ClientID and a client secret that authenticates the third party. Unfortunately, this is not enough to fetch a particular user's media. To do so, the App needs an access token that is user specific. Moreover, to get this token one needs the access code. They are two ways of achieving this process. The safer and recommended one is the "Server-side Flow" (Instagram , 2019). The ClientID is shared between the front-end and the back-end. To increase security the client secret resides solely on the back-end. The code is available with only the ClientID but to get the access token one needs the client id and secret. This gives the flow described in Figure 6 below when the user decides to show its recent Instagram publications on its profile.

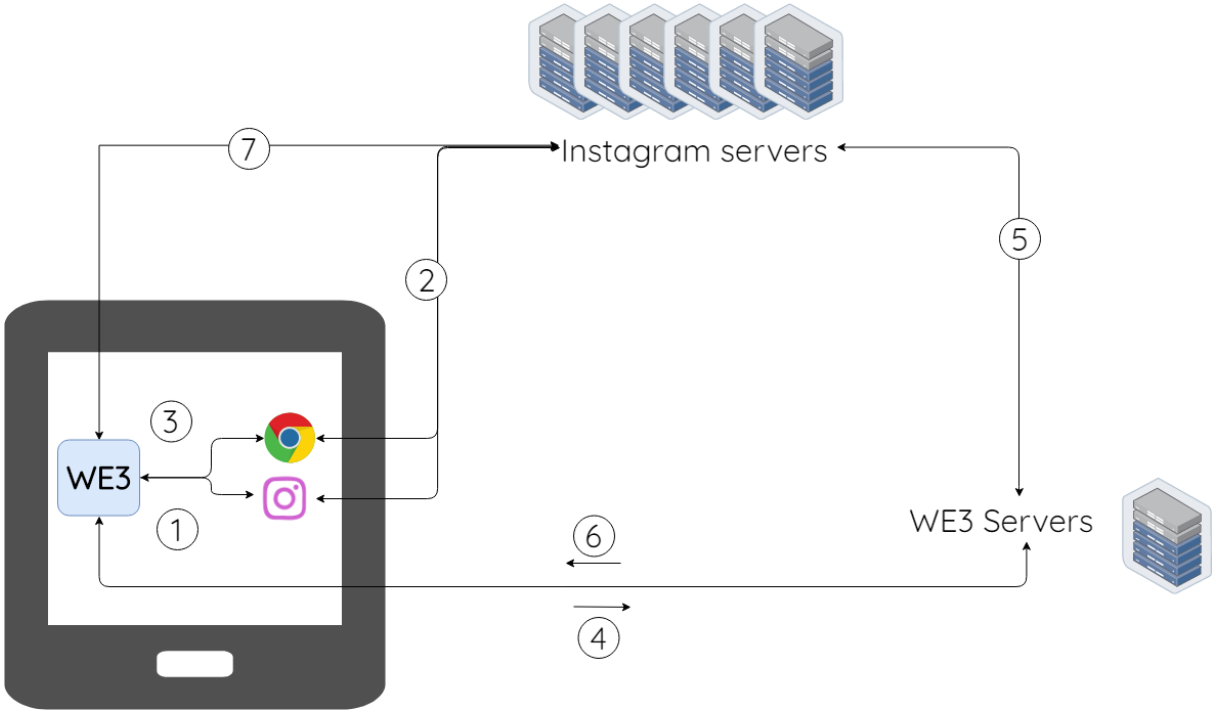

Figure 6. How to get Instagram media of a user process

1) A client clicks the button which opens a link with client ID that redirects to a browser or the Instagram App
2) The user signs in to Instagram and gives permission to share their media with We3. Then the request is transferred to Instagram with a redirect URI.
3) The user is redirected to the We3 App with a user specific code
4) The app requests the access token to the we3 api

5) The back-end fetches the access token thanks to the client ID, the client secret and the code the user previously fetched.
6) The access token is saved in the back-end and sent to the front end
7) The recent media from the user are fetched thanks to the user access token

Once this tedious process is complete, only step 6 to 7 are required to get the media of your profile or someone else's.

Once this was in place in the back and front-end, only showing the media to a user's profile was left. A subset is shown in the page. Clicking on one makes a popup appears in the form of a slideshow at the image clicked as the Figure 7 below illustrates.

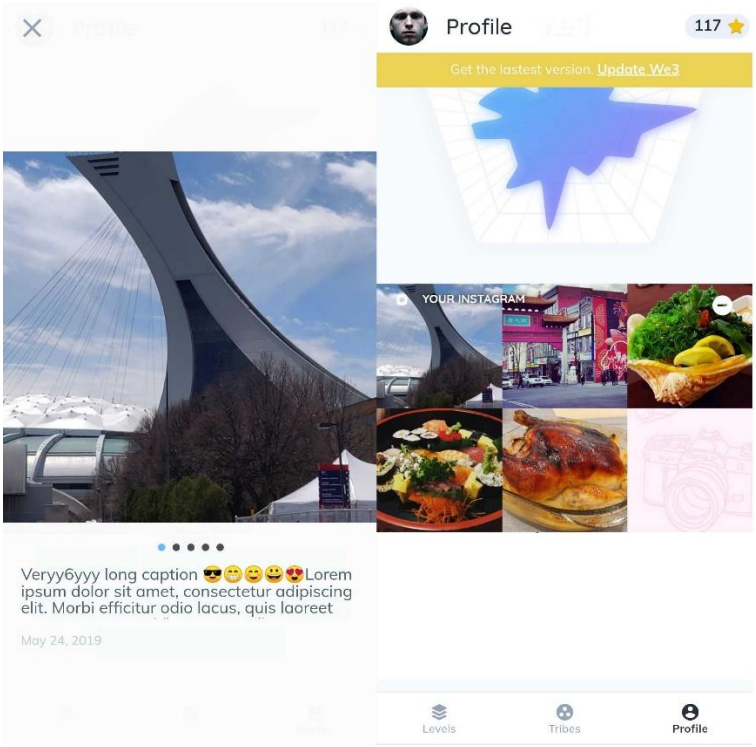

Figure 7. Instagram integration result in the app

b. UI improvements

Once the integration was complete, only front-end styling issues remained. Most of the work here was to reproduce as closely as possible the designs. However, the designs didn't cover every case. Indeed, the app framework and requests to the backend are timely which means that the page opened but appeared empty at first. The duration of this state depends mostly on the internet connection of the mobile phone and its computation power. As soon as the data is

loaded and computed the page filled up instantaneously. This rises two major issues in customer-based product. First, the user is left without any feedback on whether the data is actually being fetched. This builds up frustration that can, to a point, make users uninstall the app. Second, the sudden appearance of the data is unexpected and overwhelming. That's why I added while the data was loading indications that the app is not stuck but simply processing. And I added transitions to smoothly show newly fetched data.

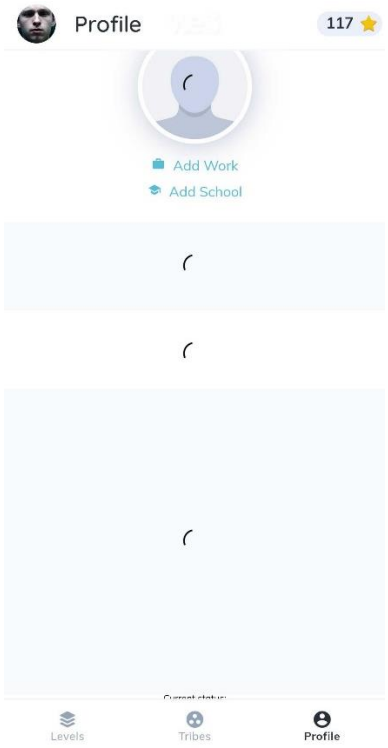

Figure 8. Loading feedbacks

In parallel to my work on those new feature and style improvements the team was actively working on migrating the app to a new version of Ionic. By the time I was done, the goal was to release the app as soon as possible. That happened to be harder than we thought.

3. Help push a new version - Quality Assurance

Before releasing the app to production which is open to the public in the different app stores, the standard process is:

1. Get every branch merged in the dev repo for both ends
2. Release the app in staging[1]

[1] Staging: the new version of the app available to download but only for developers.

3. Thorough testing
4. Document found bugs while testing
5. Fix bugs
6. Repeat

All these steps are mandatory to avoid introducing major issues in the app in production that could lead to bad user experiences which translates into bad reviews which hurt the business down the line. Besides, when developing a new feature, we only test this feature. Unlike big companies, start-ups that struggle to be profitable usually do not use unit testing. This may save some developing time but cost release delays. That's what happened in this version. It seemed that the migration to the new Ionic version broke a lot of pages throughout the app.

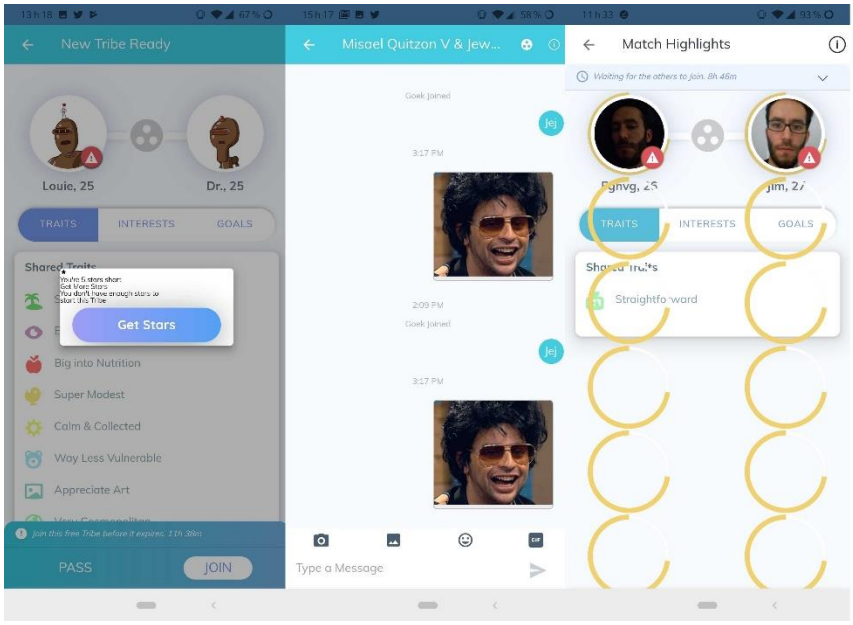

Figure 9. Bugs I found during testing

Since I wasn't familiar with the entire app yet, it was a good opportunity to discover it and discover bugs. Bugs usually appear in corner cases the developer forgot to take into account or appeared due to new features that lead to new states. As an inexperienced user, I had no preconceptions of how the app should be used which is an advantage to discover bugs. The second screenshot of Figure 9 above is a great illustration of this kind of bug. The process of fixing such a bug follow this pattern. You try to use the app as much as possible until you see a bug, that's step 1: discovering the bug. Step 2 is usually way harder: reproducing it. You may document the bug in the project management tool asana we use. But if they are no steps to reproduce, fixing it is much harder. In this case to reproduce the bug, you

had to go to a chat, put the app in the background and come back. Then, the entire chat would be duplicated.

As a developer in the team it was also a great opportunity to discover the code base behind the app. When I discovered a bug and managed to reproduce, I always tried to go and take a look at the code behind to see if I could fix it right away or not. In both cases, you learn a lot about the app and even discover bugs by simply trying to understand the behaviour of the code to a certain point.

The target date to launch the new version was around the time the Huffington article (Kirkey, 2019) release date. The publicity and the brand-new version combined would be a great free boost of popularity to the app. However, the version was not nearly finished. Julian decided not to push a broken version simply to synch with the article release date. The final version was out nearly a month after the article and but a small increase in traffic was observed around the release of the article.

The decision to postpone the release came partly from the fact that this release was not the most important one for the app compared to the next one.

4. Improve filtering and Enable further filtering for premium

Once the version was out and the production bug fixed, it was time to get working on the next version. This next version introduces brand new features: subscriptions plan. My first goal for this version was to unlock filtering for premium users.

What is filtering? Until this version, the filtering was automatic, a user could search for a tribe of users with the same gender that fell in their default age range. Which means that users could only be invited by same gender users. My task was to change that.

To do that, I needed to change the back-end structure and use of the filters. Only user's characteristics and parameters were needed to do automatic filtering: their gender and their age plus the size of the age range. For example, a 35 years old male user can only match with other male user whose age is between 28 and 42. Now, users could search for other genders in a different age range and users have the ability to change the acceptable age and gender of their tribe fellow in the case when they are invited. The challenge was to store all that information in an efficient way and more importantly a modular way. A possible design for future version of the app enables users to search for a tribe with a certain trait, for example, they could look for a tribe that play volleyball. This could widen the target market for the app.

The solution came from simple observations of the processes: a search is an action a user makes at a certain time whereas waiting for invitations is passive, a state of the user. Thus, we decided that invite preferences are to be directly stored in the user table but the search preferences are only stored in the search object that is created when a search is initiated.

Once the data structure is set, I had to change how the code accessed this data previously. I worked in parallel on the front-end and back-end to modify the endpoint for searches and how it's called in the app. The tricky part was to modify the query in order to get it taken into account. I'll go deeper in the search process in a section further down this report.

5. Make full use of questions and help resurrection with a bonus level

During the beginning of my internship, Julian created a brand-new set of questions to improve their matching algorithm. The previously created set of questions focused mostly on the user. The team felt that there was a dimension missing in a user's profile: what does the user want to find through this app? It can be critical because compatible persons could have different mindset when using the app. Some would want to find friends to go out with every night of the week even though they don't right now. Whereas others don't have or want to spend that much time with others. Problem arises from this. Old users that completed all levels will never answer those new questions so you lose the benefit of those questions. The solution is to create a bonus level. Actually, there was a problem before Julian introduced new questions.

Figure 10. Illustrations of the questions in the app

To have a personalised experience, users do not answer the same questions. To be precise, they are two different set of questions: one that do not depend on other questions and the others. All users are presented all the remaining questions in the first set at the end of the last level. A question of the other set of questions is presented only if the user answered a particular way to a specific question or follow up.

At level 8, the last level, the user is presented all the remaining questions which include the ones left in the first set and the ones he triggered previously in the second set. That's the problem. Keep in mind that all questions from the first or second set can trigger dependencies... You may ask yourself:

"What about the questions triggered by the answers to the Level 8 questions?"

You would be right. At the end there may be questions triggered but not answered. A first solution is to make a new level “9” to present them. Great. But what if those questions trigger others? And the others can trigger others? This is a recursive problem. To solve this Julian came with the idea of a dynamic bonus level. Instead of creating new levels, only one is presented and ends until no questions are left. The problem may be recursive, it is at most, limited by the number of questions. The main issue is to have a dynamic user interface that gives feedback of how many questions remain dynamically.

To achieve that, I reused as much as possible the original mechanism. In a nutshell, when a user starts a level, the back-end queries the database. As input the database needs the level number to get the maximum number of questions to present and the previous answers of the user to get the next questions or statements the user has to answer to. To make the bonus level fetch all questions, you can simply repeat this process when all current questions are answered instead of moving to the level complete page. When the database returns no questions that means, that all questions with triggers were answered and no other were triggered. Ok so there is a way to present all questions to a user. But how do we make it dynamic for the user?

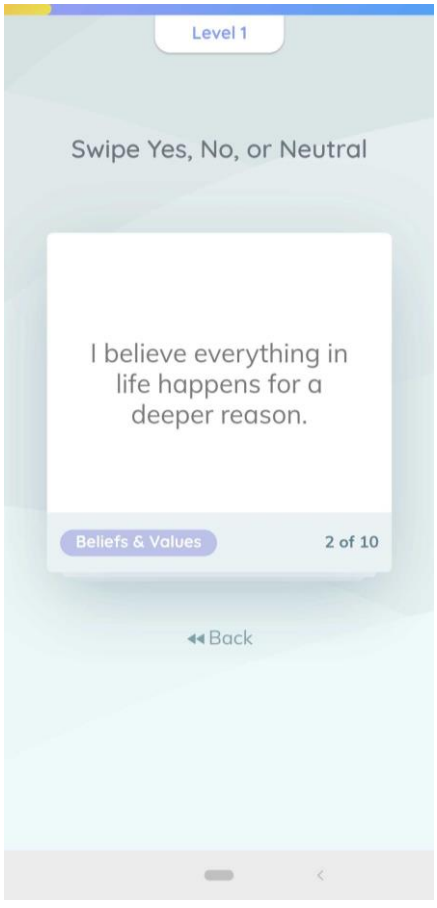

Figure 11. The statements’ page

In terms of user interface, two indicators of progress are shown in the page of the questions. On the bottom right, a counter gives the number of statements answered out of the total number of statements. On top, a progress bar shows the same information in a more graphical way. The goal here is to make them take into account questions to be fetched in the next iterations of the bonus level. Essentially, the dependency logic needs to be added to the front-end. The app needs to know that a question in the database has been triggered according to an answer a user just did. One way of knowing this is: once a batch of questions is fetched for the bonus level, the app also fetches the dependencies those questions could trigger. That way the app can keep track of how many questions were triggered to update the total number of questions the users will have to answer. At this point, all questions that do not trigger were answered in the previous level so the updated count is exact. The app can even deduce when a batch of questions is done if it should fetch or not a new batch of questions to save query time.

The last tricky part of this is the “back” functionality. In every level, this button enables users to go back to the previous question if they want to change their answer. In the bonus level case this means that when the back button is pushed, the potential dependencies triggered by the removed answers are also to be removed. This is ok but another case is more complex: what if a user presses back at the beginning of a new batch of questions? To deal with this we decided to make it impossible to go back at the beginning of a batch.

Through my involvement in developing the application I learned a lot; I improved my technical skills by discovering an unknown field that is mobile apps but also how a business is run for this customer facing market.

However, full stack developer is not the only role I had the chance to play during this internship. It became quite clear in the first conversations I had with Julian about the internship that I could help them gathering and extracting intelligence from their data that they cannot or more accurately don’t have the time to do otherwise.

B. Data analyst

My first task as a data analyst was to assess how fair was the app in terms of tribe invitations and suggest mechanisms to improve the situation. My second task was to make use of their freshly put in place analytics system into a Google Data studio report to give them more information on what’s going on in the app, what kind of experience do users have.

1. Improve Invites repartition - User experience

The algorithm’s first goal, is to find compatible people. Itself it does not and probably should not consider the popularity of users. So, Julian and the team had a strong hunch that their system was quite unfair. The problem I had to assess and improve was the distributions of tribe invites. Invites are quite critical to retain users. Even though they have enough stars to start a search when completing all the levels, they don’t have many more freely. If the first tribe fails because of some reasons, users have two choices: pay for stars to try again or wait for an invitation to a better tribe. Without a good experience, users will unlikely pay for stars and even less likely subscribe to plan to get weekly stars. Invites can therefore be a great way to retain users.

As it turned out, the situation was really as unfair as predicted and could be improved by other mechanisms without touching the algorithm. Let us first understand the system that leads to the invites.

a. Searches – Goal

The entire process starts when a user visits the search page of the app. In this page, there is a first validation that checks whether the user meets all the requirements. Those requirements include having finished level 5, having the location of the user, having a way to be notified through push or SMS, having a complete account (name, gender, age) among others. If the user meets all the requirements, the search button is enabled. Before clicking the button, the user can change default preferences for his search. The most important one is the locality. Users have the choice to look for a local tribe or a global one. This is very important for sparse areas when users don't necessarily want to meet the others but simply chat. As of recently, all users have the ability to change the age range they want the others to fit in but only premium users will be allowed to search for opposite gender users. When the user clicks the search button, all those preferences are sent to the back-end. While the back-end processes this request the user is redirected to a loading page until it receives an event that specifies the outcome of this search. In the best-case scenario, the search succeeds and the user that searched, called the **initiator** is taken to a page with the highlights of this new tribe with the other users. In this page, the initiator has the choice to start the tribe or to reject it. If and only if the initiator starts the tribe, he pays stars and the others are **invited**. Searches can also fail, in this case the app encourages users to change their preferences or wait.

The previous flow is not the only way to get invites sent. Once the tribe is started, the invitees receives a notification that takes them to the page associated to the tribe they are invited to. In the same fashion as the initiator, they choose whether to join or pass this tribe. The tribe does not fail, just yet, would they choose to pass. To prevent such fragility in the tribes, a swap mechanism takes place. This mechanism searches in the same fashion as the initial search for a third member that could replace the user that left. If a user is found, the other users that joined the tribe need to approve the replacement. If one does not approve the swap retries. If all of them (could be one member if only the initiator joined) approve, a new invite is sent to the freshly swapped user.

That's the whole mechanism from the point of view of users. That's the goal of the system. This process is quite complex but one challenge remains, how to choose out of all users the ones to present the initiator?

b. The filtering processes

To achieve this, the app filters users in several steps. To be clear, the input of this entire process is a user and some search preferences and the output is a tribe made of three compatible users: the initiator noted u1, and two others u2 and u3.

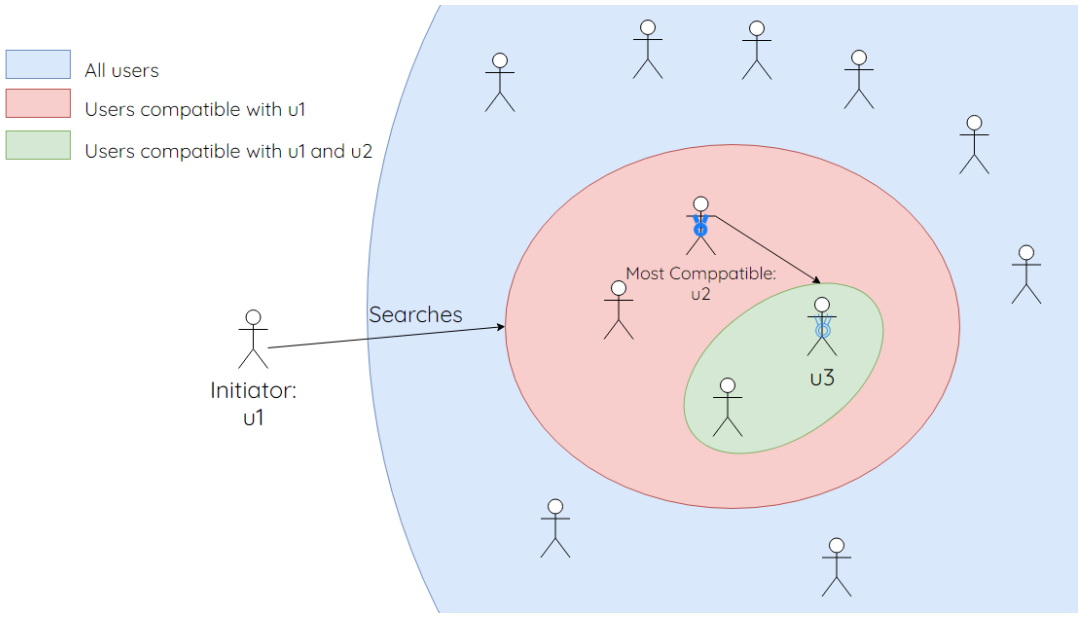

Figure 12. Filtering users for searches

The steps illustrated in Figure 12 are:

Step 1. Out of all users, keep the ones that meet the requirements to be in the matching pool[2], that fit the search preference and whose invite preferences fit with the initiator.

If a user u is closer to the initiator than the search limits he could be kept after Step 1. However, if u has a different gender than the initiator and accepts invites only from the same gender, he will not be in the output pool of users of step 1.

Step 2. Evaluate how compatible are users of this pool with the initiator and filter out those not compatible enough. Finally, order them from most compatible to least compatible.

This step is the business core. As previously explained, from the answers of a user, a profile is drawn and a similarity function is able to measure of how good a match are two users. The bar which determines how compatible two users are is another parameter to ensure a certain tribe quality.

Step 3. If the pool is empty or contains only one user, the search fails. Otherwise, take the most compatible out of this pool called u2 and filter the pool according to u2 preferences.

The steps 1 and 2 ensures that those remaining are compatible with u1. But since tribes are groups of three, the third also needs to respect u2 preferences. For

[2] Matching pool: All users that can be invited to a tribe

example, u2 and a candidate for u3 are close to u1 but u3 is too far away from u2 so this candidate is taken out of the pool filtered for u2.

It's worth noting here that invite preferences for gender are restricted to the gender of the user or any gender whereas the search preference target a single gender. This simplifies the filtering.

Step 4. Make a hybrid user from u1 and u2 and evaluate the compatibility with the users in the pool that is the result of Step 3. And filter out the non-compatible enough.

Step 5. If no user is left, go to Step 3. Otherwise take the most compatible labelled u3 to build the tribe with u1, u2 and u3.

All those steps lead to the creation of a tribe that is suggested to the initiator. Once again, invites will be sent to the others if the initiator accepts to start the tribe. If the initiator passes this tribe, the other two will temporarily be placed in their black list. If the user searches right after, the previously suggested users won't be picked.

As one can observe, the invitations are the output of a complex process that depends on many parameters. A more precise definition of the problem needs to be made

c. Analysis Scope

A good approach to this problem is to list what is not the subject of this study. First of all, user's location is not the focus here. In a very sparse area, if there are not enough users to choose from, the algorithm has no way to be improved. That's more a matter of marketing and distribution of the app. Another important factor is the search rate: how many users search? How frequently users search? If no one searches, no invites are sent. The invites system is limited by this outside factor. Finally, the preferences (age, gender) are also a limiting factor.

This study aims at studying: does users that answer a lot of questions and choose a lot of items in the follow up lists are attraction centre for the invites? Do lower level user have invites as well? Is the minimum compatibility score being too low or high? How to improve it? All of this without the other factors.

Given this, I was to try and assess the situation as well as improve it. A first thought could be to use historical data. By extracting data from the database as well as the firebase and big query analytics I could analyse how the system behaved. They are a couple of issues with this method.

First, I would need to duplicate the data on my personal computer. This introduces a security breach. Even though the data could be anonymised, the security of the data would now also lie on the security of my computer. Moreover,

this process would be quite long and tedious to duplicate data as well as merging the different data source. And finally, to have accurate results I would have to process the data to try and remove the influence of factors I listed above.

To cope with those problems, I came up with this solution. First, historical data would strictly be used to assess the situation by fetching only the invites data, not the answers of users or any personal data. Then I would run simulations to test the system and improve it.

d. Simulations – Generation

Running a simulation as an alternative to historical data has many advantages but needs to validated. Does it model users well enough for the results to be generalised to real users?

Before running simulations, one must first generate users. I simulated them as pseudo code below describes:

Algorithm 1: Generate users

Input: The number of users to generate and the user to clone: *n* and the source user to initiate the simulated users, *u*

```
1  Function generate_users(n,u):
2  for i=1 to n do
3    ui = User.create_from_clone(u)
4    ui.level = random(3,8)
5    for level=1 to ui.level do
6      for question in ui.next_questions(level = level) do
7        a = ui.answer_random(question)
8        if a.has_follow_up then
9          a.answer_follow_up(random(0,a.follow_up.size)
10       end
11       save_answer(ui,a)
12   end
```

Users are therefore all the same in terms of profile. The only real differences are the levels and answers. This way, Step 1 and 21Step 3 filtering are no longer a concern. Only the compatibility remains.

An important aspect is to present bots questions in the same fashion as users. The next set of questions should depend on the previous answers. This reduces the performance of the algorithm but is critical to model users.

The random levels and the random numbers of items chosen in the follow ups of questions were drawn through a uniform distribution. It means that in the simulations there is approximately the same number of users per level which is sensible. For answer's follow up however, this does not model user's behaviour very well. Users in a least effort fashion, tend to choose very few items. Therefore, the correlation between the number of items chosen and the number of invitations, if there is one, should be overestimated.

Once all users are generated, simulations can begin. The basic simulation consists in making all users search one at a time. For simplicity, we consider in simulation that all initiators start their tribe. In this conditions u2 and u3 will be invited.

To evaluate how fair the system is, we used the Gini coefficient (Wikipedia, 2019). A Gini coefficient of 0 is the ideal situation. To have that, all users would need to have the same numbers of invitations: pure equality. If you order users by the number of invitations and then take the normalised cumulative curve, you get a straight line in an ideal situation. To get the Gini coefficient graphically, one needs to do the same transformations and compute the air between the ideal curve and the actual curve times 2.

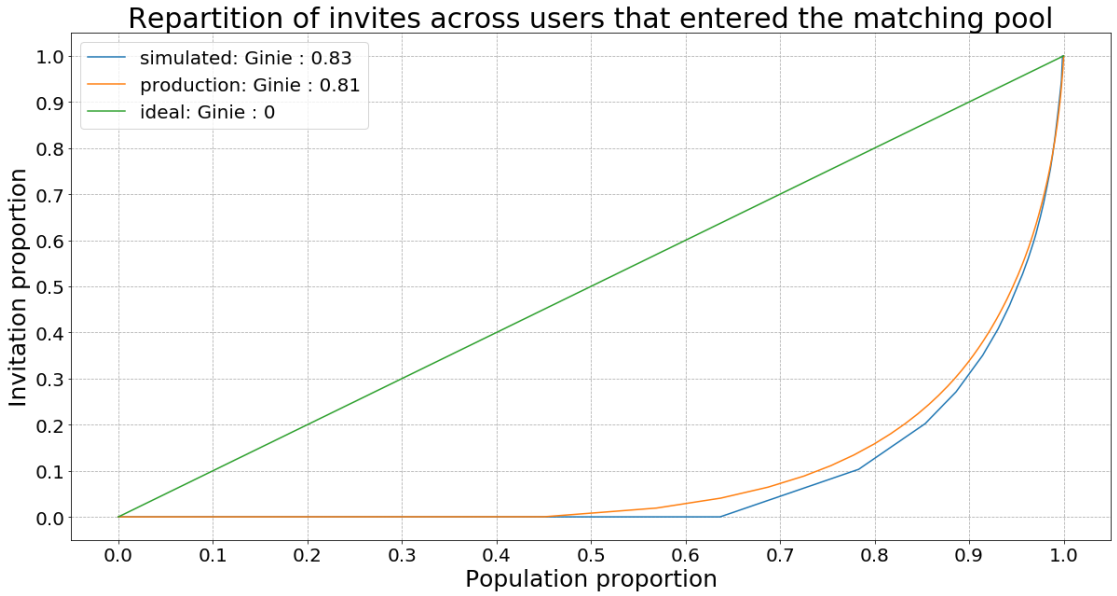

Figure 13. Comparison of real and simulated systems

Figure 13 shows the normalised cumulative curve for the ideal situation, the situation in production and the simulated situation. Those curves are called Lorenz curves (Wikipedia, 2019). The simulation consisted in generating 1000 clone users that searched. As we expected, the situation is quite unfair. This chart tells us for example that fifty percent of all invitations are sent to only five percent of users. There are two main problems that arise from this chart:

1. Among those who receive an invite, the invites are not evenly distributed
2. Almost 50% of users never received an invite.

As previously explained, those problems could be the result of external factors such as location or even due to data collection but when compared to the simulation that removes the influence of those factors, the problem remains. This means that even in an ideal environment where everybody could be matched only a few would receive invites and some users would receive plenty of invites.

e. Investigation

A great advantage of using simulated data is that we have total access to it. To avoid security reasons, no actual personal (information or answers) were gathered which would prevent us from investigating the situation, trying to understand it. Simulated data enable us to do that. Since the simulation has been validated to mimic the situation, we can infer from the study of the simulation the same conclusions to the production environment with caution nonetheless.

A first hint to explain the inequality was to say that users that picked many items have a greater chance to receive invites. Figure 14 and Figure 15 attempt to validate such hypothesis.

```
nb_answers     0.104476
items_count    0.339790
level          0.197231
matches        1.000000
Name: matches, dtype: float64
```

Figure 14.Correlation between number of invites and answer related parameters

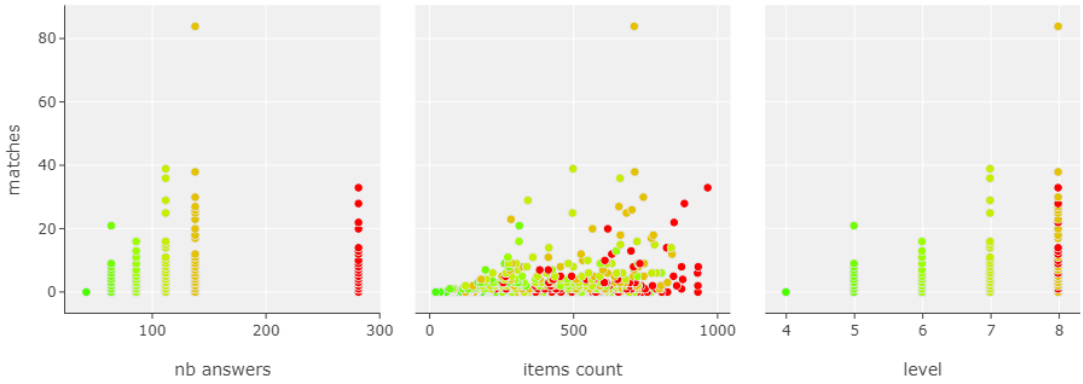

Figure 15. Plot number of invites versus answer related parameters

The dot's colour represents the level of users with a small difference that red users finished level 8 whereas orange just finished level 7 but considered level 8. Both figures show that the strongest correlation with the number of invites a user gets

within those parameters is the number of items chosen. However, it's lower than expected so it does not entirely explain the invite distribution.

Another interrogation one can rise is what's the actual perfect situation? Indeed, the situation where everyone gets the same number of invites is not realistic, the question is: Does every user can actually get invited? The fact that compatibility gets into play when choosing to invite when a search is made can limit the number of users that could receive an invite. To investigate that, I recorded "potential matches" which represent users being kept after Step 4 which means they were compatible with 1 and u2 but were not the best choice (including u2 and u3).

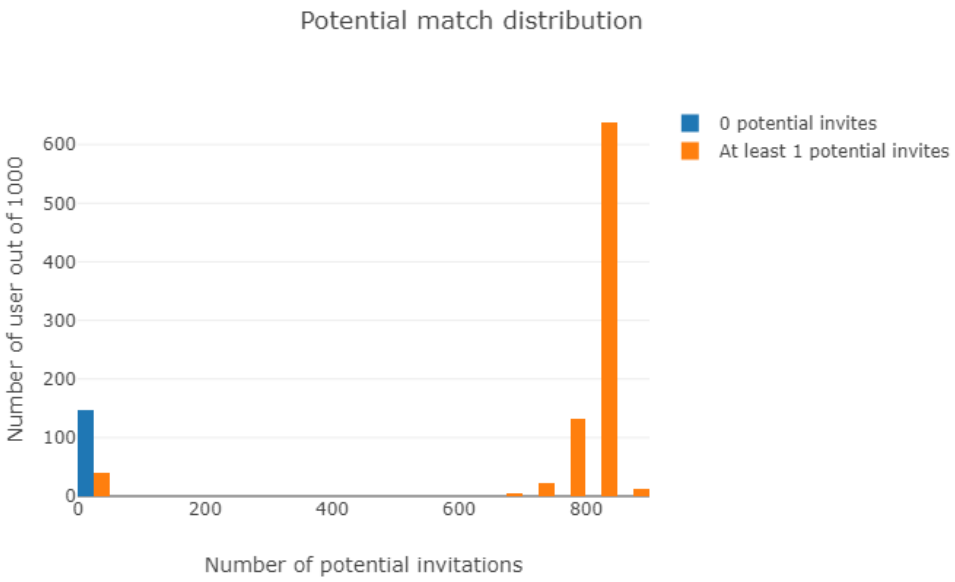

Figure 16. Potential matches distribution

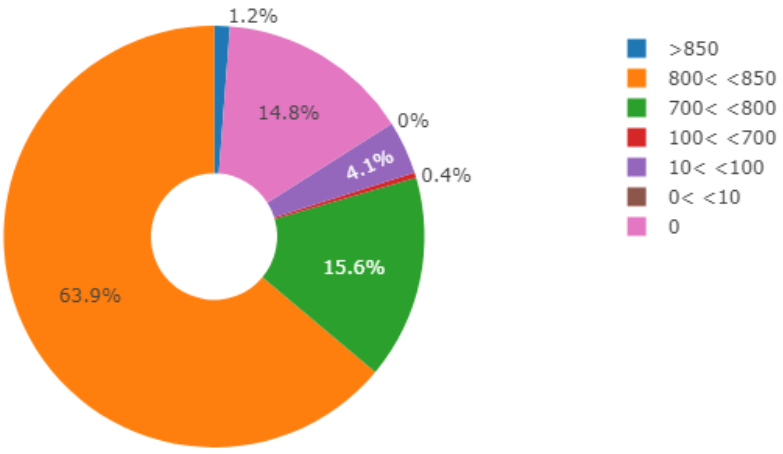

Figure 17. Potential invitations repartition

Figure 16 and Figure 17 shows that there are two main groups: those who get very few potential matches and the ones that get many potential matches. Those numbers were generated with a 1000 user population that searched. The maximum number of potential matches is equal to the number of searches. We therefore, have almost 15% of users that couldn't even have an invite due to factors outside of this study. The best situation we could achieve is given by the Figure 18 with a Gini coefficient of 0.2 instead of 0.

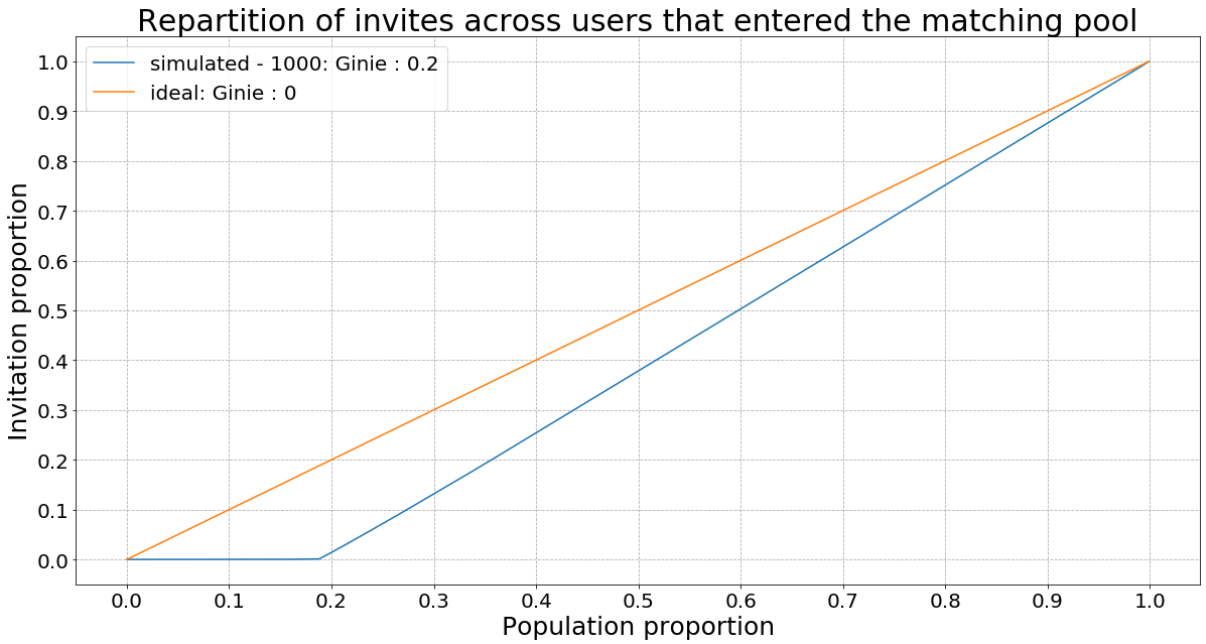

Figure 18. Perfect invite distribution

Now that the I shed some light on the situation and more importantly to what extent it can improve, I tried different mechanisms and evaluated each one of them.

f. Improvements mechanisms

The problem is not the measure of compatibility. Instead of modifying the algorithm that computes compatibility, the goal is to prevent users to attract all invites when other users are compatible as well. A first way to improve the distribution is to introduce a penalty.

The idea is that if a user searches, finds a u2 but there are couple candidates for u3 that are compatible. If the most compatible has just been invited, he shouldn't not be chosen over the next that never got an invite for example. To do that, I introduced a penalty. This penalty is applied after users were filtered out due to compatibility. It simply inserts 2 similar steps Step 2.5 and Step 3.5. Those steps consist in:

Step [2 or 3].5 For every user left, compute a penalty and sort the pool by *readjusted compatibility = compatibility – penalty* in a descending manner.

The core problem is how to compute this penalty. To keep computations simple, I chose a linear model for penalty. When a user is invited, his associated penalty is set to a certain maximum penalty. With time, his penalty decreases until reaching 0. This requires two parameters: the maximum penalty and the time interval whilst the penalty is still not null.

The goal of this study is to find the best parameters for the system. However, the time interval is very dependent on the activity of an area and other factors outside this study so I focused on the maximum penalty and set the time interval at 1. The total duration of the simulation gives the interval between searches.

This mechanism is great but aims at solving mainly the first problem, which is the inequality distribution among those that received invites. To solve the second problem which is to help users get their first invite, I studied the effect of a bonus mechanism. In the simulations this mechanism was pretty simple, a user would get a constant bonus until he gets invited. The added steps remain the same but with a negative penalty when a user has a bonus which boosts his adjusted compatibility. To get as much as information on the effect of those mechanisms, I tested them separately as well as together.

g. Results

As previously explained, the Gini coefficient is a great metric of invitations distribution, Figure 19 shows the evolution of this coefficient with the different mechanisms in place.

Figure 19. The impact of penalty a bonus on the Gini coefficient

This graph shows three curves: with penalty-only in blue, with bonus-only in orange and with bonus and constant penalty in green. One can observe that the penalty improves this coefficient but not as much as the bonus mechanism. The improvement is almost insignificant beyond a penalty greater than 11.

This coefficient is a great indicator but lack interpretation power in terms of the distribution of invites across users. The Lorenz curves given by Figure 20 give a better insight of the effect of both mechanisms.

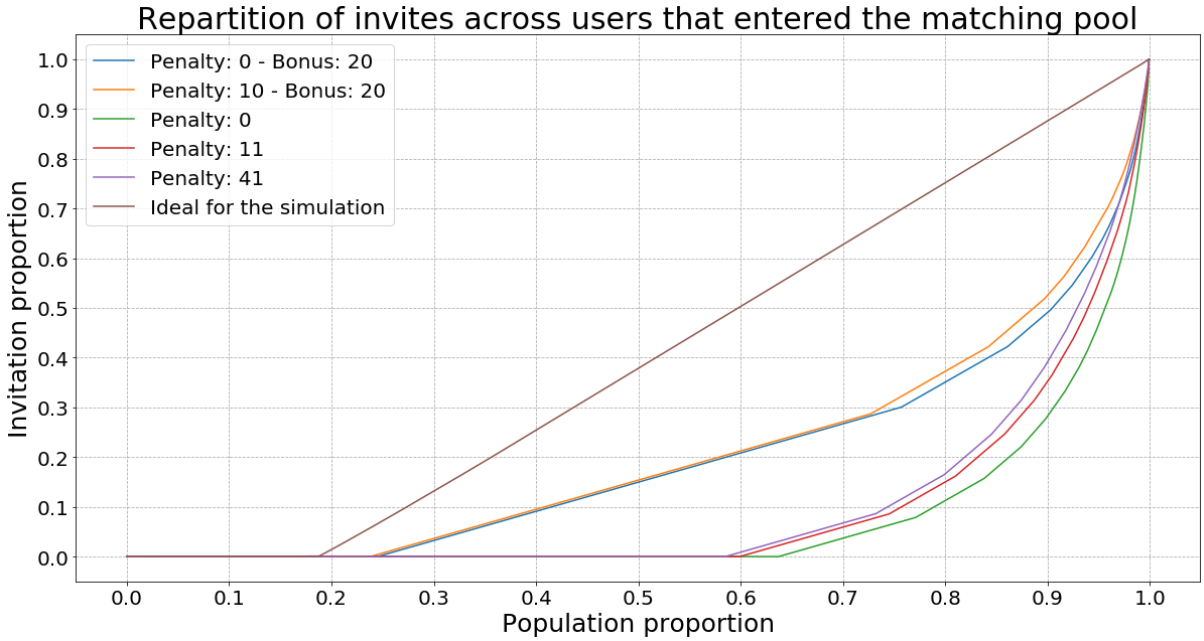

Figure 20. Lorenz curves for different simulations

For visualisations sakes, not all configurations were represented here in Figure 20. The previous results are clear here and can be explained. The three curves with penalty-only in this graph show that the proportion of the population with no invites decreases but in the range of a couple percent. The real impact of penalty is to flatten the curve. Just as I hinted, the penalty improves the repartition of invites across those who already got invites in the first place.

The bonus mechanism on the other hand, has a significant impact on reducing the number of users with 0 invites. The proportion of such users drops by more than 61 percent compared to the initial situation. It's also interesting to see the impact when a penalty is added as well as the bonus. The number of users with no invite drops by 4% compared to bonus only but more significantly, it clearly straightens the curve. This means that some invites given to very popular users are redistributed among the others.

Those results are great indicators of the impact of parameters but are far from actually representing them. This problem is a graph by nature, users are connected when they are matched. I therefore used a different visualisation to

show the impact inspired by (Weser, 2017) code associated with the article that explains graph with forces in d3; Figure 21 table shows the results.

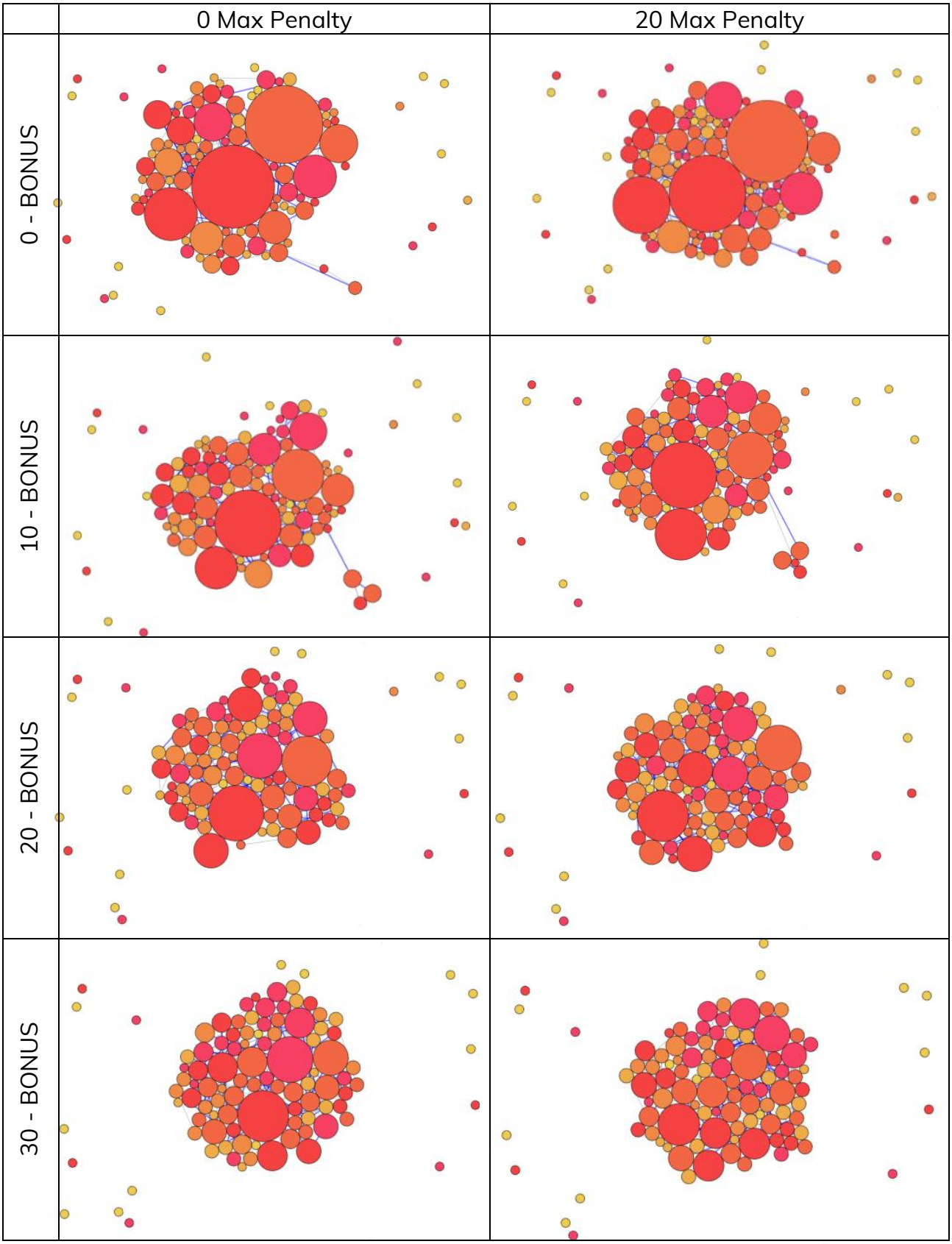

Figure 21. Nodes visualisation of the impact of penalty and bonus on a population of 50 users

For visualisation and computing power limitations reasons, the visualisation comes from a different population of 50 users. Each user is represented with a disc. Its colour represents the level of the user from orange for level 4 to dark red for level 9 (completed last level). Their size encodes the proportion of invites they received. They are linked when belonging to the same tribe. As we could deduce from previous analysis, without any penalty or bonus, the invites are mostly distributed to a small fraction of users. They are a couple of clearly distinguishable large discs that attract all others. Moreover, they are many unattached discs, meaning a lot of uninvited users. As bonus increases, this number of unattached users decreases. Penalty mechanism tends to even out the size of the discs. The colour and size combined are a great indicator of that. The initial situation is mostly big red disc whereas with 30 penalty and bonus, it's more even size discs of different colours.

This visualisation is not meant for precise and generalisable deductions. Indeed, it's computer intensive which limits the size of the population and the visual nature limits it to subjective interpretation. However, it's a great side tool to better understand the Lorenz curve and the Gini coefficient.

We may have an idea as to which parameter is the best to improve the situation but an issue remains. Before choosing the parameters, we have to make sure the negative impact the mechanisms will have is not damaging.

Indeed, those mechanisms will tend to choose less compatible users than the initial flow, but are the users chosen not a good match? To evaluate the negative impact, I studied two related parameters that indicates tribe quality: tribe score and the highlights count. The tribe score is a formula computed from the compatibility of users and the tribe highlights count. The latter is interesting to isolate because it directly impacts the users. When users a presented a tribe once they searched or because they are invited, they are presented the tribe highlights. The decision of starting or joining the tribe will be based on this. The overall quality measured by the score gives hint whether the tribe will start and have a successful output. Figure 22 and Figure 23 evaluates the impact on both parameters. Figure 22 shows the average tribe score for every set of parameters. As anticipated, the more penalty or bonus is applied, the lower the mean tribe score gets. It is interesting to see that the effect on the Gini coefficient of penalty is no longer significant after 10 but the tribe score still decreases. The bonus has also a strong impact lowering the score by almost 10% with a bonus of 20. Adding the penalty to the bonus has almost no effect on this metric. We can expect a similar behaviour for the tribe highlights count.

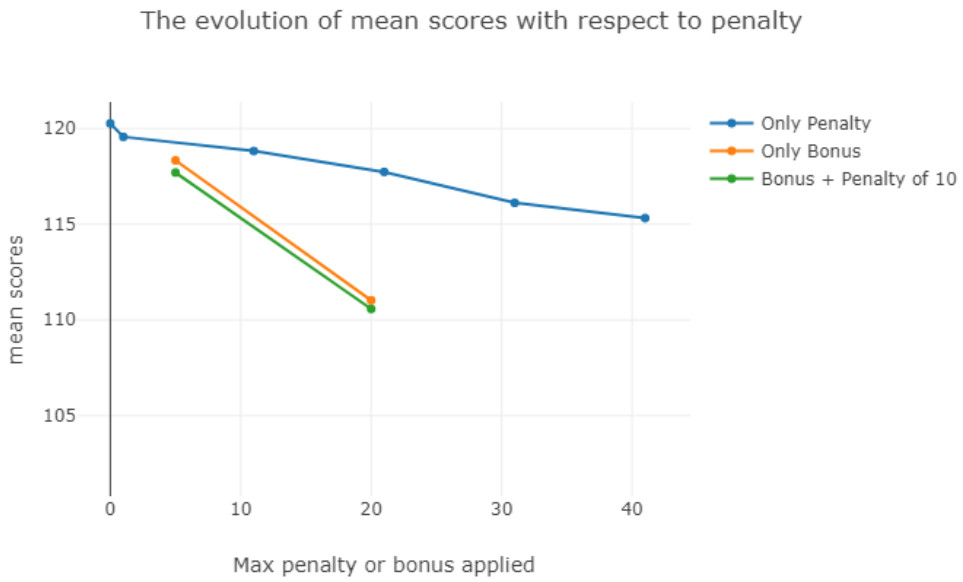

Figure 22. The impact of penalty and bonus on tribe quality

Figure 23 takes a different approach. It shows for some configurations the evolution of the tribe highlights counts according the tribe index. The first point of the curves represents the average of tribe highlight count for the first tribe of users. Since bonus is applied only to yet to be invited users, we can assume its impact will mostly affect first tribes. That's what one can observe. A first remark is that according to the previous Lorenz curves we know that few users have more 1 tribe and even fewer have more than 2 and so on.

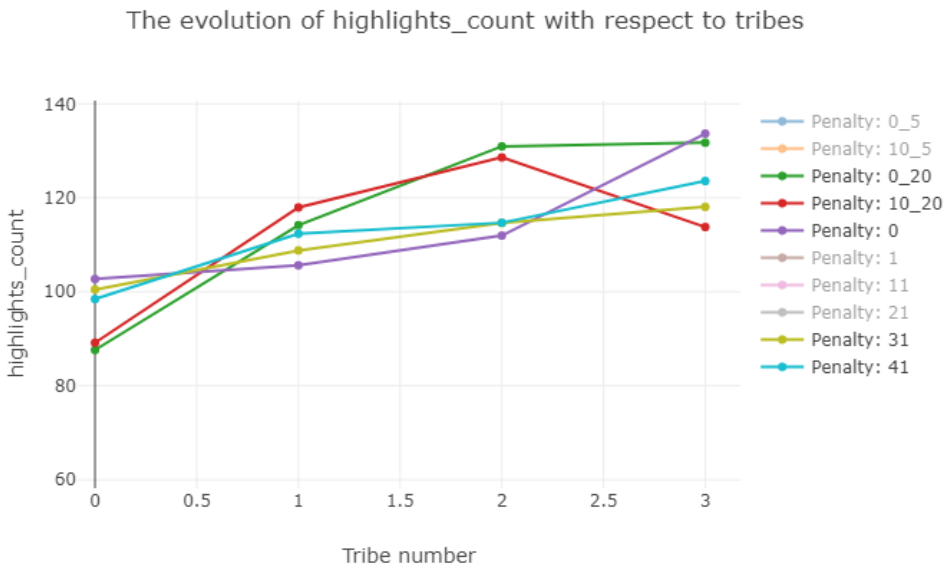

Figure 23. The impact of penalty and bonus on tribe highlights count

Figure 23 shows that indeed, the number of tribe highlights count is affected mostly by bonus. Other configurations with penalty only follow the same trend that the baseline: no penalty nor bonus. For first tribes, the number of highlights is reduced by around 10% with bonus.

This study shows that first a bonus mechanism is great for user to get their first invites but bonus only would mean that, given the pool is closed, once all possible users got an invite, the situation goes back to the initial unfair situation. A penalty mechanism also needs to be implemented. Penalty focuses on evenly distributing invites among the most compatible users. Both have negative impacts on the tribe quality. Bonus affects mostly first tribes but it seems that its impact is greater than the penalty.

h. Decisions and Implementation

Those conclusions are very important but to apply them, some business sense is needed. First there is still the time interval when the penalty is applied to determine. Moreover, this simulation is an ideal scenario with a dense area not frequently achieved by the app distribution across the world. To get more business sense in this, we scheduled a presentation for me to show the results to Julian and the team and to agree on an implementation of this mechanism.

After an almost day long presentation and debate we decided to set the time interval to 3 months and set the bonus equal to the penalty to 20. More importantly we decided to change the formula to compute the penalty and bonus. Instead of penalty dropping by zero, it keeps dropping until reaching max bonus. For implementation reasons, we kept the bonus constant until a user is invited. The problem here is to get the entry time to the pool if we want a bonus of 0 at first slowly increasing. The reason for this is that tribes don't only start by invites. When users go the onboarding process, they very easily start a search and still easily start a second one for free. This study focused on invites only. Every number did not, rightfully so, take the initiator into account when counting invite.

In a traditional application, those mechanisms would be implemented and active feedback from users would give the information to measure the actual improvements in production. To have more accurate and frequent data, an analytic system is usually put in place in the app.

2. Build a live dashboard in Google's Data Studio

Indeed, a traditional way to test change in an application in terms of design, mechanisms is to pay testers to give feedback. This process can be long, inaccurate and costly. Instead, a more efficient way of measuring the response of users in the app is to record their actions.

a. The analytics system

Analytics is exactly that. Every time a user does an action the team considers worth to record; an event is recorded. To do that; our application sends the event to a third party with all the information needed. To be more precise, in our case, there are two analytics system: the front-end and the back-end. Every event about navigation in the app or action made by a user is recorded by the front-end analytics. Whereas every event that requires multiple users to do a certain action is recorded from the back-end. That is a first issue.

Analytics are made to be viewed and analysed to detect problems in the application, bottlenecks that limit user experience or even revenue. Both ends due to their different nature send the events in a different third party. The application sends events to Firebase whereas the back-end to BigQuery. Both have limitations.

As a first attempt to globally analyse the events and unify both data sources, Julian created a Google Sheets that list all the events to be tracked. Without any connections to Firebase or BigQuery, one needs to do the query by hand and report the result in the sheet and redo them to update the results. This a quite a tedious process. To improve this process, I was assigned to do a report on Data Studio to make those queries and visualise them. But first, what exactly are we tracking with those analytics?

b. Funnels and audiences

A simple metric to have is the number of a certain event that fired. For example, how many users visited the page to subscribe to the premium version of the app? To really analyse this, one can also want to know how this metric evolved with versions or two months ago compared to last month. Those simple filters can be tedious to query, that's one feature data studio gives, through simple fields, all queries can be filtered by fields.

This event-based database can also be analysed through funnels. A funnel is a series of sequential actions a user can go through in the app. For example, the events:" started_level_1, complete_level_1, started_level_2, completed_level_2, started_level_3, completed_level_3" is a funnel. One can measure how many users went through each step and identify problems or target the efforts to for example make level 2 shorter. However, this is not such a good way of measuring that. By processing steps independently, some noise can be added. Indeed, you always considered those events in a certain time frame. If this time frame covers the entirety of the availability of those events, there is no problem. Otherwise, in this example, a user could have started and completed level 1 before the beginning of the time frame and finished level 2 inside the time frame. Therefore, if you compute the fraction of users that completed level 1 that also completed level 2, the numbers

of people considered for the level 2 completion is overestimated. This is called an open funnel. To get a closed funnel, you need to solely consider users that entered the top of funnel. Out of the users that started level 1, how many completed level 2? This gives the closed funnel value for the event completed level 2 under the assumption that events are all steps are mandatory before completing level 2. A closed funnel will have better accuracy but is more complex to achieve.

Another important concept is audiences. Audiences are built on top of funnels. In a nutshell, audiences are users that followed a certain funnel. This is particularly useful for small apps like We3. One type of audience could be the users that enjoyed a "perfect experience" described by a funnel like completed all levels and got matched with users within five kilometres two times. Those particular users are very important because they represent the goal of the app. Out of those filtered users, you could analyse how much do they spend on stars (if any). This would theoretically give you the final conversion of the app and you have the final revenue per user after the app is fully distributed. This stand on strong assumptions. First is the ideal experience really achieved with the current application? Those users are but a small number of people, generalising their behaviour can lead to inaccurate predictions.

Those metrics can be very powerful to do forecasts as well as focusing the company's efforts in the best direction according to users' behaviours. Extracting audiences and funnels can be quite a challenge. One of the biggest challenges is combining and unifying data sources from Firebase and BigQuery.

c. Firebase and BigQuery

First of all, Firebase and BigQuery are non-relational databases. BigQuery sits on top of BigTable, a column oriented distributed database. In practise, it means they are no foreign key pointing at another table. It's a simple list of all the events containing all the parameters. To efficiently store this, each row can have multiple values for the same column as illustrated in Figure 24.

Row	user_id	event_timestamp	event_params.key	event_params.value.string_value	event_params.value.int_value	event_params.value.float_value	event_params.value.double_value	event_name
1	53014	1569073900602078	search_initiator	null	53016	null	null	search_pool
			search_id	null	2221	null	null	
			score	null	null	190.334643054719	null	
			penalty	null	null	19.98351389664732	null	

Figure 24. BigQuery row example

Parameters in a json fashion associated with the event are stored by adding a key and value to the event parameters associated with the event and the values are stored in a column grouped by type (string, int, double, etc).

Noticeably, Firebase goes even further with it. Indeed, in Firebase, a user has some kind of state stored in user properties. One can change those through the api of Firebase and when a new event is fired, the latest user properties will be attached to the event.

Those storing solutions are extremely efficient to store in terms of scalability and event query time but are complex to query and combine. Google data studio aims at facilitating this.

d. Google Data Studio

Google Data Studio is a free visualisation tool similar to Tableau™. There are many connectors to import data from many different sources including BigQuery and Firebase. Predefined graphs are available to quickly visualise trends, numbers and even do computations. This tool has interesting features as well as limitations.

Data Studio is indeed quite flexible in importing data. A great feature is to be able to import data from a BigQuery query. With this data, one can create charts and every chart have specificity but allow filters and data sources merge. With a simple data structure, one is fully able to do complex query and visualise the results through the user interface without any SQL prior knowledge. Chart filtering is also very powerful. Filter control elements are associated with a field of the data, they will filter. On the top right of Figure 25, lies a filter control that acts on dates. It can be set to control all the charts in the page. This way one can very simply change the time range to be studied. This filter can be applied on may fields such as the app version but it will act on the charts that are associated with data that contain this field. For example, back-end data do not have an app version field, so filtering it will not change those charts.

Figure 25 is a portion of a page of the resulting report. It illustrates the formation funnel. The funnel is the tribe formation process that consists in three steps, the tribe starts, its chat is unlocked and finally it is formed. It roughly corresponds to a first user joining the tribe, a second one joining and finally, a third one joining. On the left one can see the open funnel figures and on the right are the values for the closed version of this funnel. To get them, I blended data sources in Data studio terms, which corresponds to a SQL left join. The tribe ids are extracted from the event parameters and become the join key. One can choose to keep any fields or metrics from each blended data source. To get the numbers of chat unlocked in the closed funnel, you blend the tribe started and chat unlocked data sources and keep the event names of both. Each row corresponds to a tribe as illustrated in Figure 26. Since it's a left join, all tribes have that event. The number of tribes whose chat unlocked is the number of tribes which column associated with the chat unlocked is not null. Three examples are shown in Figure 26.

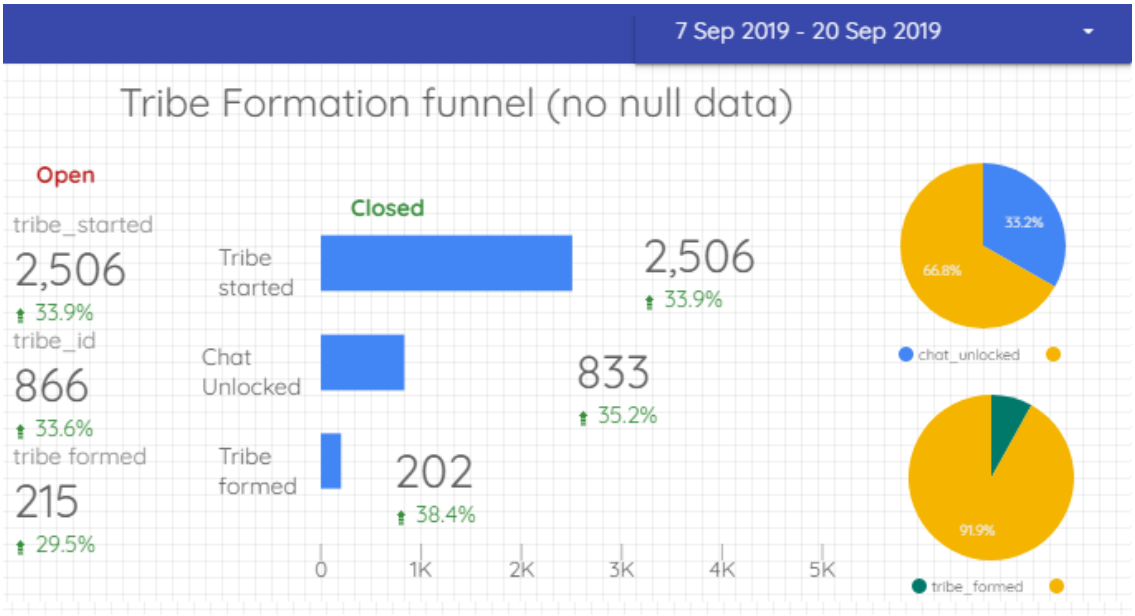

Figure 25. One funnel in data studio

	tribe_id	tribe_started	chat_unlocked
1.	232143	tribe_started	chat_unlocked
2.	232004	tribe_started	null
3.	232103	tribe_started	null

1 - 100 / 2506

Figure 26. Table view in data studio

Once that is done, a good metric is to get the actual ratio of the chat unlocked tribes over all the tribes that started. I was not able to make the score card that displays a number and its different computations to give it. One way to do so is the pie chart you can see on the right of Figure 25. With this chart, one can display the proportion of the different values of a column. Here the column has two values: “chat_unlocked” and null which one can see in the legend. The label of the chart gives you the exact ratio.

This example is a great illustrator of some struggle I came across with Google Data Studio. There are many others like the fact you can not join multiple data sources with different keys or that charts are quite limited in terms of personalisation. In conclusion, this tool can be a great one for quick and simple visualisations and quite accessible for non-programmers but lack flexibility with complex data.

This concludes the extent of my work throughout this internship. Thanks to the flexibility of this start up I was able to go from full stack developer working directly on the app to data analyst in order to improve their app core business systems and finally help their business intelligence by making their analytics live and clear. To resume and finish this report, I'll compare my experience with the initial objectives.

III. My experience

Those objectives were evaluated throughout the report but indirectly. The initial objectives were separated into two categories, technical and personal objectives.

A. Technical Objectives

1. Strengthen their knowledge and skills through the application of concepts, methods and tools learned during their graduate training and applicable to a problem of interest to the industry;
2. Acquire practical knowledge relevant to their discipline by using the approaches, methods and systems of the host organization;
3. Develop their critical abilities regarding approaches, methods and systems used in the workplace;

The first objective was reached. Indeed, in web developing technologies as well as data science I was able to apply my knowledge and skills as well as improving them. An example is coding the application visualisation in d3. I learned d3 and visualisation techniques in the INF8809 course and was able to apply and learn from their application in in-production product.

I partly reached the second objective by working hand in hand with more experienced developers. It could have been even more exceeded if I could have worked with a full-time data scientist. For this part of my internship, I was in charge of solving the problem thanks to my skills only.

The third objective was broadly reached. I learned a lot about start-up methods. Its strength lies mostly in flexibility in the workflow but its weakness lies in part in the less robust final product that can lead to time heavy overheads.

B. Personal Objectives

1. Develop or improve their teamwork skills by collaborating with other engineers in back-end design and development.
2. Demonstrate their ability to write a summary targeting the problem, the methodological approaches and the results obtained during the internship.

The first objective was reached. Working with a more diverse group than in your own department forced me to improve my communication skills by adapting them to the target audience. I learned a lot from senior developer as well as from the design part of the app.

The second objective was also reached mostly when I worked as a data analyst. To make my work be fully incorporated in production, I needed to present my findings to the team. It goes hand in hand with the first objective. I first needed to sum up my work and results and then to communicate them as well as I could to make it worth it.

Conclusion

My internship was really formative. As a first real industry experience, I had the chance to work on very diverse topic that fit nicely with my formation. I applied my knowledge and skill to real life problems and data in order to help deliver a great product. I also learned a lot in fields I do not follow in school that is business. Running a start-up requires a lot of investments and well-balanced risks. The team is fully committed to the product even though it has yet to be proven a profitable and sustainable one. This requires a lot of business measures to follow its evolution to determine whether or not this product is worth so much effort. Apart from that, the team was great to work with. I was fully integrated to it right in the beginning. That's why I decided to keep on working part time on the application. The application is on the brink of moving to a new phase. The subscription model, once implemented will give more insights on the potential of the app. Once that is demonstrated, the app will go through funding and start to grow. Scaling a system is also quite a challenge. If the app scale worldwide, new challenges will rise. I would like to help overcome those challenges if this experience is as rewarding as the internship was.

References

Instagram . (2019). *Authentication*. Retrieved from instagram.com/developer: https://www.instagram.com/developer/authentication/

Kirkey, S. (2019). Researchers are working on a pill for loneliness, as studies suggest the condition is worse than obesity. *National Post*.

Maija Kappler, K. H. (2019). Navigating Loneliness Is Hard, Even Though We're More Connected Than Ever . *Huffington Post*.

MDN web docs. (2019, August 18). *Paths*. Retrieved from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial/Paths

Munzer, T. (2015). *Visualization Analysis and Design*. CRC Press.

Orlov, A. (2017, October 16). *Radial Bar Chart built with D3*. Retrieved from bl.ocks.org: https://bl.ocks.org/AntonOrlov/6b42d8676943cc933f48a43a7c7e5b6c

Weser, R. (2017). Interactive & Dynamic Force-Directed Graphs with D3. *Medium*.

Wikipedia. (2019, September 12). *Gini Coefficient*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Gini_coefficient

Wikipedia. (2019, July 30). *Lorenz Curve*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Lorenz_curve