Final Report

Group 3: Guazzaloca, Guidi, Liso, Lorenzoni, Marzolo

Document Structure

Final Report 1
Document Structure 1
Introduction 2
Glossary 3
Initial Product State 3
Product Scope 3
Generated Backlog 5
Initial Process State 5
Suggested and Mandatory Tools and Practices 6
Development Team Structure 9
Scrumble Team-building Exercise 9
Sprint documents and organization 10
First Sprint 10
Sprint Backlog 10
First Sprint Final Activities Report test 11
SonarQube Usage Report 13
Sprint Burndown 13
Additional Information 14
Second Sprint 14
Sprint Backlog 14
Use cases diagram 15
Second Sprint Final Activities Report 15
SonarQube Usage Report 16
Sprint Burndown 17
Additional Information 18
Third Sprint 19
Sprint Backlog 19
Third Sprint Final Activities Report 20
SonarQube Usage Report 22
Sprint Burndown 23
Additional Information 23
Fourth Sprint 24
Sprint Backlog 24

Use cases diagram 25
Fourth Sprint Final Activities Report 28
SonarQube Usage Report 30
Sprint Burndown 31
Additional Information 32
Conclusion 32
Collected Tweets 32
Current Product State 33
Possible Future Improvements 33
Issues and Suggestions 33

Introduction

This document is the final report on our experience creating a twitter dashboard and analyzer as the project for Software Engineering course in University of Bologna (year 2020-21).

The scope of the project was twofold: the development of the software itself, and the documentation of the process we stuck with, which was supposed to be, for the most part, Scrum. At the same time, there were some limitations to our interpretation of Scrum; in the first part, this was also due to our inexperience. We also made some compromises to increase our productivity: as an example, we avoided daily scrum meetings, since we could not work every day, let alone for a full day.

Following what was just outlined, in this document we will analyze the choices behind both our process and our product. In order to better represent the life cycle of the Agile framework, the professors Paolo Ciancarini and Marcello Missiroli let us know in advance that our requirements were likely to change; for clarity, we have decided to include all additions, in chronological order, in the same section. We were also left with a lot of creative freedom, so our implementation choices are our own, even the choice of using a web app instead of a mobile or desktop app. Instead, we were more tightly constrained with the choice of the tools we would use, as we will outline in the "Initial Process State" section.

Although we had creative freedom, I'd like to make it clear we were still supposed to check in regularly with the Product Owners to ensure customer satisfaction with minimal risk. This was one of the sources of confusion for us: we were held accountable by two separate entities (the "Professor" and the "Product Owner"), represented by the same people. In the rest of this document, these two figures will be more or less split: we will use the words "Professor" and "Product Owner" related to, respectively, product and process followed. Instead, during development, feedback was given on both aspects at the same time, and the two sides clearly affect one another. This created unnecessary complications.

Lastly, I want to mention how we picked the language of this work: because we're from an Italian university, but development is deeply rooted in English, we've tried to keep English as the "official" language. We made this choice because of three main reasons: we find it easier to voice our implementation thoughts in English, since a large portion of our vocabulary is forced to be in English; English increases our reach, and since we plan to make our work public it would have been short-sighted to prefer Italian; this gives us a chance to practice! At the same time, most of the material we were given is in Italian, and we have decided it is not worth it to translate it for the sake of posterity, so although English will be the official language, parts of this document will be in Italian.

Since we don't know the proficiency in either English or Software Engineering terms, we decided to include a short glossary for reference to the reader. Definitions in this are NOT complete, but should be sufficient for a basic understanding of this document. We hope this will clear up any misunderstanding and avoid needing to read any redundant outside information.

Glossary

Agile Development Practice: A set of software development practices focused on discovering requirements and developing solutions though self-organizing teams and a direct relationship with the customer.
Scrum: An agile methodology, iterative and circular.
Product: The result of the work by the development team.
Process: A set of steps to be executed during a given period of time. In this document, also used to refer to all activities and necessities that are outside the Product itself.
Product Owner: Professional figure responsible for maximizing the value of the product. In this document, one of the roles covered by the Instructors.
Professor: In this document, the role covered by the Instructors that relates to teaching the course.
Instructor: The physical people who oversaw the course. Paolo Ciancarini and Marcello Missiroli.
Development Team: A group of 3/4 developer and a Scrum Master.
Scrum Master: A person in the development team that coordinates the developers and their relationship with the POs.
Practice Patience: A serious game to identify and prioritize process improvements.
Sprint Review: An informal meeting between PO, SM and Development Team. Due to time constraints, never done with the PO present.
Sprint Retrospective: A meeting were the people involved in the project suggest option to improve the process.
Essence: A standard for the creation, use and improvement of software engineering practices and methods.
React: A JavaScript component framework for client side application.
NodeJS: A runtime for JavaScript.
Express: A JavaScript server side framework for Node.js.

Initial Product State

The project was presented by the Instructors very early, in October, as the continuation of a thesis. Because we were learning about agile development at the same time, most of the focus was put in explaining the process rather than the product. The thesis had resulted in a desktop application written in Java + JavaFX with basic twitter analysis, from which we were supposed to take inspiration. Because of our previous experience with JavaScript, we decided we would work on a Web App written in React.

Initially, we thought our best bet was to work with the relatively recent NextJS, but during our second sprint we realized that that avenue, although it simplified our initial setup, was quickly getting out of hand. So, we decided to switch to NodeJS, and our final product is a NodeJS + Express WebApp written in React (we'll talk about this in more detail in the following sections).

Porting code from the previous iteration of Twitter Tracker quickly revealed to be, at the very least, impractical, but we took inspiration from it to guide our initial work. We also looked at a few existing solutions to understand the current landscape, and after a meeting with the POs we realized we should focus on the map capabilities.

Product Scope

Luckily, the Product Owners had reached the same conclusion. Our requirements were introduced with an initial document (linked in the next section) which outlined the main use cases in an informal fashion. At the same time, we started understanding the main features of agile development and scrum, but, as we would realize in development, mastery and true understanding is deeply rooted in practice, so we can't really say our initial interpretation of scrum practices was accurate. Moreover, almost all requirements related to the process given by the Professors were considered just a hindrance but tolerated as "restrictions". Although we knew they were an integral part of the course and development in general, we did not find any advantage within them.

Requirement Document

This is the initial document we were given. It was not formally expressed as User Stories (which would have been a requirement in scrum) because the Professors wanted us to understand three important lessons: Product Owners may not be fully compliant all the time, as many times they are not used to receiving specific instructions to the development

team and would consider the work required to follow those instructions outside of the realm of their duties; extracting User Stories from long-winded documents that only summarize uses is not easy and requires a specific vision for the final product; User Stories do not only serve as tight constraints, but guide development towards a specific vision. Finally, I want to include a personal observation: User Stories, because of the point of view they take, support describing user experience features very well, but are less suited to explaining technical or implementative choices; moreover, they are completely useless when what is required is describing a specific developer need (e.g., All developers must request a separate API key, in order to guarantee redundance in following stages of development).

Requirement Analysis

Below the analysis of the initial specifications of the project; the file containing them is included as an attachment with the rest of the final report documents. We only included parts that we believed were relevant in creating new User Stories. The following section was written during the First Sprint (when we received the requirements)

1. *"Mettendo uno o più hashtag relativi al terremoto posso mostrare i risultati geolocalizzati su una mappa"*

   Notes:

   - Could be done using twitter stream API through geolocation filters
   - Do we need to include some default hashtags? (e.g., "earthquake", "derby", "protests")
   - Client-side handling? How can we forward tweets to the client?

2. *"Posso includere un posto o un hashtag e poi raccogliere foto da quel posto"*

   Notes:

   - Provide different ways to specify a certain place (e.g., drawing a rectangle on the map, entering a hashtag containing the name of a place, etc...)
   - How could we handle tweet images?

3. *"Inserendo uno o più utenti posso raccogliere i loro tweet e geolocalizzarli, anche inserendo le foto contenute nei tweet sulla mappa"*

   Notes:

   - Implement different behavior for images. Review this in the future

4. *"L'applicazione dovrà permettere di visualizzare, consultare i tweet con certi hashtag e - sotto certe condizioni- attivare una procedura di allarme"*

   Notes:

   - Unclear, needs more information but we suggest a variant: the user could start a stream and set a certain threshold of tweets to collect, when the threshold is surpassed the system informs the user through some sort of notification.
5. *"Si richiede di aggregare i tweet in una dashboard interattiva, possibilmente mostrando viste coordinate che presentino diversi dettagli dei dati (e.g. le posizioni su una mappa, una word cloud dei termini usati, un diagramma a barre con il numero di tweet raccolti in un certo periodo, ecc...)"*

   Notes:

   - Map and word cloud are feasible, bar chart takes more time; we could consider it in the third sprint.

Evolution of our architecture

We started building the app using a React based framework called *NextJS*. We decided to use it because it allows us to create a single codebase containing both client and server code, thus speeding up the development process and better reasoning about the code. Later, though, we stumbled against some *NextJS* oddities and limitations, such as the fact that *NextJS* kept rebuilding modules for no apparent reason thus removing all data from memory, included the ones related to streams. This made us move to NodeJS + Express.

Generated Backlog

Here we will include the Backlog that we generated after our first "pass" though the requirements. We would like to mention that we knew this backlog was incomplete, but it made little sense to convert it to User Stories before learning what we were working with. Due to our inexperience, we have since reworked this list many, MANY times, so this should only be a general idea of what we were going to implement in the first phases.

id	subject	description	Time est	Acceptance Criteria
1	Development server setup	Have a server that can be used to run self-hosted versions of services.	4	service in cas and our application needs to be accessible from the external
2	Cas setup	Verify CAS is working as intended, include it in self-launch	2	all service available in the cas enviroment are up and running (optional bugzilla and logger)
3	Taiga running	we'll need a running version of Taiga to start working.. But start working to get a running version of Taiga.	2-10	Able to use Taiga SOMEWHERE, even on website if we're sure we can export
4	API study	Let's have at least 1 expert in Twitter API capabilities	5	The expert has an opinion on what can be achieved, and can guide development as to which endpoints to use
5	User Last Tweets	As a user, I want to be able to input a username and receive a list of the last few tweets he sent. Was initially included in sprint 1, later removed in grooming and made optional	8	a user can input a twitter username in the main web page of our application and his last tweets are showed
6	Starting Geolocated Filtered Stream	As a user, I can input a coordinate box and start collecting tweets coming from that box.	15	After inputting coordinates, as a user I can receive the tweets that were collected. OK both in bulk and single tweets.

We have collectively decided it would not make sense to discuss any of these User Stories any further, since they were quickly reworked and reworded (or deleted entirely), but we still believed it was important to understand where we started from. In addition, we initially used different formats for User Stories, but we made these old ones fit in our new standard for the sake of recording them.

Initial Process State

Unfortunately, while the Product State was presented clearly, we cannot say the same for the process. We considered glossing over our difficulties and focusing on what we achieved, but we think that would give an unfair view of our process evolution (and hinder the instructor's ability to make the course better for next year), so we will include our negative opinions about the initial "setup" in the final section of this document.

Suggested and Mandatory Tools and Practices

The instructors taught us the basics of Agile Development and Scrum, but they were confident in both our ability to pick it up as we went and the importance of practice. Because of the nature of the course, we did not approach this in the fastest, most effective and least comprehensive way, but instead considered various alternatives and recent advances: a clear example of this is the use of Essence Cards. We will outline this and other tools and practices in this section.

Proposed tools

We would like to preface this section by mentioning that the Instructors decided we would be using open-source tools, a decision we supported whole-heartedly. This means that all the tools that were suggested (except for the logger, discussed later) are open source and preferably available in a self-hosted instance.

Essence, Product State and Practice Patience

Essence, as mentioned on the creator's website, "is a standard for the creation, use and improvement of software engineering practices and methods, which is maintained and published by the OMG international open standards consortium." In our case, it was useful for three main reasons.

The first one is that it allowed us to avoid lengthy documentation: condensing information to the size of a card is a great way to keep a reader interested and give a bird's eye view that aids understanding, without getting distracted by specific details and missing the complete picture or, on the opposite side, skipping key parts of the process.

The cards themselves would prove to be very useful as well, since we used them in two "serious games" which helped us approach the Review and Retrospective activities. The first one would probably barely classify as a game, as it simply consists in going through the seven "Alphas" - the key elements and areas of interest common to all software projects – and for each of them identify which state the current product resides in. The seven Alphas are Requirements, Software System, Team, Work, Way of Working, Opportunity and Stakeholders, and their states are complemented by checklists, informal ways to move in-between states. This provides the readers with two great advantages: a clear idea of where the project sits (and which Alphas still need to be worked on), and a clear path ahead, due to the checklists. We completed this activity at the end of all sprints, since we found it extremely useful for the two uses outlined here.

Instead, in the Retrospective, we adopted a serious game called "Practice Patience". A detailed description can be found here, while in this we'll only outline the main advantages we identified: keeping conversations on track, giving actionable feedback, and not forgetting about any unspoken doubts. While I believe the first two are described in the article, I have learnt not to underrate the third one, especially in the initial sprint reviews.

Lastly, I would like to mention an "honorable" usage of Essence cards that I initially considered but later scrapped: using Essence to formalize the process of getting feedback and acting on it by the Product Owners (and possibly even the Professors); due to the universality the Essence kernel aims at, it is possible (and suggested) to include additional practices by using Essence to formalize practices from different development frameworks. I believe this would be an enlightening guided activity in next year's iteration, as it truly shows the power of Essence as a (somewhat) unopinionated, agnostic "standard" (the word the authors use) for formalizing and streamlining practices from different frameworks. At the same time, it would be unreasonable to expect student to have reached a sufficiently deep understanding of the standard to complete it on their own.

Taiga

Taiga, as its website says, is "an open-source project management software that supports teams that work Agile across both Scrum and Kanban frameworks". Clearly, this software is one of the two foundational tools we used, together with GitLab, to manage and organize the software development process. Its capabilities are vast; so vast, in fact, that we found some of them useless for a project of our size, and we ignored them. They range from the basic Kanban board, to a sprint task board for each sprint, the availability of a point breakdown of each user story and task, an issue tracking system, a comment system, different roles with varying responsibilities and a lot more. For us, Taiga was the most sensible alternative to the partly closed-source, paywalled Open Project, since Jira was out of the question as fully open-source and frankly enterprise-oriented. Our use of Taiga will be documented in the next section.

GitLab

GitLab needs no introduction: as the most popular open-source alternative to GitHub, it was suggested by the Instructors and we quickly adopted it as our (only) version control software.

BugZilla

BugZilla was initially proposed as a complementary service to Taiga for issue tracking, thanks to its endless integration possibilities and historical relevancy. We decided not to use it, as our organizational overhead was already far too large to include yet one more tool we were not likely to use.

Mattermost

Mattermost was the solution of choice by the instructors for day-to-day communication and light issue tracking (which would then be moved to either BugZilla or GitLab, integrated with Taiga). We can't really say anything bad about it: it was light, simple to install a self-hosted instance of, had a well-working mobile app and an appealing interface. Unfortunately, all of us had been using Telegram for the longest time (by now considered open source), so even though we completed the setup we jointly agreed we would favor Telegram.

SonarQube

SonarQube was new to all of us: it's a code analysis tool that runs static analysis on your code and brings vulnerabilities or possible future problems to your attention. Then, it generates a report on your current position, and proposes changes to better your code. We used SonarQube starting in the second sprint, and it helped us find some otherwise invisible vulnerabilities (for example with our use of CORS), but we realize it is probably of more help to larger codebases.

Logger

The Instructors were also interested in our productivity data, for future research purposes. This was not really made clear in the beginning, nor was it properly set up. This resulted in some misunderstandings on our part and raised privacy concerns that, luckily, we were able to bring up with the Instructors. We had two main problems with the logger: its scarce availability constrained us to a single IDE we were not comfortable with and we were not given a self-hosted version to install on our own server. We will now tackle those issues and describe the steps we took to resolve them.

The plugin availability was a direct consequence of the nature of the process: this Logger (possibly called Innometrics, although the project name seems to have changed during its course) was the result of a university project, built by students in both Innopolis University and Bologna University. Due to this, its adoption is limited to say the least, and the only plugins that were made available to us were for Eclipse and IntelliJ. Most of our development team uses VS Code, or alternatively vim for remote development, so we were not thrilled to be requested to move to (and learn) a completely new (and sizable) IDE with IntelliJ WebStorm. Eclipse was out of the question as some of us find it so outdated, especially when working with JavaScript, to be unusable.

For what pertains the self-hosted version, the issue was twofold: the code of the backend was only partly available on a GitHub repository that later disappeared (to be installed with a docker container), and we never managed to get any kind of frontend. This meant that: we could not install a self-hosted version of the logger and would have to use the university-provided instance; we had no way to know what kind of data the logger would send to the server; we had no obvious way to visualize it and would have to dive into its source code to understand how its database was set up and how to query it.

These factors, combined, led to our decision to investigate for alternative ways to track our time - possibly less intrusive than a literal key-logger - in order to provide the instructors with comparable data we were comfortable with sharing. We'll see in the next few pages what we came up with.

All self-hosted tools were hosted on a Google Cloud machine. We picked this because of both ease of use and how generous the "first use" credit is: thanks to the initial credit and how long it lasts, we were able to use it for free. All management and system administration tasks were completed by the development team.

Final Tool Configuration

Throughout the first two sprints, our tool selection varied. What we will now describe is our final usage, what we used since the end of the second sprint. We find that this selection allowed us to respect the Instructor's whishes without adding so much configuration work to go beyond a full sprint's number of hours.

GitLab

We already discussed GitLab in the previous section. GitLab was the cornerstone of our tool architecture and contained the complete source code and branches. Initially, we mainly worked together on one branch, but later we moved to a feature-branch approach, with new features being developed in experimental branches and merged in master once they were completed and tested. Tests were man operated.

Taiga

As we mentioned in the last section, Taiga was our management software of choice. We installed a self-hosted version of it in the same cloud as the other services, but initially used the web version while the setup was being completed. Our management of Taiga is somewhat discussed in the first Sprint Reports, but we will outline the main characteristics here.

In order to include both User Stories and Development Needs, we used Taiga cards as "Backlog Items". Each Item would then go through a pipeline of stages from New to Archived. The stages were New, Wait Approval (by the POs), Wait Verify (confirm the request is doable with Twitter API), Ready, In Progress, Ready for Test, Done, Archived and Rejected. If either approval or verification failed, or if we deemed them too minor to include them at all, they were Rejected. User Stories could be moved to Done once they respected the Definition of Done; they were then Archived after the end of the Sprint. User Stories not explicitly required by the PO were marked as "optional". To distinguish between User Stories and Dev Needs, we used tags.

We used the Wiki to archive Sprint Documents, the Definition of Done, the Team and Scrum Master Diary (I only completed the Scrum Master Diary until the end of the third sprint; afterwards, I included that info in the final Sprint Report), and a Useful Links section. We included the Scrum Master role, but the point attribution quickly got out of hand, so we only used it partially. Each User Story had an estimated time attribute.

User Stories were divided into tasks, both at the start and during a Sprint. Each task has a field for recording how long it took to complete it, and the Tasks follow a similar pipeline to User Stories (but don't need to be approved or verified). Not all tasks are assigned, because we often worked in pairs or groups and Taiga does not allow multiple assignees for Tasks. At the start we used Epics to divide Items into sections, but we soon realized that it was not helping and stopped.

SonarQube

We used SonarQube since the second sprint. We did not include it into an automatic pipeline but ran one-off scans. In each Sprint Report a section is dedicated to SonarQube metrics and performance. SonarQube was installed as a self-hosted service in the same Google Cloud account as the previous two.

Telegram

Because of our familiarity with Telegram, as soon as we picked our group members, we made a Telegram group and started chatting there. This made switching to a different service complicated; even more so considering that no matter how clean Mattermost looks, it's still in its infancy regarding some features. Telegram is pretty much open-source, and we won't go into details on MTProto here. We used basic messaging features, occasional polls, file upload and pinned messages the most.

WakaTime

WakaTime was our time-tracking software of choice. Although all its plug-ins are open source, the server code and front end are not. We understand this was a compromise on the Instructors' position, but it was driven by urgency and ease of use. In a more organized setting or a future installment, we suggest switching it with either Kimai, fully open source but requiring more customization for our chosen use, or looking into other alternatives, such as Super Productivity or

GitLab time tracker. We only used Wakatime to track IDE usage, but it has a Word plugin and a chrome extension available.

Etherpad

This is a service we used for short-lived text, shared and collaborative documents. We decided not to host it on Google Cloud and use riseup instead.

Discord

This is one of the two completely non-open-source tools we used. We used Discord because of our familiarity with it and the vastness of its features. We tried replacing it a few times with open-source tools such as Jitsi Meet, but its reliability made us use it more often than not. Still, I would not consider it unavoidable: there are many tools which provide similar functionality and, with some time and effort, we're sure they could be used instead.

MS Live Share

For the sake of completeness, we mention that we frequently used the Microsoft extension Live Share while pair-programming. It is not open source.

Development Team Structure

Our development team was composed of five members: Mattia Guazzaloca, Enea Guidi, Lorenzo Liso, Matteo Lorenzoni, and me, Paolo Marzolo. Our roles shifted during the project, but our main responsibilities were as follows:
Guazzaloca: Among us, he knows React best and has worked with it extensively in the past. He was the main author of the inner workings of the front end, and its relationship with the server code. He also set up most of the project structure.
Guidi: Above all, he worked at the final redesign of the UI. Before that, he was the main author of the email notification section and worked closely with Lorenzoni with other sections.
Liso: Liso took on the role of sys admin and is the main author of the server-side code, especially in the early stages. He also wrote a script for collecting geo-localized tweets from his home Raspberry Pi.
Lorenzoni: Most of his work went into integrating various libraries into our main codebase, such as the word cloud, the filters and the graphs. He also helped with the UML and diagram-related parts of the sprint reports.
Marzolo: As a Scrum Master, I handled Interactions with the Product Owner and tried to steer us towards a well-applied Scrum. During the sprints, I mostly worked in pair programming with Guazzaloca and Liso, focusing on twitter authentication, tweeting, and the map.

Scrumble Team-building Exercise

We report the original report, in Italian.

Match 1 (11/10/2020)

Per la prima partita, abbiamo organizzato un meeting parzialmente online domenica 11/10 pomeriggio: Liso, Guazzaloca e Marzolo insieme di persona, con la partita gentilmente hostata da Guidi e trasmessa su Teams anche a Lorenzoni.

Le regole ci sono state inizialmente un po' oscure. In particolare, non eravamo (o siamo) sicuri di quando rimuovere dalla board le pedine User Story, e se il team debba ripartire da zero. Una discussione al riguardo ci ha fatto scegliere di testare, in due sprint successivi, entrambi i metodi che abbiamo considerato possibili.

L'opinione complessiva è che la fase più interessante (e con più interazione) sia quella preliminare allo sprint: l'assegnazione dei pesi, l'ordine delle priorità e la scelta di quali storie affrontare per sprint. La parte realmente di sprint, per citare uno dei giocatori, "la potrei simulare in python".

Purtroppo, non abbiamo svuotato l'intero product backlog. Abbiamo giocato per un totale di 4 sprint, e avendo già visto tutte le meccaniche del gioco (e avendo già perso uno dei giocatori per mancanza di tempo) abbiamo deciso di terminare la partita.

La parte più utile per me, in qualità di Scrum Master, è stata la necessità di testare le mie conoscenze: le differenze tra scrum e il gioco hanno fornito un ulteriore spunto di riflessione che mi ha permesso di riempire i (molteplici, come mi sono reso conto) buchi concettuali che mi erano rimasti.

Sprint documents and organization

After the thorough descriptions of the previous pages, in this section we will go through most of the documents related to each sprint. Every sprint will contain:

- Sprint Backlog: this represents the stories that were completed during this Sprint. I want to make it clear these are not necessarily the ones that were picked at the start; this is especially true for sprints 2 and 3, as we had to move the main US of sprint 2 to the third and the same happened for sprint 3. This is documented in the respective sprint reports. The four US that were included but later rejected will be marked with red color.
- Sprint Final Activities Report: this contains the report that was completed and submitted after every sprint. The structure of it varies among sprints.
- SonarQube Usage Report: as the name says.
- Sprint Burndown: the graph of how points were collected during sprints.
- Additional Information: optional space for comments and observations not included in the report.

First Sprint

Sprint Goal

Main Codebase, few features!

Sprint Backlog

id	subject	description	Points	N° task	Time estimation	Time effective	Acceptance Criteria
1	Development server setup	Have a server that can be used to run self-hosted versions of services.	28	3	20	17	service in cas and our application needs to be accessible from the external
2	Cas setup	"Verify CAS is working as intended, include it in self-launch deps: 1(done)"	17	6	5	5	all service available in the cas enviroment are up and running (excluded bugzilla and logger)
3	SonarQube setup	SonarQube setup, to run in our server. Acceptable to find alternative. Every developer also has to install SonarLint.	16	4	5	5	we can test our git repository of the project with sonarqube
4	Taiga setup	Taiga has to be set-up in order to run self-hosted in our Google Cloud server. It is acceptable to find an alternative; OpenProject works and is fairly easy to setup but does not	29	2	10	11	"Taiga is up and running on the self-hosted gcloud. basic features are supported. Developers understand rudimentary usage notions and promise to use it <_< OPTIONAL: event support,

		allow boards in the free version.					download support, gitlab integration"
5	API self-study	Everyone has to have a basic understanding of how Twitter API v2 + v1 work.	1	13	5	3	If you understand the backend, you're good.
6	User Last Tweets	"As a user, I want to be able to input a username and receive a list of the last few tweets he sent. Was initially included in sprint 1, later removed in grooming and made optional"			8		a user can input a twitter username in the main web page of our application and his last tweets are showed
7	HashTag Last Tweets	"As a user, I want to be able to input a HashTag and receive a list of the last few tweets that mentioned it. Was initially included in sprint 1, later removed in grooming and made optional"			8		unlikely to be completed, as historical search is not currently supported. either way, i put in a hashtag and a list is updated with the LAST tweets that contained it.
8	Trending Hashtag	As a user, I want to be able to see a list of the Trending Hashtags.			8		trending hastag are showed on the main web page in a list
9	Word Cloud to map	As a user, I want to be able to select a term from the wordcloud and see a map of where the collected tweets were sent from.			20		pretty clear from the description: after clicking on a term in the wordcloud, the interface gets updated to show, on the map, only the tweets that contained that word
10	Mattermost setup	setup mattermost on self-hosted gcloud	6	1	4	1	mattermost is reachable by both browser and phone, and running on self-hosted gcloud
11	Starting Geolocated Filtered Stream	As a user, I can input a coordinate box and start collecting tweets coming from that box.	20	6	15	17	After inputting coordinates, as a user I can receive the tweets that were collected. OK both in bulk and single tweets.

First Sprint Final Activities Report test

Introduction

This is a report containing concise descriptions of the first Sprint Review and Sprint Retrospective. Due to the unofficial “Didactic Scrum” we’ve been following, we have decided to merge our reports for the Review and the Retrospective in a

single document, even though they'd be intended for different purposes and different readers. The document has a simple structure: in the first part we'll consider the review, and in the second part we'll take on our retrospective. Most of the content of the two meetings has no place in a formal document, so I only gave on overview of the points discussed and focused on how to better the two processes in the future.

As with most other "official" documents, we decided to write the final report for the first sprint in English. Please let us know whether we should change this in the future.

Sprint Review

Duration and General Approach

We decided to hold the sprint review on Sunday the 1st of November, although the sprint officially ended on October 31st. Since the sprint review is intended to include the PO, we made a short video outlining the main features of the increment. We understand the time the PO can dedicate to each team is limited, so we decided to delay PO approval until the next Sprint Planning to avoid having to schedule two separate meetings. We found it challenging to properly time the sprint review, as we ran into (multiple) technical issues, but it took approximately 1h 30min. We had two activities we wanted to tackle for the review: making the video and discussing the current state using essence cards. We considered adding other "serious games" and activities, such as the "responsibility game", but the decided against it, as we found it could have easily become an "agreeableness" contest while providing little chance for insights.

Product State

We decided to describe the current product state (although product is the improper term) using essence "Alpha State" cards. As the Scrum Master, I found that these provided the shortest possible path to the resolution of the two main issues I could foresee: clearly describing the product state from various angles and finding concrete measures to better those states. We won't go into details here, as we believe the cards with our added short comments are clear enough, but we're ready to explain if anything is unclear.

From the practical point of view, we provided, together with this Report, a short video with light editing that should help form a clearer picture of what we've achieved so far, and what we focused on. As we outline in the video, we preferred making a minimal viable product to completing a complicated interface with map capabilities but being unable to provide a proof of concept for backend code. Lastly, as points on Taiga show, a sizable chunk of our time went directly into the setup of the development environment, for which we have little to show except our working services.

Problems completing the Review

As I mentioned in a previous paragraph, we ran into a few issues. I'll outline the main ones here and describe how we resolved them.

1. Picking a video conferencing app. We assumed this would be an easy task: we've been using Discord among ourselves and we use Teams in class every day, but since the spirit of the project is to strongly prefer open-source alternatives we tried Jitsi Meet. We knew there were some open servers on iorestoacasa, but we had various issues, ranging from connections problem (a classic) to some members being unable to hear everyone else, so we switched to Discord. In the future, we may try Jitsi again, but I, as scrum master, will need to take some time with the team to test it before we need it.
2. Picking a collaborative editing diagram maker website. Since I had been looking for a while, I knew of a few different alternatives: we settled for this one, and then it took us quite some time to share the document with the whole team.
3. Recording the video. Because of our remote nature, we had to record the video by recording the screen. This made things easier for what concerns the "video"side, but complicated the audio, for reasons I won't go into here. We recorded a few different versions with varying presets and then I ended up splicing together different pieces to avoid rambling and long pauses.

As one can see, we struggled quite a bit with every step of the process. Surely, it was a learning experience for all of us (especially myself, who thought that, just by knowing the name of the services we'd be using, we'd have no problem

setting them up quickly), but I must admit I would much rather have avoided these issues completely. Personally, as the Scrum Master, I'd like to point out a fourth issue, more fundamental than those raised so far: while the past three are technical, mine concerns the way remote work is completed.

4. Scrum as a remote work development framework. I found scrum as a collection of techniques and beliefs extremely interesting, but I don't think they generalize well "directly" to the remote environment. All our activities feel a bit forced, and it's honestly quite hard to coordinate tools that are made to emulate what in real life would just be physics and a whiteboard. I'm sure, as COVID keeps on ramping up, new suggestions will come up, and I will get better as a Scrum Master; at the same time, I wanted to underline that I truly wish we could do this in person. Tools shouldn't make it harder to get stuff done, but during our review and retrospective I've felt this way multiple times.

Sprint Retrospective

Structure and activities

We held the retrospective the afternoon following the review. This was meant to help the team voice their doubts about the current state of the product, but ended up putting a strain on all of us due to the amount of time we had to dedicate to both processes. This led to some adjustments by me, the scrum master, in order to shorten the retrospective and make it less painful. I decided against formally implementing the PO approval process with essence cards: I introduced this in class and was hoping to complete it during the retrospective. Instead, after quickly introducing the retrospective as a process, we jumped into practice patience, which took some time due to (a) the team being very vocal about what they cared about and (b) the selected tools, once again open source in the form of diagrams.net (formerly draw.io), couldn't deal with my horrible internet connection. We included the result in the bottom half of the pdf file on Taiga. Then, we focused on giving actionable feedback to improve on the problems we found, by pretty much following this "Follow Through" advice. We decided against writing down every probability and suggestion and focused on communication and dialogue.

Issues and Solutions

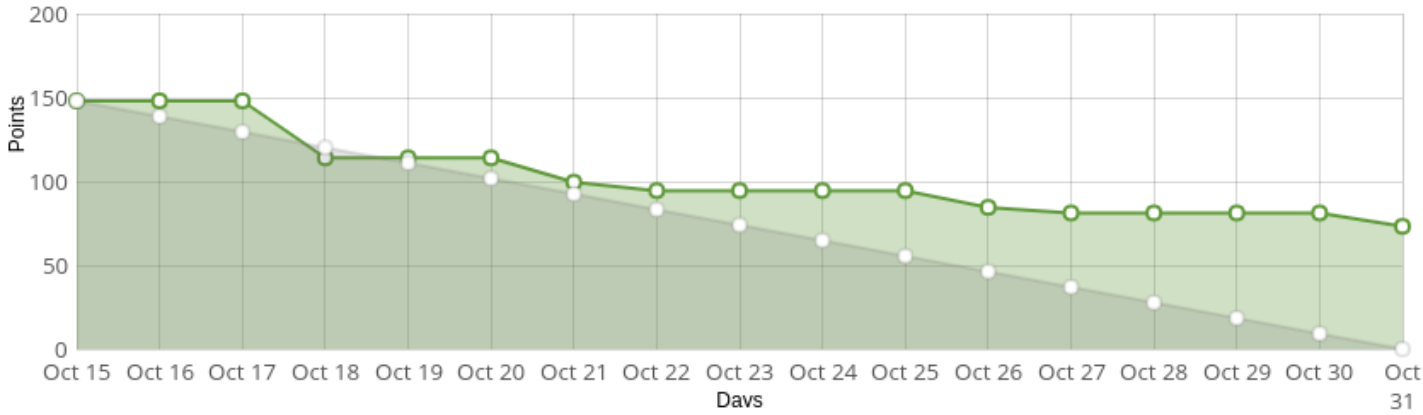
Once again, we struggled with emulating what would come perfectly natural in the real world with online tools, which made the process, already difficult due to the novelty of it, somewhat tedious for all parties involved. Due to the multiple meetings we needed (Review and Retrospective), not all members were available for the retrospective, which obviously hinders the usefulness of the process itself. In addition, I couldn't get through to part of the development team: I had hoped everyone would see the advantages of working like this, but clearly, I couldn't convince all of them. I firmly believe the retrospective is a fundamental part of development process, especially when working with talented individuals such as our team, as developers often already have a workflow which is rarely focused on team effort. As a scrum master, I want to be able to convince them not of the importance of scrum or essence themselves, but of having a moment to look back at the way we worked recently and realize, together, how to better ourselves and our workflow.

SonarQube Usage Report

We did not run SonarQube on the first Sprint.

Sprint Burndown

As one can see from the graph, our velocity was not constant at all. This is because we overloaded the Backlog, underestimating how much time tool setup and code boilerplate would take us.



Second Sprint

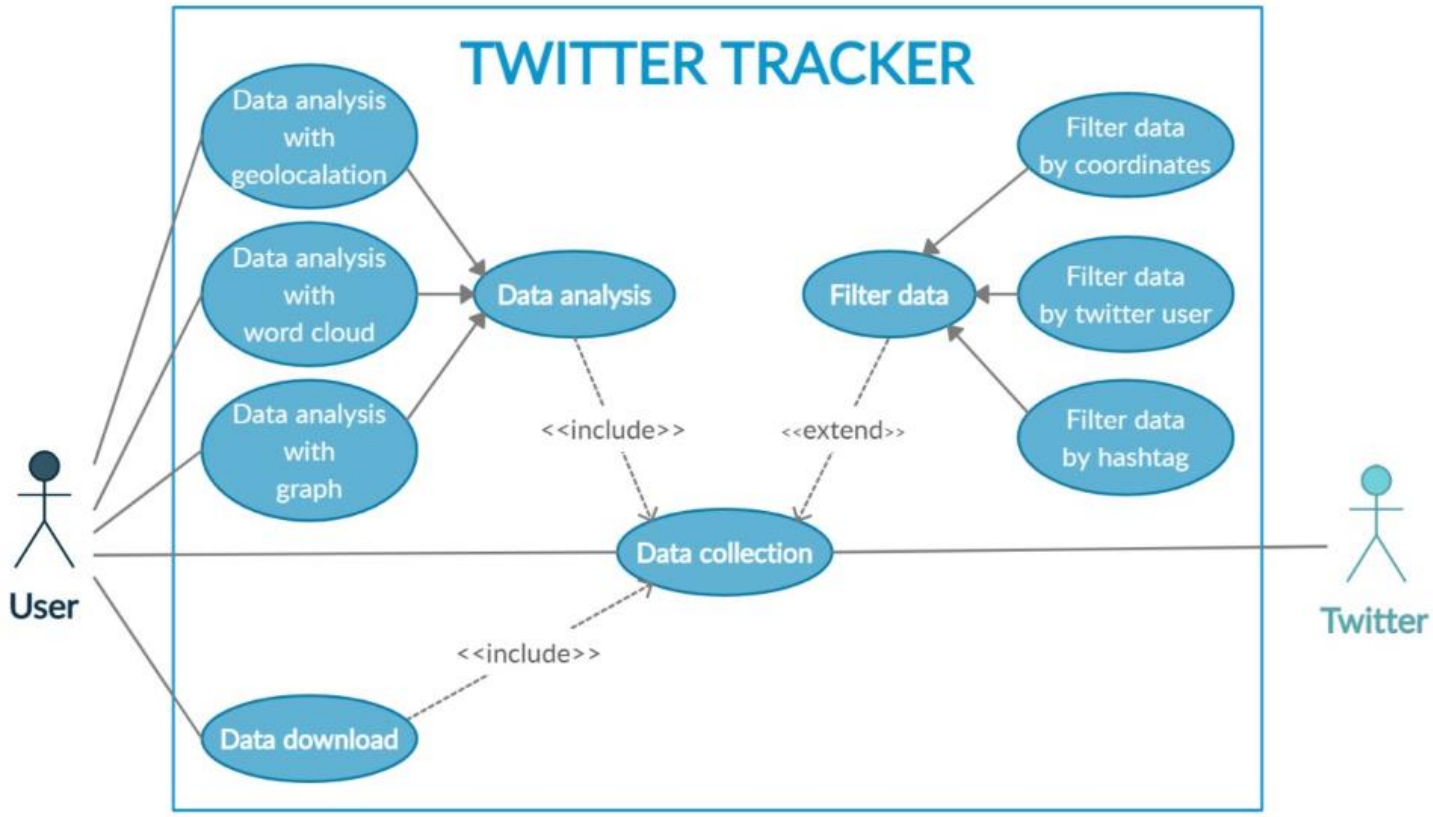
Sprint Goal

Maps and Geolocation are our bread and butter.

Sprint Backlog

id	subject	description	Points	N° task	Time est	Time eff	Acceptance Criteria
12	Export	As a user, after I have collected a number of tweets, I am able to download the collection.	16	4	3	3	clicking the export buttons trigger a download events in the browser of the json containing all the collected tweets
13	Start Building Database	As a developer, I want to have a teeny tiny database of tweets (geolocated, possibly mentioning our hashtag) to: - test stuff out (play with json files, investigate maps in next sprint...) - show that our collecting function is working correctly - build a database for next sprints Database: literally a single or multiple JSON files	17	3	10	5	Somebody somewhere has a database I can gain access to, even if it takes time. It's important it exists, ease of access is secondary
14	Make features REALTIME	As a user, I'm able to see the counter of the tweets increase, a subset of tweets, and some preview in real time as tweets are collected. Consequently, all features are updated in real time	25	4	10	9	Client-received tweets are updated not in bulk but in small pieces (acceptable either time-frame or every tweet)

Use cases diagram



Second Sprint Final Activities Report

Introduction

This report contains a brief overview of the activities we conducted as part of the end of the second sprint: the Sprint Review and the Sprint Retrospective. Most of the considerations we offered in the first Report still hold true, and since we've now developed a workflow for conducting these meetings we will just quickly point out the main thoughts that arose.

Sprint Review

Duration and General Approach

Once again, we held the Sprint Review on the Sunday following the end of the Sprint. Once again, only four members were present, due to incompatible schedules, but the effort we collectively put in was distributed fairly. Like last time, we made a short video documenting the user experience and evaluated the product state using essence "Alpha State" cards.

Product State

While evaluating our process with State cards, we decided we had progressed in the "Stakeholders" alpha but regressed in the "Requirements" alpha. Opportunity, Software System, and Team are the alpha which changed the least, and are the unlikeliest to change in our opinion. While the Requirements regression may seem like a problem, it only signals the growth of our understanding of them. In particular, the Twitter API only allows for selecting tweets who satisfy ANY of the constraints selected: this means that we need to discuss with the PO whether centering the product around strictly geolocated tweets is an acceptable solution; the only alternative would be manually combing through the sizable tweet amount to check whether the coordinates are inside the specified rectangle, requiring a useless amount of effort both in coding and processing power. To discuss this, we will request a meeting with the PO soon.

Video

The video highlights the main features (and consequently, user stories) we implemented during this sprint. Although the interface is still rough around the edges, we believe it presents three of the main components we will implement: the constraint selection form, the map, the tweet list and the word cloud (still absent). All of these have yet to be completed, but the basic feature set is already clearly shown. On the technical side, the video does not show that we have switched from NextJS to NodeJS + Express.

Sprint Retrospective

Practice Patience

For this retrospective, we used a similar approach to last time's and used essence scrum card to evaluate the state of our workflow. The result of this is in the attached table. We decided to start from the position we had reached in the First Sprint Retrospective, as this would allow us to reconsider our progress and internalize what had been improved upon. There are three suggestions we find especially important: increasing PO involvement (a concern that was raised for both Product Backlog and Product Owner items), having better task management (some members of the team find it uselessly time-consuming to record what they're working on and what else needs to be done) and maintaining the balance we've reached in the Sprint Backlog.

For what concerns PO involvement, we will soon have a detailed meeting. Task management is an issue we need to talk about further: during this sprint some members have done additional work which, because it wasn't agreed upon beforehand, ended up having to be discarded. Lastly, the Sprint Backlog was clearly at full capacity, if not underestimated: two user stories will be completed in the next sprint, because PO input was required in the last few days of the sprint.

Personal Evaluation

There are three main other areas of concern we'd like to bring forward: the necessity and accuracy of productivity metrics, the irregularity of our progress and the difficulty of turning scrum values and suggestions into actionable feedback.

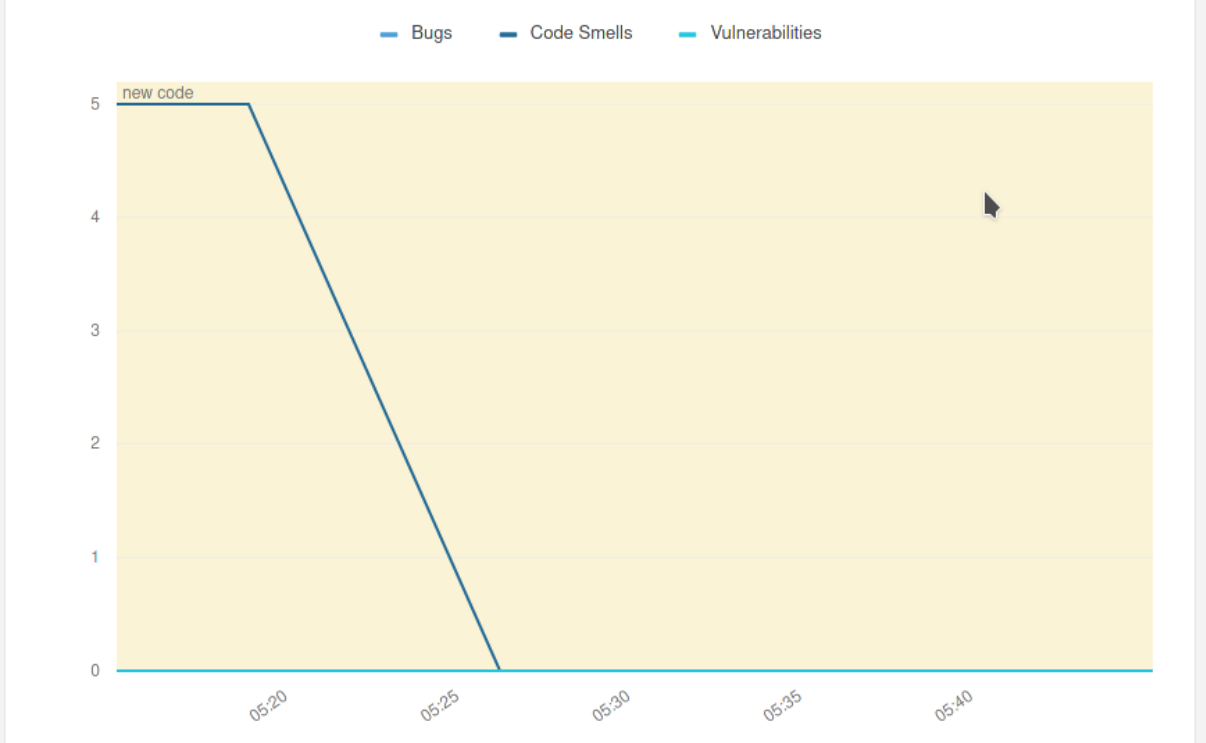
For what concerns productivity metrics: we have realized Wakatime only allows to download statistics about the last 14 days; this creates yet another necessary manual intervention upon an automatic tool. We're unsure whether we should switch to a different system or create a simple script to update a shared repository: both options require a substantial amount of research that we weren't willing to put in, while in the final week of the sprint. This means that we will need to discuss this issue with the professor and find a solution that can satisfy both our needs.

The irregularity of our progress was one of the points that Wakatime allowed us to raise. We don't believe it is an issue in itself, as it just highlights how this project can only claim a portion of our time. At the same time, I believe it is deeply connected with our difficulty with coming to terms with the ratio of time spent in meetings compared to coding. This is what I meant with our third area of concern: I, as Scrum Master, don't believe the way we're applying scrum makes sense, due to the low amount of time the team members are physically able to pour into the project. That said, we will continue to follow the PO and professor's guidelines.

Lastly, I'd like to provide a short list of the documents that will be submitted for this sprint: the UML diagrams (Use Case and Classes), along with an explanation; the video highlighting our progress; the diagrams resulting from Practice Patience and Essence Alpha State Evaluation; a short outline of how we used SonarQube; this report.

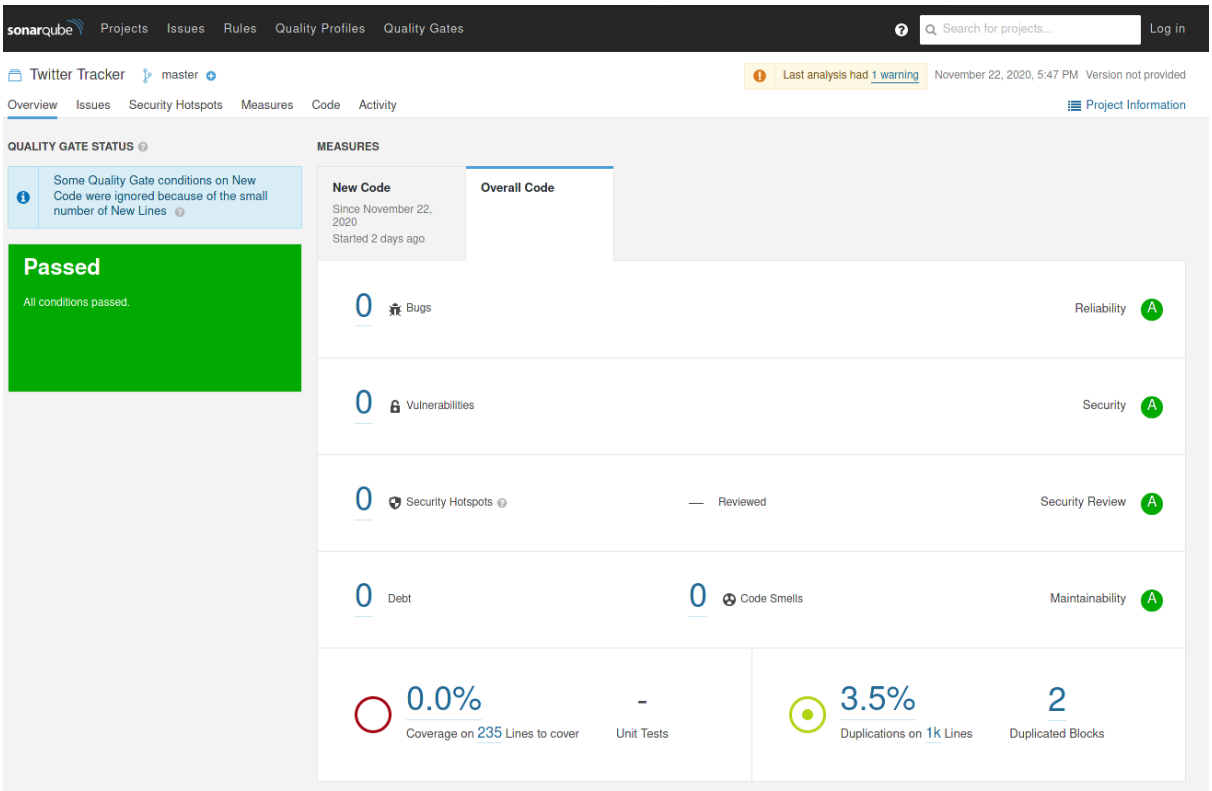
SonarQube Usage Report

We only used SonarQube at the end of the sprint: there wasn't a specific plan behind this, but a few of us had been "scolded" by the initial setup of many of these tools (we had already set up SonarQube as well), so nobody was especially excited by the idea of getting back into it. As it turned out, after the initial setup there really isn't much to it, so we quickly completed the final steps and ran our first scan: this revealed 5 "code smells" that we resolved easily, and one "possible security issue" (the use of CORS) that took a few minutes and one extra file.



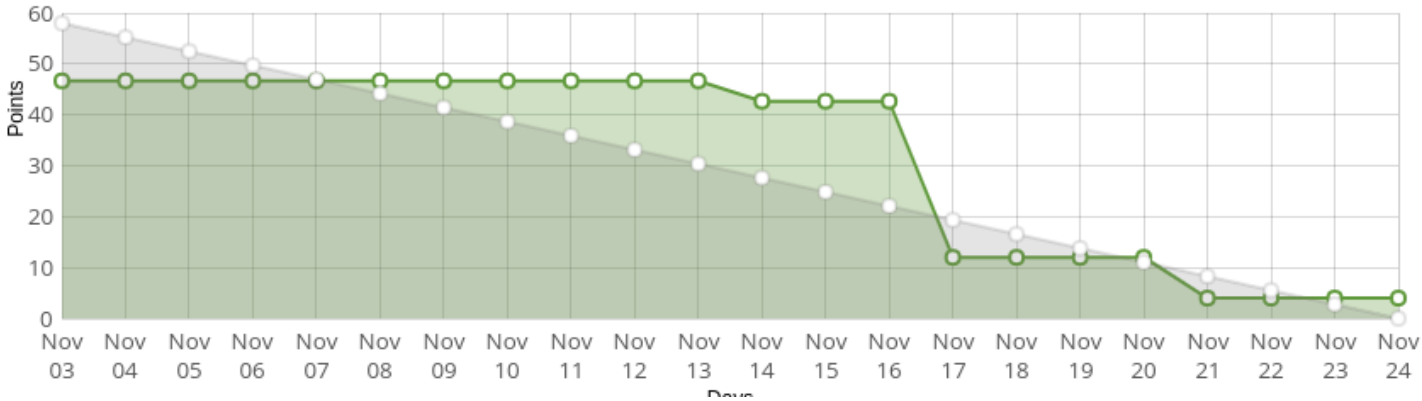
Current Status

What follows is a snapshot from one of the main views of SonarQube. As said above, we have resolved all the issues SonarQube revealed, and the only code duplication remaining is for a file containing a couple tweets' JSON files for testing. We believe SonarQube was fairly useful, and our most interesting finding was the possible security issue.



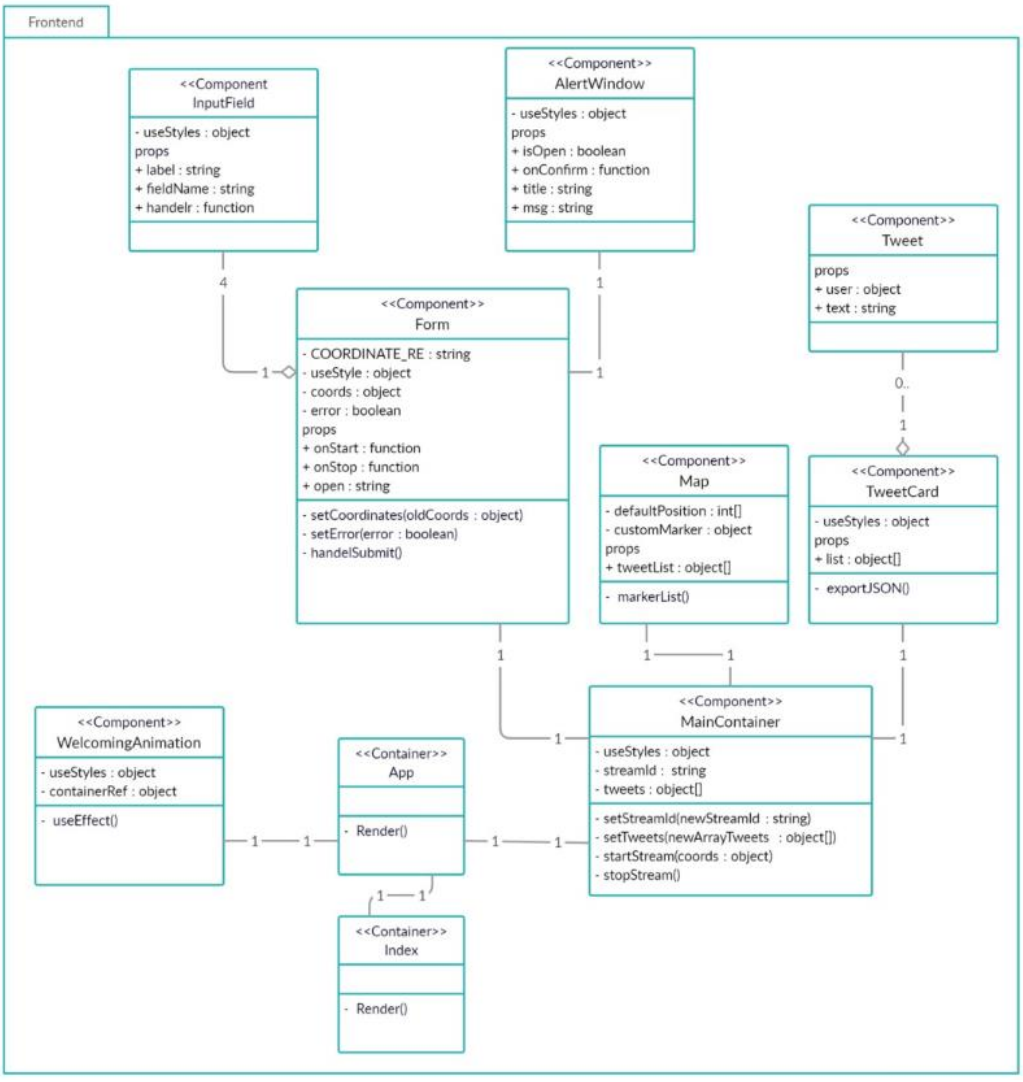
Sprint Burndown

This sprint burndown showcases one of the weaknesses of the point burndown: the remaining points are a rough estimate, and when tasks are tightly correlated (which means they flow together though the stages) estimated points don't really reflect the progress.



Additional Information

Class diagram



Third Sprint

Sprint Goal

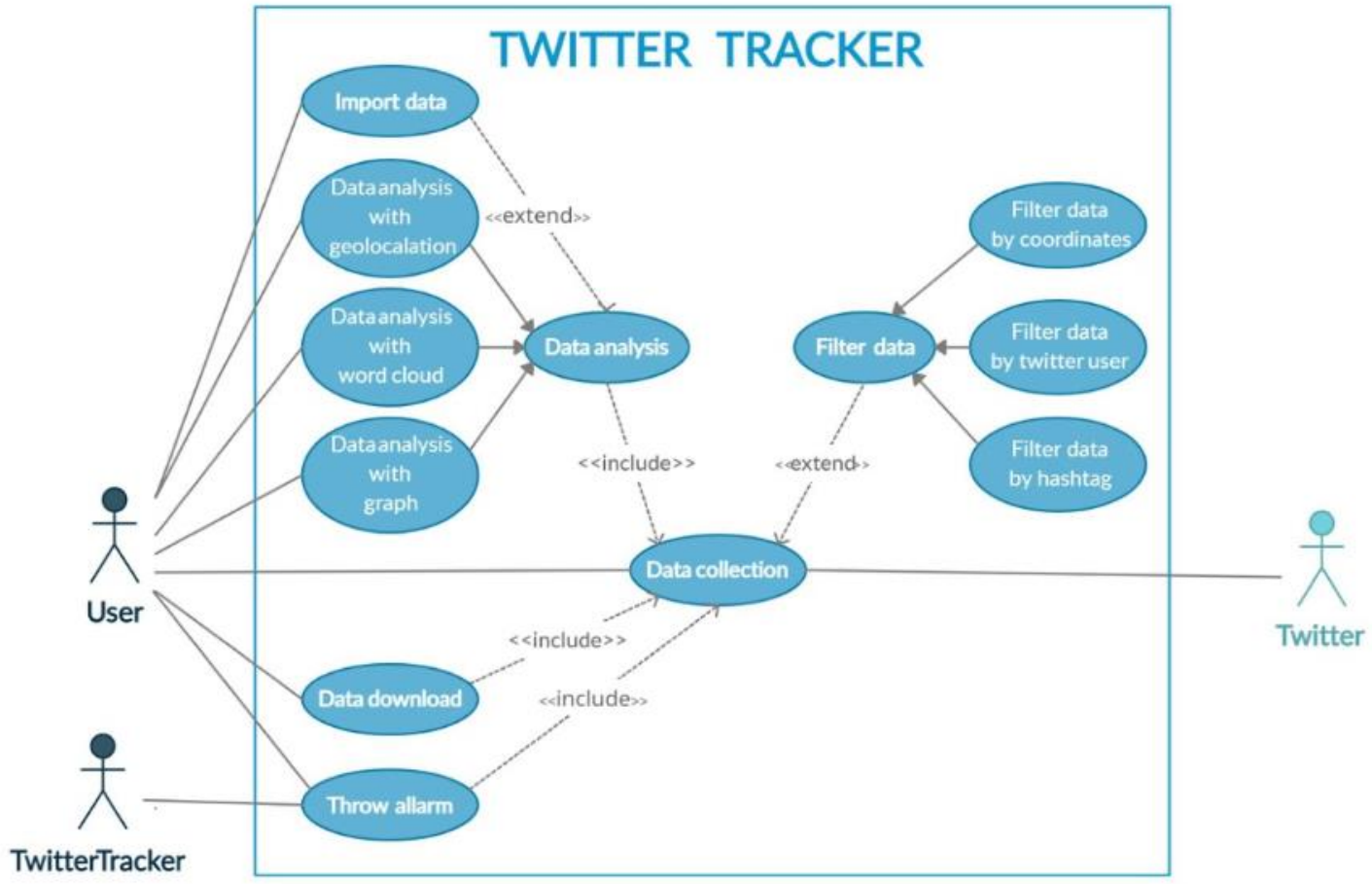
The perfect workflow is here?

Sprint Backlog

id	subject	description	Points	N° task	Time est	Time eff	Acceptance Criteria
15	Word Cloud from stream	as a user I can view a word cloud with terms used in the tweets in selected stream	17	8	10	15	a word cloud is correctly generated and showed on client screen, updates with new tweets
16	Map visualization of hashtags	"As a user, I want to be able to visualize geolocated tweets from one or more hashtags on a map "	28	13	36	27	"after the stream is started with an hashtag, as tweets are received a clickable marker is showed on the map. Clicking on the marker shows the full tweet. If too many tweets are made in the same place, just one marker is showed."
17	Import tweet collection	As a user, I am able to upload a collection of tweets to "analyze"	16	3	10	7	a json previously exported can be imported and tweets are showed in the map
18	Time tracking for prod metrics	"Consider various alternatives to collect time tracking data, in particular https://wakatime.com/integrations and https://www.kimai.org/about/"	18	3	3	6	Time tracking tool approved by POs
19	Map visualization of users tweets	As a user, I want to be able to visualize geolocated tweets from one or more users on a map	21	2	10	6	"After starting a stream and collecting some tweets, I can push a button (?) and a map opens, showing the locations of the tweets collected. ALTERNATIVELY As tweets are collected, their location is shown on a map, which gets progressively populated."
20	Photos from stream on map	"As a user I can insert an hashtag and see (eventually also download) a list of phots that are insert in the tweets that mention my hashtag. Extra points: place photos corresponding to tweet on map"	20	3	30	9	After inputting a hashtag, I can collect some tweets that mention it. Afterwards, I can download all images contained in bulk. Test with: #apple
21	Notification on threshold	As a user, I want to be able to set up a notification when a certain condition is met,	15	3	12	8	FIRST TEST: condition based on amount of tweets: when enough tweets are collected, send out notification by email.

		of which the details are left to the implementers, which notifies me by email.					

Uses cases diagram



Third Sprint Final Activities Report

Introduction

Once again, we remind the readers that most of what we explained in the previous two reports still holds true. The contents of the report do not change in structure: in the first part, we'll outline our main takeaways from the Sprint Review, and in the second part we'll highlight our criticisms of our workflow as they emerged in the Sprint Retrospective.

Sprint Review

Duration and Approach

This Review was especially short: we were all very up to date on what had been achieved and were eager to be done with the ritual and back to our regular lives, as the project is nearing its end. As the Scrum Master, since the team was finally comfortable with the "Product State" process, I introduced a new short segment, as a means to create a new outlet for thoughts and opinions. This was only accepted by part of the team, as a couple member were, if not critical of the idea, somewhat uncooperative. In total, it lasted about 45 minutes, on Monday the 14th of December.

Product State

As I mentioned, our use of Essence Cards was quick and efficient. We identified a growth in 4 Alphas: Stakeholders, as expectations and requirements seem to now be finalized, and we've received useful feedback from a meeting with the

PO; Opportunity, due to the solution being finalized and the discussion about the constraints we had with the PO; Way of Working, because of the advances we made in task division, parallel progress and integration; and finally Requirements, although, as we noted in the attached document, the quality of our work isn't yet up to our standards.

Video

The video showcases our main advancements, on multiple fronts: the Word Cloud, the image handling on the Map, and the email notification at threshold. We've decided to avoid showing the scheduling widget, because it was only implemented on our last day, and has yet to complete the steps and testing required by our DoD (which means it is working on a feature branch, specifically `feat/periodic`). We also quickly completed what was left from the Second Sprint (part of the tweet visualization on the map) and the tweet collection import.

Team Feedback

Lastly, I would like to highlight some of the feedback that was given in the last part of our meeting, in a few simple anonymous quotes.

"Interface definitely needs to be reworked. clearly, we focused on getting features done, but I worry the responsive part will take longer than we think." Due to this feedback, we created a new product backlog item, which has yet to be discussed with the PO, for picking and completing a compelling interface.
"I really wish we could have had a few more days to complete our User Story. Even just one day would have meant not carrying it over to the next sprint..." As the Scrum Master, I'd like to receive some feedback on this from the instructors: would it have been okay to delay the end of the Sprint? All of the development team new this was not due to negligence, and the solution of moving it to the next sprint is handled horribly by Taiga.
"The DoD helped us catch a sneaky bug on slower connections that I'd have ignored otherwise." This genuinely made me proud of our advances, so I decided to include it.

Sprint Retrospective

Practice Patience

The Retrospective was held right after the Review, but the team was cooperative in keeping comments related to our workflow for the second part of the meeting. We completed Practice Patience again. We can gladly announce we have no scrum cards in the red state anymore! Some important notes include: the clarity of the Product Backlog, underestimating the effort and time required for the periodic branch, and the enhancement in task division. The first one is partly due to some time we took to focus solely on this: I found that assuming the Team will clean up the backlog is wishful thinking, so I sat down with a cooperative member and combed through our backlog to increase its readability, completeness and usefulness. The second one is related to the same issue that was mentioned by a team member in the quotes, which annoyed us all. The third one was probably the most important one: due to a better understanding of both the strategy behind Scrum and the tools at our disposal, we were able to distribute the work among ourselves fairly evenly, and work much more efficiently than in the first two sprints.

Team Feedback

As I did in the Sprint Review, I asked the dev team to express some thoughts about our workflow. Here are some of my picks:

"Due to our initial division, I mostly worked by myself in the first part of the Sprint, focusing on my feature [word cloud] and only requiring feedback as part of the DoD and when merging in master. This worked well, as I did not have to coordinate too much with other members' tight schedules." This was important feedback for me: in the last sprint, some members mentioned they had been slowed down by the cooperation overhead, and I would be lying if I said I never felt the same way.
"Feature division in branches cleaned up our workflow." As I mentioned before, I agree with this statement. I think it is especially relevant because changes to the DoD can have a large impact, so I'm happy it worked well.
"Because I had to interact with parts of the repo I wasn't familiar with, I asked for help to the other members, and they quickly delivered useful feedback." I felt it was important to include this, as I don't want the first quote to mislead the

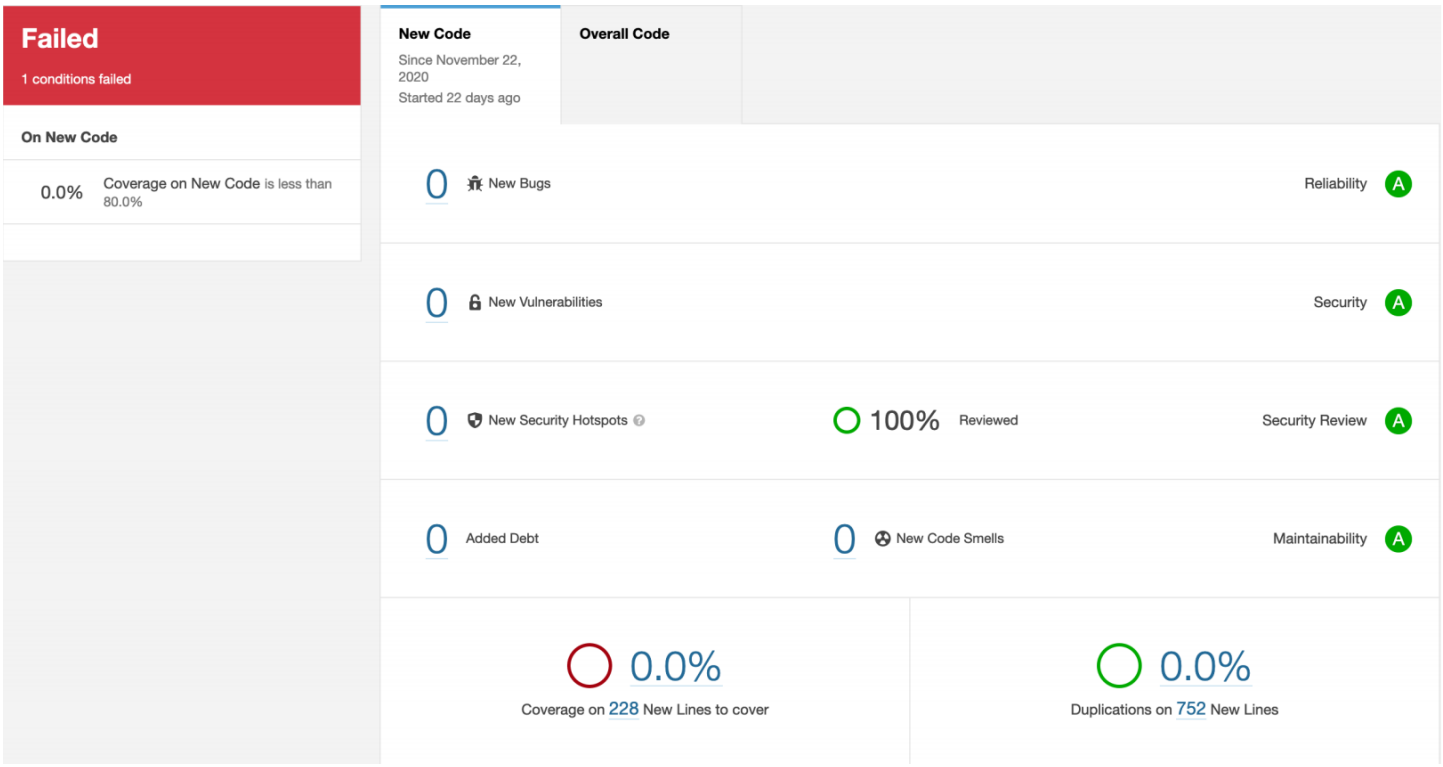
reader into thinking our cooperation has only been lowered: we have removed unnecessary overhead and sped up necessary interactions.

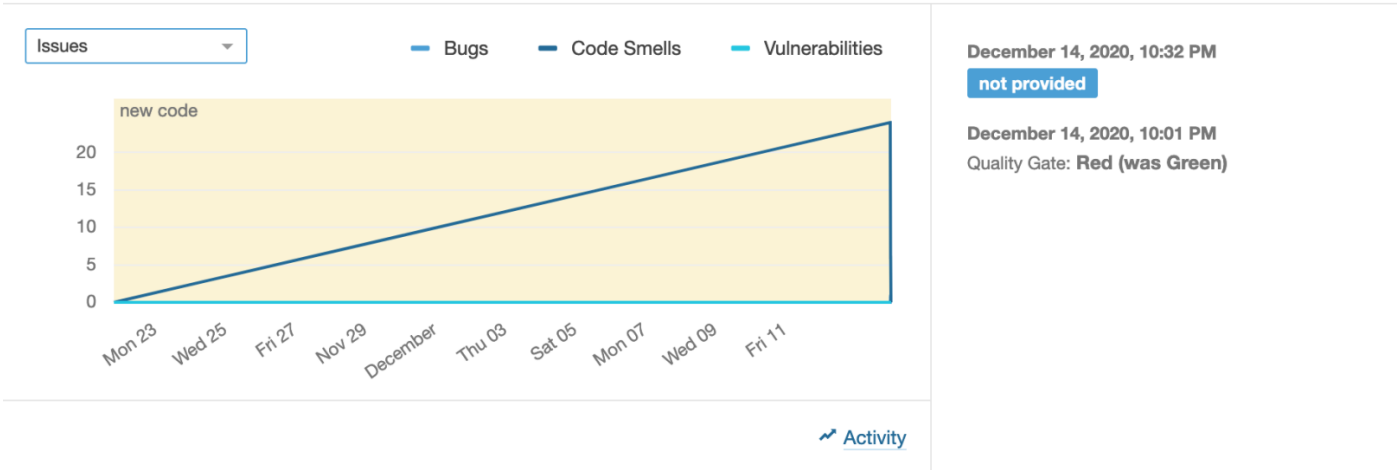
Conclusion

I believe this was our most successful sprint yet: I hope the Taiga mishandling of moving US to following sprints won't lower the perceived success by the PO. Time tracking was strictly followed by the whole development team, and the relevant files will be uploaded on the repo as exported from WakaTime. All the other documents are attached to this one, in the same wiki bookmark.

SonarQube Usage Report

As in the previous sprint we decided to use SonarQube only at the end because we believe it is a very useful tool to refine the written code. We then performed a first scan of the code to detect any bugs and code smells that we proceeded to fix, as you can see from the following snapshot. The reason for the failure of the report was due to the fact that within the project we are not using any testing framework and therefore coverage value is equal to zero. In addition, we were able to find a way to exclude some files from the scan and this has allowed us to reduce to zero the lines of duplicate code that were due to a JSON file containing a dump of some tweets we used during development.

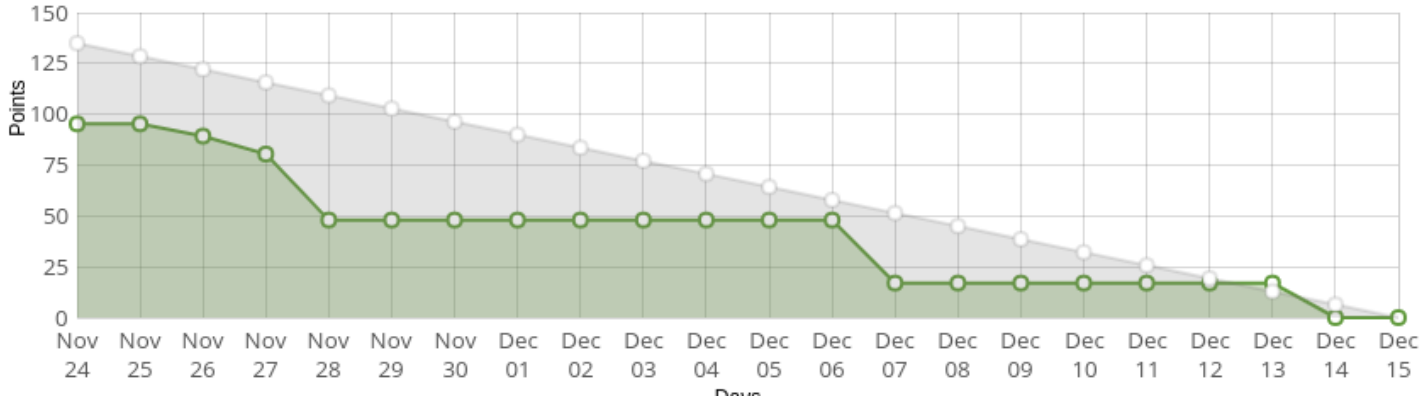




Sprint Burndown

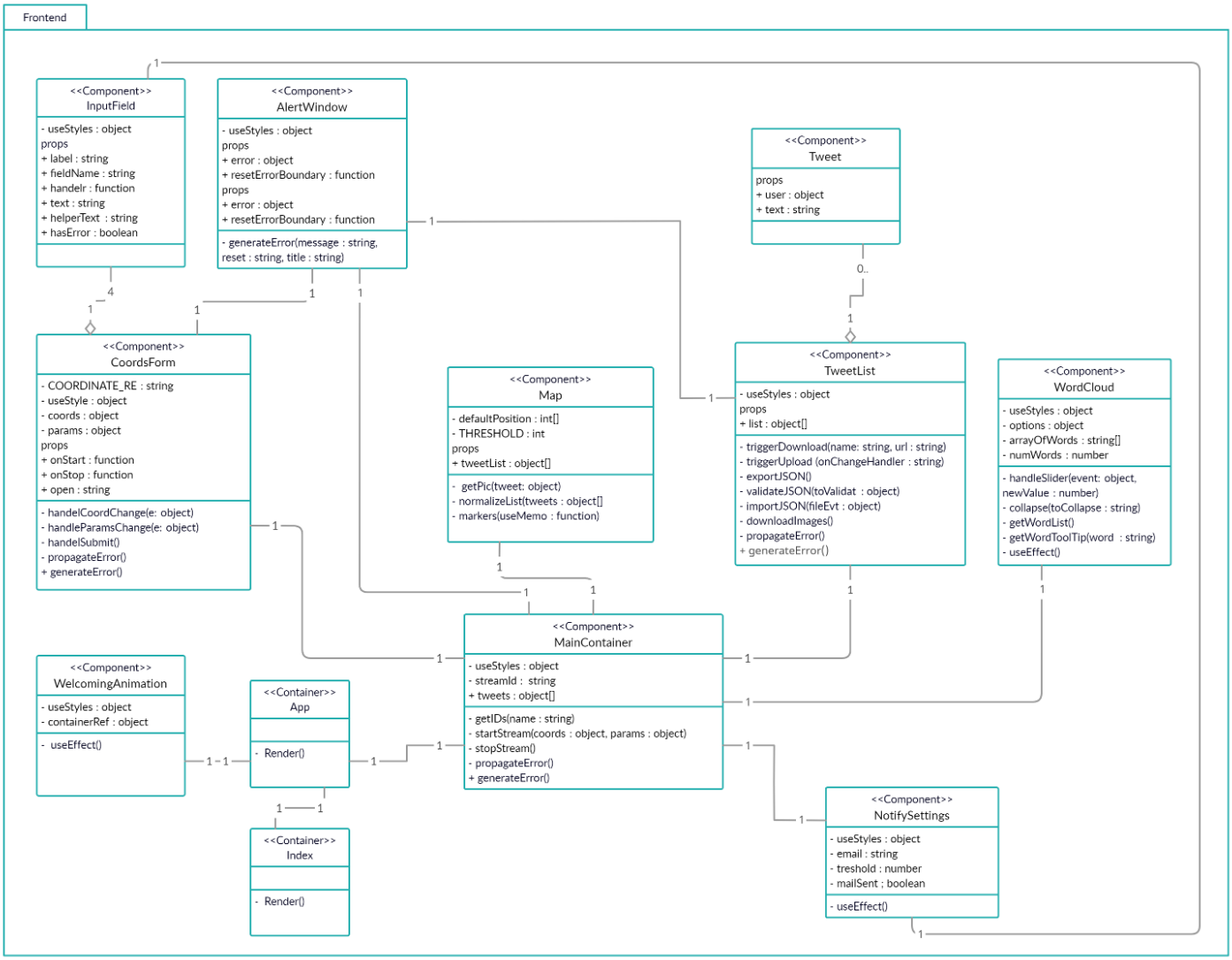
Because of the task we moved from sprint two to sprint three, the initial point estimate was lower. Then, because one more US was moved from sprint three to sprint four, the entire graph lowered by the amount of points that story had.

In addition, it is clear from the diagram how we cannot work every day, and instead tend to focus on long weekends.



Additional Information

Class diagram



Fourth Sprint

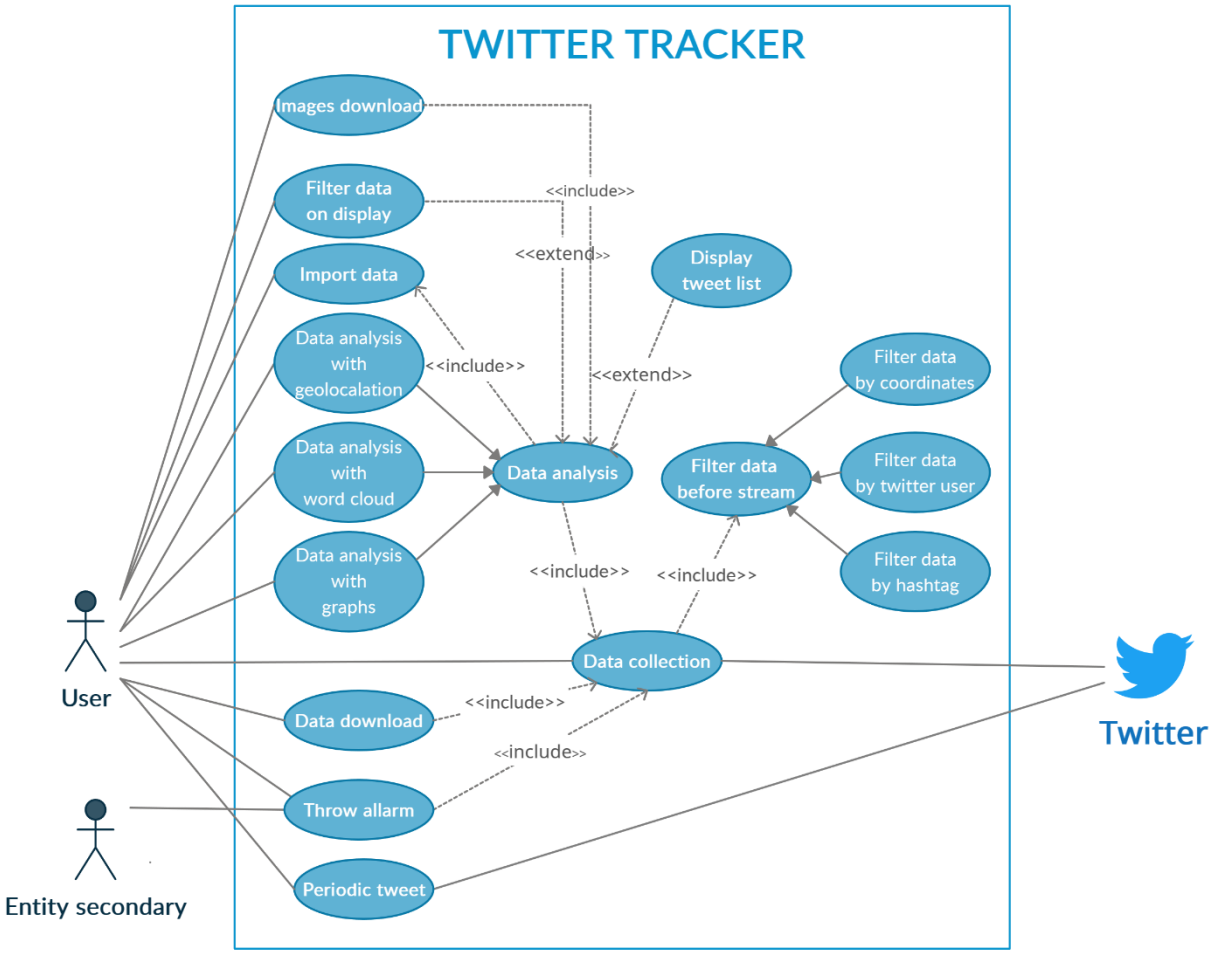
Sprint Goal

Let's get this cleaned up.

Sprint Backlog

id	subject	description	Points	N° task	Time est	Time eff	Acceptance Criteria
22	Tweet amount chart by time	As a user, I want to be able to search a hashtag and see the volume of tweets by unit of time in a given period, to judge the impact an event has had.	23	6	25	17	UNLIKELY. a graph shows the volume of tweets received per unit of time. the tweets are the ones that respect the filters
23	Periodic tweet	As a user, I want twitter-tracker to automatically and periodically tweet one or more of the analysis on a stream I started.	30	13	30	40	A tweet is sent out, FROM MY ACCOUNT, with the information I selected. I can setup the tweet, test the authentication worked, see the changes both in the client and the result in the tweet. Timeframes: a week maximum, for sure. minimum is hours.
24	Background searches	As a user, I want to be able to start a search and close the window, then find it again when i log back in.	20	7	15	17	The browser window can be closed, and there is some identification method that allows me to "log" back in and see the updated search
25	BIG interface rework	As a user, I want a nice and complete interface.	31	5	22	21	Rework the interface completely. Main interactive part will be the map. include common user interface for email/tweet/filters.
26	Make coordinate interactive	Find a way to make input of coordinate interactive. it can be two movable markers, drawing a square, or selecting from a list of pre-drawn country squares	23	9	17	18	Will have to be defined when a good compromise between doability and effectiveness is found. UPDATE: after drawing a rectangle on the map, i can start a stream which has as constraints the correct coordinates.
27	Filters on data	As a user, once the data is displayed, I want to set more descriptive filters than those used during the stream	15	7	20	19	Once the filters are selected, the number of tweets displayed increases or decreases depending on whether they respect the filters entered. changes are applied to the whole list, so they affect all visualizations.

Use cases diagram



Scenarios

Below are some of the scenarios created relating to the diagram of use cases of the fourth sprint.

In total nine scenarios have been created, only three are put here as demonstration. If you want to see them all you can find them in the relative pdf.

USE CASE: TWTTR_TRCKR03	**DATA DOWNLOAD**	**Date:** 05/01/2021
		Version: 0.01.001
Description:	It allows the user to download the collected data	
Priority:	Medium	
Duration:	Seconds	
Primary actor:	User	
	His interest is to download the data that he had just collected and that he wants to save them, so he can see them again in the future	
Secondary actors:	(Empty)	
	(Empty)	
Precondition:	User has already started and finished the data collection phase and he did not receive ad empty collection, so he has some material to work on	
Failure guarantees:	No data is downloaded, the button used for this task cannot be clicked	
Success guarantees	All collected data are downloaded in a json file	
Start:	The data download button is pressed	

Main scenario

	USER	SYSTEM
1.		It check if tweet list contains at least one tweet, if the verification is confirmed then it makes the relative button available
2.	He presses the download button	
3.		It download the collected data to the user's personal computer

Error scenario

Tweet list is empty (no tweet avaible to download)

	USER	SYSTEM
1.1		It check if tweet list contains at least one tweet, if the verification is not confirmed, the relative button is still not available
2.1	He can't press the download button	
-	END	

USE CASE: TWTTR_TRCKR01	**DATA COLLECTION**	**Date:** 05/01/2021
		Version: 0.01.001
Description:	It allows the user to collect data from Twitter (such as tweets or photos) and then view, analyze or download them	
Priority:	High	
Duration:	Variable (depends on the need of the user)	
Primary actor:	User	
	His interest is to get data to work on them	
Secondary actors:	Twitter	
	His interest is to provide data to those who request them through public API	
Precondition:	Working Internet connection and both the application and Twitter should not be under maintenance therefore not guaranteeing its services	
Failure guarantees:	No collection is made. Thus no data are collected (therefore not even visually)	
Success guarantees	The request to Twitter is confirmed and the requested data from the user are collected correctly by the application	
Start:	The button of start collection data is pressed	

Main scenario		
	USER	SYSTEM
1.	He opens the application	
2.	He sets the filter inputs	
3.	Starts the data collection by pressing the start button	
4.		It checks if the three filters type contains at least one value
5.		It opens a communication channel with Twitter through its API
6.		It sends to Twitter the filter inputs
7.		It starts collecting all information received from Twitter
8.		It listens to a signal from the user indicating to end the collection data
9.	He decides to stop data collection	
10.		It closes the communication chanel opened with Twitter
11.		It makes the collected data available for analysis or download

Error scenario User doesn't insert filter values		
	USER	SYSTEM
4.1		It sends relative error message
4.2		It doesn't start the collection
-	**END**	

Alternative scenario Tweet list isn't empty		
	USER	SYSTEM
11.1		It appends new tweets with thoose collected with previous collection, forming a single big list
-	END	

USE CASE: TWTTR_TRCKR02	DATA ANALYSIS	Date: 05/01/2021
		Version: 0.01.001
Description:	It allows the user to display data collected (from stream or import) with three different types of visualization: displaying tweets and photos into a map (geolocalization), displaying a word cloud (populated by the words present in the collected tweets) or displaying some infromation through different graphs	
Priority:	High	
Duration:	Seconds	
Primary actor:	User	
	His interest is to view the data that he had just collected and that he wants to analyze in some form	
Secondary actors:	(Empty)	
	(Empty)	
Precondition:	User has already started and finished the data collection phase and he did not receive an empty collection, or he has just imported a json file containing a collection of tweets previously downloaded	
Failure guarantees:	No data is displayed	
Success guarantees	All collected data are displayed in the various forms available	
Start:	When button of data collection stop is pressed or when the import phase is completed	
Main scenario		
	USER	SYSTEM
.	(He has just pressed the button of data collection stop or he has just importend an own json file)	
1.		For each form of visualization, it selects what infromation type (e.g. : photo, twitter...) to display
2.		It displays the collected data
Alternative scenario Tweet list is empty (no tweet avaible to display)		
	USER	SYSTEM
2.1		Doesn't display any data
-	END	

Fourth Sprint Final Activities Report

Introduction

As always, we begin by reminding the readers that this is intended as an increment (pun intended) on our previous reports. The structure is what by now has become the standard, split between product and process (i.e., Sprint and Retrospective).

Sprint Review

Duration and Approach

Because of our previous gripes with final activities, and how rushed we felt at the end of the sprint, we decided to shift our approach slightly. The duration was in line with our last Review (40-45 minutes), and we still completed the product state activity, but the focus was on the team feedback, more than the product state itself.

Workflow State

We identified a growth in 6 alphas; this was a snowball effect from both the end of our project and the feedback we received from the Product Owner (although we had a bit of a rocky meeting, finding one of our major bugs). These two main factors resulted in a growth in Stakeholders, Opportunity, Software System (where we would like to make it clear performance is acceptable, but mainly for low volumes), Requirements, Way of Working, and most importantly Work.

Video

At the time of writing, we have yet to record the video for this sprint: the reason for this is that the changes in this sprint were so radical, from a UX point of view, that we're finding it difficult to condense it in an easily digestible video. At the same time, when this report will be read (also, at the time of writing), the website to test it on will be ready, so I'm sure the POs will have all the chances they want to test it. The website showcases our work in all fronts: the UX has been

completely redesigned, streams are automatically started in background, email notification was reworked (and moved to the server), coordinates are now included on the map, instead of a separate textbox, the website has both a dark and a light theme, you can filter the list to only show tweets that are relevant to you, and lastly, tweets are not lost by leaving the page (as long as cookies aren't cleared).

Team Feedback

During this review, we all agreed that sharing the team's feedback was in fact a better way to voice our opinions than most other "serious games" or activities. In my opinion, though, this was not always the case: although the dev team was very open about their doubts and trophies in this fourth review, this was not the case in some of the earlier ones. This is why I believe the Review and Retrospective process should evolve together with the team, as games and activities make it easier for the team to open up but stifle how direct they can be in their answers. Here are a couple of extracts that highlight how the team is satisfied about our result:
"We successfully built a fully functional and rich user interface with an overall complete user experience"
"We completed all the stories we agreed upon, and the final result is quite impressive, although with more time we could have done even better"
"We actually got a lot done! it was supposed to be a chill sprint focused on workflow and final reports, but A TON of code was added, which made us move the deadline. worth it though!"

Sprint Retrospective

Practice Patience

Once again, we completed the Practice Patience serious game, with an especially low amount of patience involved. Due to an overbooked schedule, during this game in particular only three members were available, but we made sure to get feedback from everyone else as well. There are three main takeaways that differ substantially from last sprint's Retrospective, and which made us move cards around.

The first one is about the definition of Done. At the end of the sprint, due to the different possible conditions our software was being run in (different browsers, development vs production, different machines) I realized I needed to make sure to have a coherent test setup for all members: to achieve this, I wrote a simple list of things that everyone needed to try before being allowed to say "I tested it and it works" (since this has been a problem in previous sprints). This simple setup worked quite well, as at least we could make sure all features were tested (even though we all know an automated test setup would have been the best option, we decided it was not feasible).

The second one involves the meeting with the PO, and just notes that, although the meeting went so-so, it allowed us to find a sneaky bug and got us some useful feedback from the PO (in particular, we enjoyed the hard-earned praise!).

The third one still represented an improvement, but the decision to move it to a yellow position was merely based on a majority rule: it is about the Sprint Backlog management. During this sprint, possibly because of the time of year (Christmas and all that) or maybe because of very few pauses in-between sprints, task management was... a step down from last sprint, to say the least. At the same time, 3 out of 5 members were absolutely on point, so to be fair it was still good enough to be in the yellow.

Team Feedback

Just like in the Review, some of the most interesting points were raised in the feedback section. Here are some of them.

"I had to work harder in this sprint because the UI restyling was more complicated than I expected and also the holydays and the exams didn't help with keeping me focused".

"Unfortunately, we all struggled to focus equally on the project due to holidays and exams. Still, we managed to achieve the goal and solve a lot of bugs in the meantime."

"Thanks to the process we used, we managed to best manage the project: in an easy way we shared the tasks and, giving constant updates to the rest of the team, we were able to know the progress of the project at all times". I

included this feedback because it was one that was completely unaffected by my own view of things. With it, I realized I should have withheld my own judgment until the end, as I think my role as coordinator influenced the other team members.

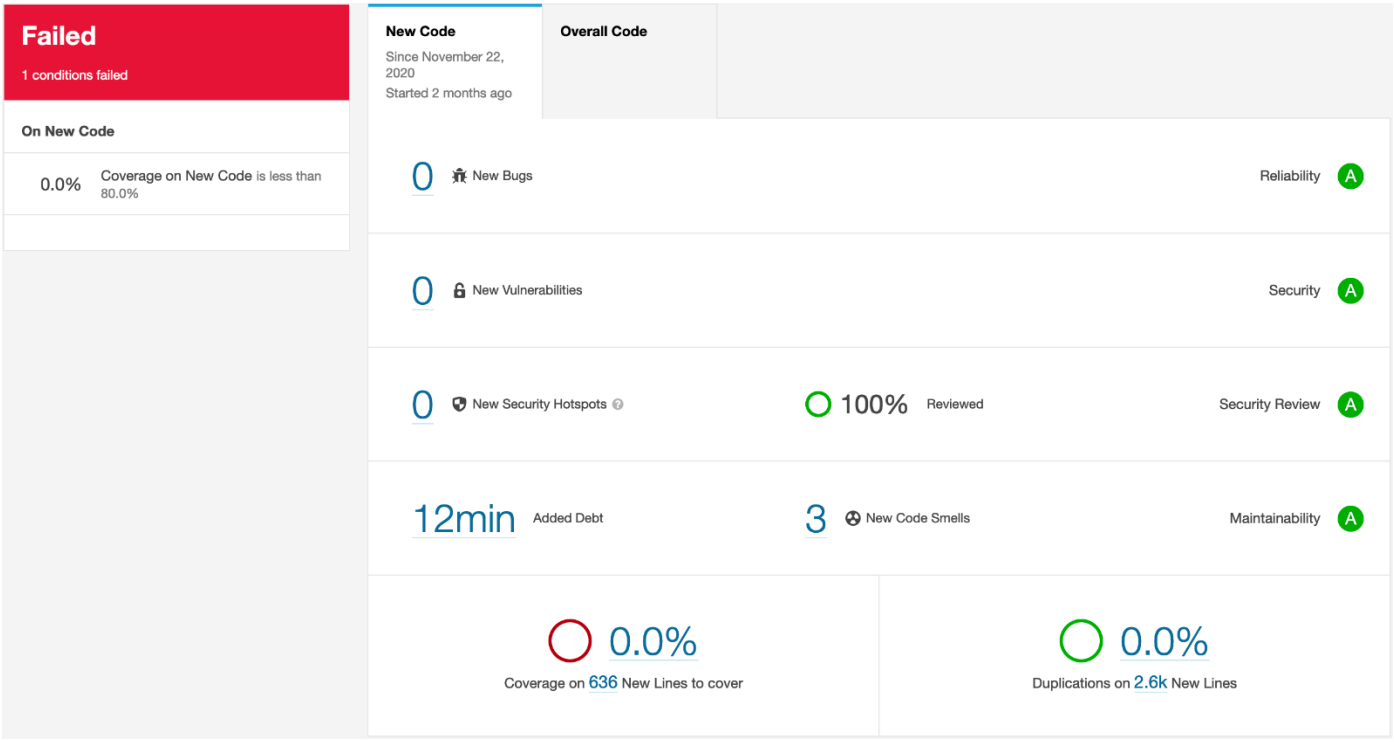
“This sprint was a bit of a letdown compared to the third. the dev team was a bit tired (understandably) and unfocused (exams + Christmas didn't exactly help), so it got a bit messy”. If it wasn’t clear, this was my feedback to the team. Still, the increment we produced was probably the largest one yet, so we valued PO satisfaction over the dev team’s small mistakes with process management. This means that, although it went OK for a single sprint, the fourth sprint’s setup would not have survived another sprint.

Conclusion

This report was the hardest one to write, as, for once, I did not find the dev team to improve compared to last sprint. At the same time, productivity was still very high, and the mishandling of the “process” was not serious enough to warrant jeopardizing the product’s completion. In the end, the increment we produced was quite impressive, and I can’t fault any of the team members for slowing down on the scrum side, especially considering all the coincidences I have outlined so far.

SonarQube Usage Report

Below we provide the main screen of SonarQube that shows a global view of the parameters related to the quality of the code. As for the other sprints the code coverage is 0% because we have not implemented any test.

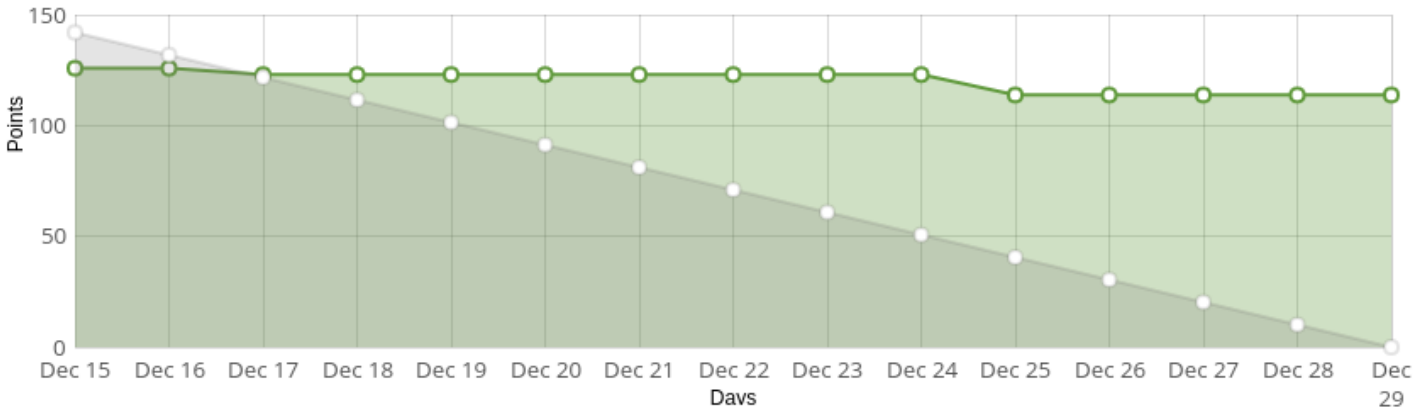


Unlike the previous sprints, we have decided not to solve all the code smells identified by SonarQube and that we report in the figure below.

Of these, the first is a false positive since the imported name is necessary for the proper functioning of a part of the code even if it is not used, while for the last two we decided that it did not make sense to change the name of the variables since they referred directly to the properties of the object passed to the function.

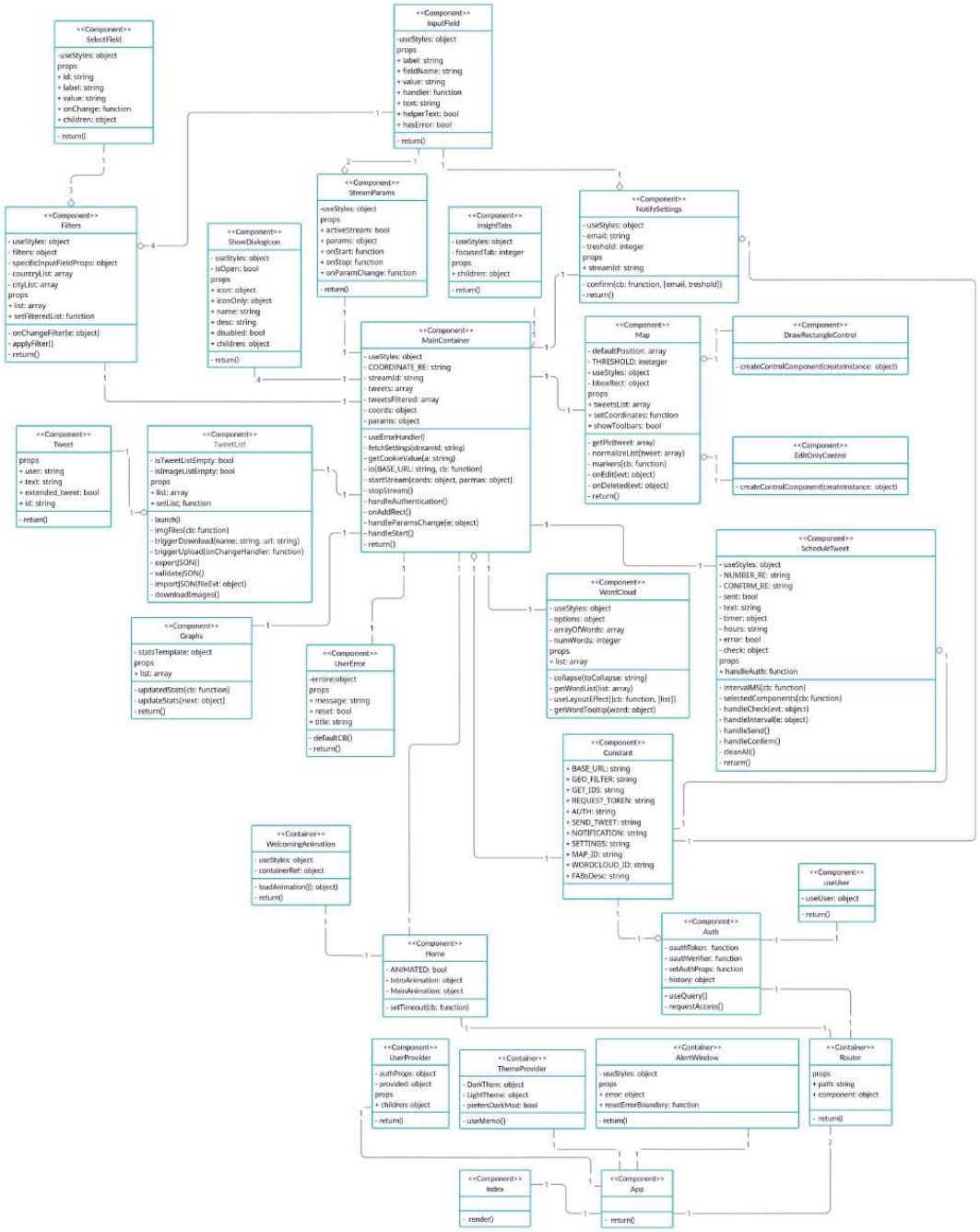
Sprint Burndown

Unfortunately, the burndown graph broke for the fourth sprint… Here is the result either way.



Additional Information

Class diagram



Conclusion

In this section, we will evaluate what the current state of the product is and what could be improved for future versions. We will also take some time to propose changes to the project structure as a whole: since the Instructors have mentioned that we are the first year they propose this type of examination to, we feel it is important to give constructive feedback to make this course even better next year.

Collected Tweets

To collect tweets, we used a raspberry pi running a python scripts using the Tweepy open library. We set as filter a coordinates box [6.48, 36.59, 18.50, 47.05] lng/lat around Italy and then applied a successive selection of tweets written in Italian.

Tweets were collected between the 3 of November and 17 of November and were stored in a TinyDB database on the raspberry and converted to Json using a custom script.

Overall, we collected 16513 geolocalized tweets and 2027 of those contained pictures.

Current Product State

We believe the product state is well represented by the fourth sprint review. In addition to that, we want to mention that we will continue hosting the source code available for all, but move it on GitHub. We are not likely to keep a running version online, due to security issues, needing developer credentials and the unavailability of a server to host it on. We will complete the README with screenshots and host videos and document inside the repo itself (the video will be a link to a better-suited website). We believe our software is fairly extensible, which means we will be open to positive changes from both the community and next year students. We will complete this with a few screenshots of the app.

Possible Future Improvements

Although we are proud of the software we produced, we were constrained by time, so we want to mention some current issues and possible improvements for future work:

- Efficiency: right now, twitter tracker is not efficient. Due to development needs, we opted for suboptimal choices (from the point of view of efficiency) that made the app much faster do develop and allowed for more radical changes. Due to our very short window for moving from development to production, we could not make some important changes that would make a sizable difference.
- Making images available while offline: in this version of the app, in order to attach images to tweets your client needs to be open. This is because we must render the components in order to convert them to an image. In a future version, it would be interesting to find a way to render specific parts of the UI on the server, so that images could be sent in background.
- Unifying capabilities of email and tweet: together with the previous change, another interesting addition would be letting email and tweet be two different mediums for the same subset of operations. This would require the previous proposal in order to work in the background.
- Reviewing the session workflow: right now, we only remember people with streams open. Although this is very privacy friendly, it makes for some non-intuitive workflows: if you don't read the documentation, I would bet it is complicated to understand the criteria for not losing your data.

Issues and Suggestions

In this final section we will summarize the main issues we encountered throughout the sprints and what we believe could be done in the future iterations of the project.

Tools and Methods

We know this was a direct consequence of how novel the course was, but we believe it is important to mention it: the tools the Instructors asked for changed multiple times at the start, and university-hosted versions were not available. Together with the requirement document, we were also prompted to consider using our own self-hosted versions and given a script to install them. Unfortunately, the script was outdated to say the least; this meant we had to dedicate a lot of time at the start of the first sprint to set up all the tools we needed. Then, a few days later, the promised university-hosted versions appeared, so we felt like we had thrown valuable sprint time away (not personal time, as we considered it an important learning experience).

The logger was another important source of conflict and confusion: at the start, we were not told about its necessity or relevance, but as the project went along the Professors made it clear that "productivity data" were an important part of the project as a whole. Because of unclear explanations, multiple students took issue with the request of installing a keylogger on their personal computer; because of this, we proposed to shift the focus from a generic keylogger to tools

that specifically tracked time. Moreover, the time tracking tools were not agreed upon at the start; this lead to confusion among teams and multiple ways to collect data, which I'm sure will make the analysis phase much more annoying.

Overlap of learning and applying

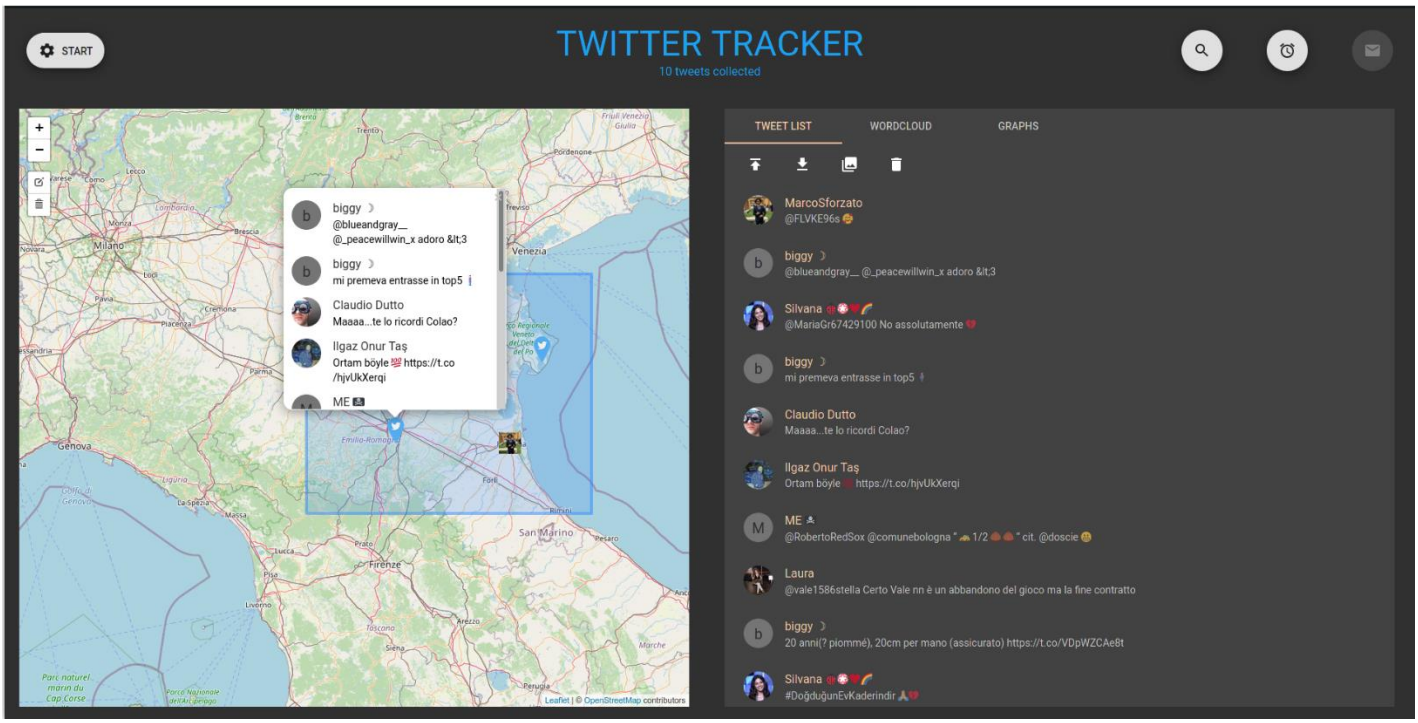
This is probably going to be a controversial statement, but I believe less overlap between the learning and applying periods of scrum especially would have been useful to us. Still, I think the Scrum Masters needed to know their future duties in advance, so we propose the following roadmap:

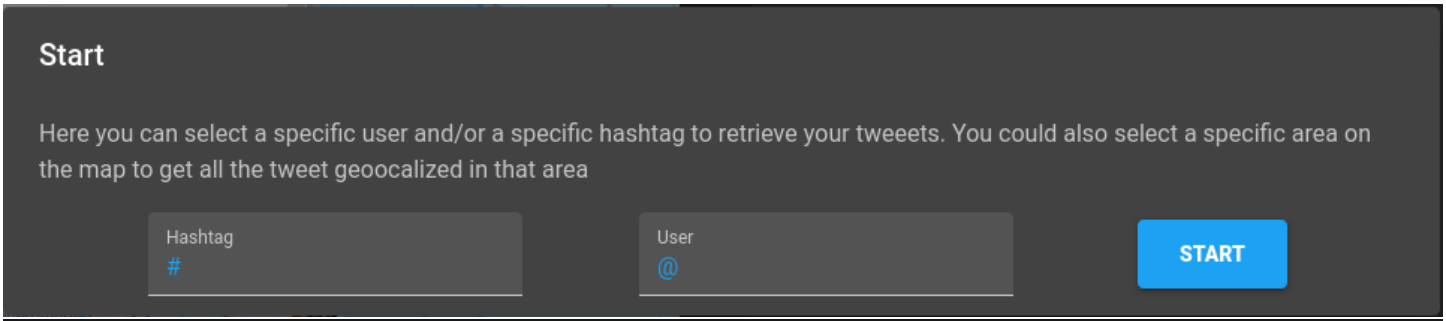
1. Course and project introduction: both the course and the project should probably be introduced in the first few lessons. The introduction should make it clear that it is not possible to start early, as work will be tracked throughout the sprints, but also that listening attentively to agile practices and techniques will help make both the coding and reviewing moments much easier. At the same time, we recognize the Instructors did explain agile workflows in more detail.
2. Group formation: we believe forming groups earlier would be beneficial to their ability of starting well and avoiding wasting time on simple tasks. For example, the scrumble game could be moved back to a week before the start of the first sprint.
3. Tool installation: this is only necessary if the students are expected to install their own versions of self-hosted software, as it is a time-consuming and difficult task that cannot be waved away by saying "aren't you guys computer scientists?"
4. Sprints begin.

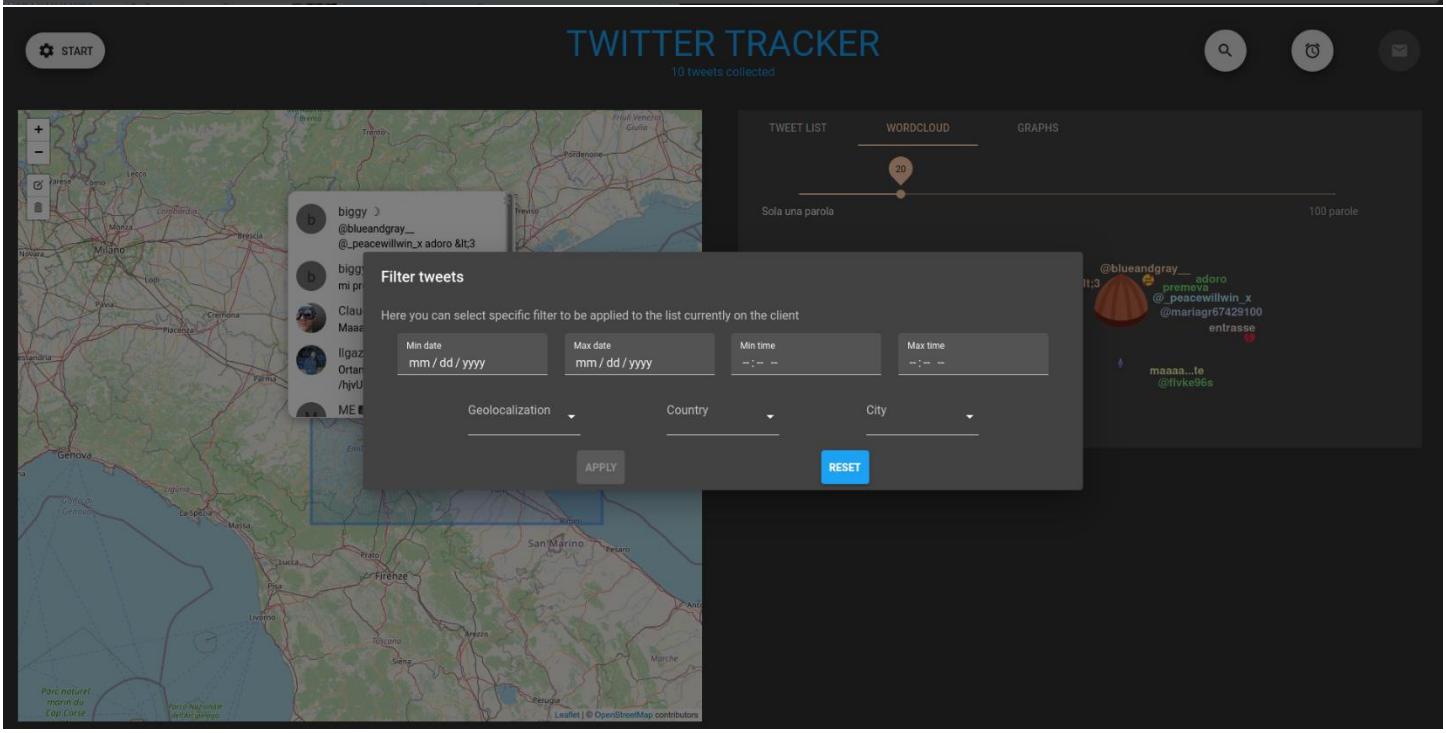
Direct scrum usage

Lastly, we want to mention that in the future it should be made clear that Scrum is only a steppingstone to a more suited and encompassing workflow: many of the requirements and practices of Scrum simply cannot be adapted to our structure. As an example, we bring daily scrums: in our context of distributed online working, they simply have no meaning whatsoever. They are inefficient, annoying to organize, and do not allow for any further scheduling, as everyone's time constraints make it impossible to synchronize. Moreover, Scrum expects a level of involvement from the Product Owner which is simply unmaintainable for a Professor with multiple teams.

As a final note towards next year's teams, we recommend making sure to distinguish final activities and sprint planning, as they quickly collapse into one giant less-than-useful activity where it is very hard to accomplish all that's needed without growing annoyed and losing all enthusiasm.


START
TWITTER TRACKER
10 tweets collected
TWEET LIST
WORDCLOUD
GRAPHS
MarcoSforzato
@FLVKE96s
biggy
@blueandgray__ @_peacewillwin_x adoro <3
Silvana
@MariaGr67429100 No assolutamente
biggy
mi premeva entrasse in top5
Claudio Dutto
Maaaa...te lo ricordi Colao?
Ilgaz Onur Taş
Ortam böyle https://t.co/hjvUkXerqi
ME
@RobertoRedSox @comunebologna " 1/2 " cit. @doscie
Laura
@vale1586stella Certo Vale nn è un abbandono del gioco ma la fine contratto
biggy
20 anni(? piommé), 20cm per mano (assicurato) https://t.co/VDpWZCAe8t
Silvana
#DoğduğunEvKaderindir
Leaflet | © OpenStreetMap contributors


Start
Here you can select a specific user and/or a specific hashtag to retrieve your tweeets. You could also select a specific area on the map to get all the tweet geoocalized in that area
Hashtag
#
User
@
START


START
TWITTER TRACKER
10 tweets collected
TWEET LIST
WORDCLOUD
GRAPHS
Sola una parola
100 parole
Filter tweets
Here you can select specific filter to be applied to the list currently on the client
Min date
mm / dd / yyyy
Max date
mm / dd / yyyy
Min time
Max time
Geolocalization
Country
City
APPLY
RESET
@blueandgray__
adoro
premeva
@_peacewillwin_x
@mariagr67429100
entrasse
maaaa...te
@flvke96s
Leaflet | © OpenStreetMap contributors