Projet TOP 2018
Groupe A2

Axel THOUVENIN Florian VOGT Pierre BOUILLON
Thomas BAGREL

Novembre 2018 - Janvier 2019

Table des matières

Introduction **5**

I Le sujet **6**

1 Exploration du sujet **7**
1.1 Cadre général . . . 7
1.2 Fichiers sources . . . 7
1.3 Contenu du sujet et interprétation . . . 7
1.3.1 Manipulations des données et statistiques . . . 7
1.3.2 Projection et affichage . . . 7

2 Etat de l'art **9**
2.1 Données d'aéroports . . . 9
2.2 Cartes et affichage . . . 9
2.2.1 Données géographiques . . . 9
2.2.2 Affichage . . . 9
2.3 Statistiques sur les aéroports . . . 10
2.4 Projections . . . 10
2.4.1 `EquiRectangularLat0Projector` . . . 10
2.4.2 `EquiRectangularProjector` . . . 10
2.4.3 Autres projections . . . 10

II Développement **11**

3 Structure du projet **12**
3.1 Introduction . . . 12
3.2 Packages et classes . . . 12
3.3 Spécifique à Scala . . . 13
3.3.1 Les *case classes* . . . 13
3.3.2 Les *lazy vals* . . . 13

4 Normalisation du code **14**
4.1 Introduction . . . 14
4.2 Style (pour le code) . . . 14
4.2.1 Choix du style . . . 14
4.2.2 Avantages apportés . . . 14
4.3 Complexité . . . 15

5 Technologies **16**

5.1 Parsing du CSV . . . 16
5.1.1 Choix de la librairie . . . 16
5.1.2 Avantages apportés . . . 16
5.2 Construction de la carte . . . 16

III Réalisation des tests 17

6 Méthodologie 18
6.1 Tests unitaires ou non ? . . . 18
6.2 Implémentation . . . 18
6.3 Méthodologie AAA . . . 18

7 Frameworks 20
7.1 Choix du framework . . . 20
7.2 FeatureSpec . . . 20
7.3 WordSpec . . . 20
7.4 FunSuite . . . 21
7.5 FlatSpec . . . 21

8 Structuration des tests 22

IV Gestion de projet 23

9 Choix de la méthodologie 24
9.1 Intégration des modifications . . . 24
9.2 Revues de code . . . 24
9.3 Cycles de développements rapides . . . 24

10 Réunions et jalons 25
10.1 Description des jalons . . . 25
10.2 Réunions . . . 25

11 Organisation du travail 26
11.1 Kanban . . . 26
11.1.1 *Issue board* . . . 26
11.1.2 Évolution incrémentale . . . 26
11.1.3 Branches . . . 26
11.2 Aperçu des sprints réalisés . . . 26

12 Difficultés rencontrées 28
12.1 Niveaux des membres du groupe . . . 28
12.2 Gestion du temps . . . 28

13 Communication interne 29
13.1 TELECOM Nancy . . . 29
13.2 Discord . . . 29
13.3 GitLab . . . 29

V Bilan et évolutions **30**

14 Bilan **31**

15 Évolutions possibles **32**

Annexes **34**

16 Compte-rendus de réunions **34**
16.1 Compte rendu de réunion n°0 34
16.2 Compte rendu de réunion n°1 35
16.3 Compte rendu de réunion n°2 36
16.4 Compte rendu de réunion n°3 37
16.5 Compte rendu de réunion n°4 38
16.6 Compte rendu de réunion n°5 38
16.7 Compte rendu post mortem 39

Sources **44**

Introduction

Ce document a pour but de décrire l'évolution du projet que nous avons mené ces 3 derniers mois. Ce document contient cinq parties :

— la première comporte une présentation générale du sujet, de l'interprétation que nous en avons faite, ainsi qu'un état de l'art analysant les technologies existantes traitant du même thème ;

— la deuxième décrit l'aspect technique de notre code, en particulier sa structure ainsi que les différentes technologies utilisées ;

— la troisième partie de ce rapport explore les techniques de test mises en place afin d'assurer le fonctionnement de notre code ;

— la quatrième partie vise à présenter la gestion de projet mise en place, son organisation concrète, les méthodes de travail que nous avons privilégiées (réunions, mesure de l'avancement...) ainsi que les moyens de communication utilisés ;

— enfin, la cinquième et dernière partie a pour but de conclure sur le travail réalisé ainsi que d'exposer les perspectives d'évolution de notre projet.

Commençons tout de suite avec la présentation du sujet.

Première partie

Le sujet

Chapitre 1

Exploration du sujet

1.1 Cadre général

Ce projet a pris place pendant notre première année de formation Ingénieur apprenti à TELECOM Nancy. C'est l'école qui a fixé le sujet, que nous devions réaliser à la fois sur la période "école" mais également en partie en dehors, entre novembre 2018 et janvier 2019.

1.2 Fichiers sources

Le sujet se base essentiellement sur un fichier `CSV` (*Comma Separated Value*) fourni par le site `openflights.org`, qui recueille les données d'identités et de localisation d'un grand nombre d'aéroports à travers le monde : `airport.dat`

1.3 Contenu du sujet et interprétation

1.3.1 Manipulations des données et statistiques

Il nous était proposé de manipuler ce fichier afin d'en extraire plusieurs informations significatives, et en particulier, de calculer les distances séparant les différents aéroports. Ainsi, plusieurs questions du sujet nous proposaient de calculer la distance minimale, maximale, moyenne ainsi que l'écart-type des distances entre les différents aéroports.

Un autre axe du sujet, qui s'est révélé être déterminant pour notre choix de structure de code, était la possibilité de filtrer les aéroports suivant plusieurs critères : leur distance à un point quelconque, l'hémisphère ou encore le pays dans lequel il sont situés...
En outre, il devait également être possible de calculer les statistiques évoquées ci-dessus sur un groupe d'aéroport restreint (filtré).

1.3.2 Projection et affichage

Les questions finales du sujet nous proposaient de projeter les aéroports sur une carte. Si ce volet ne laissait que peu de place à l'interprétation, le choix et l'étude des projections

demandèrent tout de même un travail important.

Enfin, en dehors de ces questions explicitement présentes dans le sujet, nous avons été amenés à réfléchir méthodiquement aux structures de données et de code à utiliser afin de produire un code concis, clair, sans répétitions, et surtout modulable.

Chapitre 2

Etat de l'art

Force est de constater que ce que nous avons réalisé est loin d'être novateur... Par exemple, le site `openflights.org` duquel nous avons extrait nos données propose dès sa page d'accueil une carte dynamique des aéroports et des vols, bien au-dessus de ce que nous vous proposons ici.

Nous allons tout de même explorer les technologies les plus connues et utilisées dans les domaines que nous avons explorés.

2.1 Données d'aéroports

Ici, `openflights.org` semble être le gagnant incontesté. En ce qui concerne les données sur les vols en direct (non couvert ici dans le sujet), on notera que Google propose également un service de visualisation d'un vol dès que l'on tape un numéro de vol dans sa barre de recherche.

2.2 Cartes et affichage

2.2.1 Données géographiques

Concernant les données géographiques, deux géants s'affrontent : Google avec Maps, et OpenStreetMap, offrant des données totalement libres de droits, avec une communauté active et en pleine expansion.

2.2.2 Affichage

Concernant l'affichage (dynamique) de ces données géographiques, Google joue ici un double rôle puisque Maps permet directement une visualisation dynamique de n'importe quel endroit du globe (mais encore une fois, avec une technologie propriétaire et fermée). Côté libre, cette mission est le plus souvent confiée à Leaflet (utilisée sur le site de OpenStreetMap), ou à d3 (que nous avons utilisé pour nos fonds de carte), deux librairies JavaScript très populaires.

2.3 Statistiques sur les aéroports

Le calcul de statistiques de distance sur un groupe spécifique d'aéroports semble être une problématique trop précise pour trouver un service clé en main sur Internet. En effet, si une rapide recherche Google nous permet de trouver une quantité incroyable de statistiques sur les aéroports pouvant intéresser les personnes prenant l'avion, nous n'avons pas pu trouver en ligne l'équivalent de ce que nous implémentons ici pour le projet (c'est-à-dire, sur les distances entre aéroports).

2.4 Projections

Nous avons brièvement étudié les projections existantes pour réaliser ce projet. Nous nous sommes surtout concentrés sur la projection équirectangulaire, comme demandé dans le sujet.

Seulement, il ne semblait pas exister sur Internet de définition précise de projection équirectangulaire centrée sur un point du globe. Nous avons donc décidé d'implémenter deux sous-types de projections équirectangulaires

2.4.1 `EquiRectangularLat0Projector`

Cette projection centre horizontalement (en longitude) le point choisi, mais n'affecte pas les latitudes. Le planisphère est donc "enroulé sur lui-même" horizontalement par rapport à la projection usuelle centrée sur l'Europe. Les parallèles et les méridiens sont parallèles sur la carte obtenue.

2.4.2 `EquiRectangularProjector`

Cette projection centre horizontalement (en longitude) mais également verticalement (en latitude) le point choisi. Tout se passe comme si on faisait d'abord tourner le globe pour centrer le point choisi en longitude, puis qu'on faisait tourner le globe pour centrer le point choisi en latitude. Le point choisi se retrouve donc véritablement au centre de la carte finale, mais les pays sont "déformées" par rapport à la représentation usuelle centrée sur l'Europe (sauf dans le cas où la latitude du centre est 0), et les parallèles et méridiens ne sont plus parallèles sur la carte obtenue.

2.4.3 Autres projections

Nous n'avons pas eu le temps d'implémenter d'autres types de projection, et donc nous n'avons pas regardé dans le détail les formules de projection associées. Seulement, notre code a été pensé dans le but de pouvoir implémenter *a posteriori* d'autres types de projections, y compris comme Mercator, où tous les points du globe n'apparaissent pas forcément sur la carte finale (avec notamment la classe abstraite `ProjectedPoint`, se déclinant en `OutOfMap` ou `OnMap(x: Double, y: Double)`).

Deuxième partie

Développement

Chapitre 3

Structure du projet

3.1 Introduction

Nous avons choisi, dès le début du projet, d'utiliser au maximum la puissance du langage imposé, Scala, en alliant programmation fonctionnelle (pour la concision, la clarté, et la justesse du code) avec une structure objet, au travers des *package*s, *module*s, *class*es, *trait*s et *object*s qu'offre ce langage.

Ainsi, dès le premier commit (`commit/6abc63fde8f6599fc6254ce4cfd0a93104ed4182`), la structure principale qu'allait revêtir notre code était déjà établie, permettant un repérage clair dans les tâches à réaliser ainsi qu'une compréhension facilitée de l'organisation et de l'utilisation de notre code.

3.2 Packages et classes

Héritant du Java, l'organisation du code en *package*s de Scala fut facile à prendre en main pour la plupart des membres du groupe. En revanche, la diversité des structures proposées par Scala (*abstract class*, *class*, *trait*, *object*...), souvent traduites indifféremment par des simples *class*es dans les langages objets courants, demanda plus d'efforts. Il fut décidé de les distinguer ainsi :

- *abstract class* : pour une classe "mère" décrivant un comportement générique, qui est ensuite spécialisé par les classes filles concrètes qui en héritent (exemple : `AirportFilter`) ;
- *trait* :
  - soit pour représenter une idée abstraite, une catégorie d'idées (`enum`, exemple : `HemisphereChoice`) ;
  - soit pour représenter une "facette", un fragment de comportement que peut posséder un objet (exemple : `HasCoordinate`, pour tous les objets possédant une localisation sur le globe au format longitude, latitude) ;
- *object* :
  - soit pour représenter une idée appartenant à une catégorie (membre d'une `enum`, exemple : `Northern` ou `Southern`) ;

  - soit pour regrouper les champs et méthodes statiques d'une classe (puisque contrairement au Java, Scala ne permet pas de placer de méthodes statiques dans ses classes) – dans ce cas l'objet est appelé *companion object* ;
  - soit, enfin, pour représenter tous les objets *singleton*s, c'est-à-dire le résultat de classes n'admettant qu'une seule instance (et donc instanciation) ;
- *class* : pour toutes les classes restantes (et concrètes).

3.3 Spécifique à Scala

3.3.1 Les *case class*es

On notera que les *case class* ont beaucoup été utilisées dans le code. En effet, ces dernières permettent un *pattern matching* facilité, et sont plutôt adaptées à représenter des structures de données immuables. Enfin, certaines de nos classes ne sont pas *case*, car les données passées en paramètres à leur constructeur nécessitent des petites modifications avant de devenir des attributs publics. Cependant, dans ce cas, nous avons implémenté les méthodes `apply` et `unapply` dans le *companion object* de la classe concernée, afin de bénéficier du *pattern matching* exactement comme pour une *case class* (voir diagramme de classe).

3.3.2 Les *lazy vals*

Nous avons changé une partie des méthodes de la classe `AirportDistanceMap` par des *lazy val*. Ces dernières permettent d'allier le calcul sur demande des méthodes avec la persistance du résultat offert par une *val*. Pour résumer le fonctionnement des *lazy vals*, ces dernières n'évaluent le calcul que lors de l'appel, et le stockent ensuite comme une *val*. Ainsi, le calcul de la distance moyenne n'est effectué qu'au plus une fois par instance de la classe `AirportDistanceMap`, et ce malgré son utilisation dans l'implémentation de la méthode du calcul de l'écart-type (`stdDev`).

Chapitre 4

Normalisation du code

4.1 Introduction

La robustesse du code de notre projet est principalement apportée par sa structuration et l'utilisation des atouts de Scala. Toutefois, nous avions tous conscience qu'un projet informatique est amené à être modulé et amélioré avec le temps (détection de bugs, ajout de fonctionnalités, etc.). Il nous a donc semblé crucial de pousser la rigueur avec laquelle développer jusque dans son style.

4.2 Style (pour le code)

4.2.1 Choix du style

Ne pouvant perdre trop de temps à définir une nomenclature pour la totalité du langage, nous nous sommes largement appuyés sur une des normes en vigueur, à savoir celle de Scala. Nous disposions alors d'un wiki déjà fourni et avec quelques guides ; notamment pour l'utilisation de git à ce moment. Notre choix a donc été d'ajouter une partie à la fin des tutoriels sur git dans le wiki concernant certaines bonnes pratiques à prendre pour le code du projet. Elles sont disponibles ici. Nous y spécifions par exemple qu'il faut définir au mieux tout ce qui peut l'être, de lever les ambiguïtés qui peuvent apparaître et de produire du code aussi lisible qu'efficace.

4.2.2 Avantages apportés

Ce style de développement a permis une harmonisation entre le code de chacun et il était plus facile à la fois de s'inspirer de parties développées dans d'autres fichiers (car similaires et donc plus claires et explicites) ; mais aussi de mieux comprendre le travail des autres, en ne perdant pas de temps sur la forme et en pouvant directement se focaliser sur le fond et les algorithmes développés. Pour s'assurer de l'homogénéité du projet et de son style, les différentes méthodes d'intégrations mises en place à travers GitLab nous permettaient à tous de progresser et d'ajuster le code rendu à chaque étape.

Ces efforts sur la rigueur de l'écriture du code ont été payants surtout lors du début et de la fin du projet. Au début pour la compréhension globale des fichiers, et à la fin lorsqu'il nous a fallu améliorer et altérer certaines parties. La modularité de notre travail nous a permis de gagner un temps considérable lorsque des modifications étaient nécessaires, permettant d'éviter l'introduction de bugs (car le code aurait été abscons), et permettant également d'éviter de déranger l'auteur du-dit code. Il en résulte un code propre, stable de par sa structure et fluide de par sa forme.

4.3 Complexité

La seule complexité notable de notre projet est la création de l'objet `AirportDistanceMap`, puisque, lors de l'exécution de l'algorithme, il s'agit de calculer des données relatives aux aéroports entre eux. De fait, il nous fait itérer deux fois sur chaque élément du tableau. Il en résulte une complexité en $O(n^2) qui, bien que peu élégante, est malheureusement nécessaire pour cette partie.$

Chapitre 5

Technologies

5.1 Parsing du CSV

5.1.1 Choix de la librairie

Afin de lire les fichiers d'aéroports et de pays (au format `.CSV`), nous avons choisi de faire appel à une librairie externe afin de respecter le principe *don't reinvent the wheel*, et surtout d'éviter d'introduire des bugs potentiels supplémentaires dans notre *codebase*.

Nous avons choisi la librairie grâce à awesome-scala, un *repo* GitHub présentant les meilleures et plus populaires librairies en Scala pour de nombreux usages.

Notre choix s'est donc porté sur scala-csv, sous licence Apache 2.0.

5.1.2 Avantages apportés

L'utilisation d'une librairie nous a permis de bénéficier d'un *parsing* du CSV à la fois plus performant et plus complet que l'implémentation maison que nous aurions pu produire. Par exemple, nous n'avons pas eu besoin de gérer les cas où les champs CSV contiennent eux-mêmes une virgule (et nécessitent donc un échappement avec des guillemets), ce qui aurait été fastidieux et peu intéressant à faire à la main dans le cadre de ce projet.

5.2 Construction de la carte

Pour construire la carte, nous avons utilisé la librairie java.awt, conseillée pour Scala, et disposant au final de méthodes pratiques pour travailler sur une image (`.PNG`).

C'est notamment à partir des composants de java.awt que nous avons développé nos classes `Marker`, pour ajouter un marqueur sur le fond de carte fourni par le `BackmapProvider`

Au final, l'écosystème Java/Scala aura été très utile pour trouver facilement des librairies performantes et documentées pour nous aider dans la réalisation du projet

Troisième partie

Réalisation des tests

Chapitre 6

Méthodologie

6.1 Tests unitaires ou non ?

Il existe de nombreuses façons de réaliser des tests unitaires, chacune étant plus ou moins rigoureuse et/ou fidèle au principe "unitaire" du test. Pour cette raison, il nous fallait avoir des tests cohérents à la fois dans ce qu'ils testent, mais aussi entre eux, que chaque test soit tout autant pertinent que les autres présents.

Pour cela, nous avons choisi de structurer la réalisation de nos tests, qui se sont faits selon certains critères

6.2 Implémentation

Pour chaque bout de code, le test en lui-même doit respecter certaines règles. Ainsi, il ne faut pas qu'il appelle de composantes externes, pour respecter le caractère unitaire du test. Par exemple, une méthode testée d'un objet ne doit pas en appeler une autre dans le même test.

Pour éviter de tomber dans les travers du test, nous avons tout de même procédé à quelques limitations : les méthodes de Scala ne doivent pas être testée (si Scala propose une méthode, rien ne sert de vérifier qu'elle fonctionne dans notre propre méthode) et il est inutile de tester les frameworks utilisés (ils sont supposés stables et déjà testés). Une autre contrainte implicite a été respectée, à savoir ne pas surcharger de tests le projet. Ils sont en effet nécessaire pour contrôler son évolution, mais trop en avoir aurait eu l'effet inverse en enlevant toute flexibilité et en rigidifiant les possibilités offertes par le code. Nous nous sommes donc "limités" à des tests pertinents assurant que le fonctionnement des méthodes est correct.

6.3 Méthodologie AAA

Plus verbeusement *Arrange Act Assert*, cette méthodologie se décompose comme son nom l'indique en trois phases. Initialement, on crée les composantes à tester et les valeurs à leur associer (*Arrange*) ; on effectue ensuite l'action qui va être testée, en récupérant son retour s'il existe (*Act*) ; finalement, on vérifie le résultat en le comparant à un résultat attendu (*Assert*).

De cette manière, chaque test est extrêmement lisible et concis : on sait ce qui est testé, comment, et quel retour est attendu.

Chapitre 7

Frameworks

7.1 Choix du framework

Le choix du framework de test est extrêmement important et a nécessite une véritable réflexion. En effet, peu importe les efforts déployés dans la structuration, si les tests de part leur nature n'étaient pas construit autour de la même optique, nous allions alors avoir des dissonances et perdre en efficacité.

Ainsi, différents frameworks ont été explorés et plus ou moins contestés.

7.2 FeatureSpec

FeatureSpec nous a semblé intéressant grâce à sa syntaxe, proche de celle du SQL avec un principe de mots clefs clairs et un détail de chaque scénario. Cependant ce choix a vite été abandonné car devant la redondance de certains tests et la différence entre la taille des tests (de quelques lignes à plusieurs blocs) il nous a semblé que nous allions très rapidement nous retrouver avec des tests, bien que bien structurés, totalement illisibles et difficilement maintenables.

7.3 WordSpec

WordSpec avait l'avantage de la concision et de rester verbeux même si réduit. On retrouvait ce principe de mots clefs pour garder un aspect très lisible et compréhensible, mais la succession de ces derniers impliquait une imbrication selon les cas pouvant devenir très importante trop rapidement.

N'ayant pas encore idée de la structure de tous nos tests ou de leurs complexités dans leur rédaction, nous avons préféré explorer d'autres pistes.

7.4 FunSuite

FunSuite est très simple d'utilisation et très similaire à de nombreux autres frameworks de tests. Nous avons abandonné cette initiative au profit d'une solution, au risque d'être plus complexe, nous dotant d'outils plus puissants ; voire une meilleure structuration dans la synthaxe de nos tests.

7.5 FlatSpec

FlatSpec a un aspect très fluide dans son écriture : les mots clefs et la "documentation" s'enchevêtre très bien et nous permette d'écrire des tests tant élégants, que maintenable, que lisibles et efficaces.

Chapitre 8

Structuration des tests

Le rôle particulier des tests dans un projet tends à faire défaut à la qualité avec laquelle ils sont produits. On voit souvent des tests peu utiles, pertinents, voire parfois même, leur absence totale.

C'est pour éviter ces travers courant que nous avons préféré dès le début maintenir la même rigueur et exigence pour les tests que pour le code.

Afin de rester clair et explicites, nous avons choisi que pour chaque fichier de code, correspondra un fichier de tests. Cette structure quasi dupliquée des fichiers permet du premier coup d'oeil de saisir à quel fichier correspond quel test. Pour renforcer cette rigidité, nous avons associé un objet de test par méthode testée. Une méthode d'un fichier avait donc, pour tests, un fichier correspondant dans lequel se trouvait un objet Scala, testant les différents comportement de notre code.

Pourtant, nos tests utilisant parfois des ressources communes, nous souhaitions continuer à préserver ce principe de *DRY* (Don't Repeat Yourself). C'est pourquoi la structuration des fichiers tests diffèrent en quelques points, dont celui de fichiers ressources utiles spécialement aux tests se trouvant dans le répertoire consacré à ces derniers.

Quatrième partie

Gestion de projet

Chapitre 9

Choix de la méthodologie

Nous avons opté pour un processus agile en combinant les méthodes agiles SCRUM et Kanban (Sprint Backlog) tout en s'inspirant du cadre agile "eXtreme Programming" qui nous a permis de notamment renforcer plusieurs points importants.

9.1 Intégration des modifications

Les "branches" de GitLab nous ont permises de séparer proprement les tâches à effectuer. En effet, lorsque nous voulions travailler sur un point précis, la première chose que nous faisions était de créer une branche pour celui-ci. Ensuite, nous réalisions la tâche (soit seul soit à plusieurs) puis lorsque l'équipe attribuée considérait que le développement était fini, elle le signalait aux autres membres du projet en créant une *Merge Request*, et ces derniers procédaient alors à des revues de code.

9.2 Revues de code

Représentées par les *Merge Requests* (MR) de GitLab, à la fin du développement de la tâche par l'équipe désignée, celles-ci proposaient une fusion avec le projet principal où les autres membres du groupe devaient analyser les changements effectués par cette branche, puis les valider ou refuser en justifiant par le biais de commentaires la modification. Ce fonctionnement nous a permis de chacun voir ce que les autres avaient réalisé.

9.3 Cycles de développements rapides

Nos sprints ont tous duré deux semaines, ce qui nous a permis de bien développer les fonctionnalités cruciales du projet assez rapidement. En plus des stand-ups meetings, en milieu de sprint (chaque semaine), nous réalisions des réunions d'avancement, afin de mettre en avant les problèmes rencontrés et éventuellement les corriger afin d'éviter tout blocage éventuel.

Chapitre 10

Réunions et jalons

10.1 Description des jalons

Au début du projet, nous nous sommes réunis afin de définir des catégories de jalons :

- *Open* : Élément non traité et qui ne le sera pas dans le sprint en cours ;
- *To implement* : Élément à implémenter dans le sprint en cours ;
- *Implementing* : Élément en cours d'implémentation ;
- *To Test* : [Dans le cas d'un algorithme] Élément développé nécessitant des tests unitaires ;
- *Testing* : Élément pour lequel les tests unitaires sont en cours de rédaction ;
- *To Review* : Élément à vérifier (développement et tests réalisés (dans le cas d'un algorithme), bien rédigés (dans le cas d'un document) . . .) ;
- *Reviewing* : Élément en cours de vérification ;
- *Closed* : Élément terminé : implémenté, testé et/ou vérifié.

Ils étaient donc tous référencés parmi celles-ci sur l'*issue board* de GitLab (qui nous a servi de *Sprint Backlog /* tableau Kanban). Celui-ci était mis à jour constamment et était utilisé lors de chaque réunion en tant qu'appuis aux discussions.

10.2 Réunions

Tous les lundis, nous réalisions une réunion de type "avancement", afin de faire le point sur les tâches qui avaient déjà été réalisées en mi-sprint ou pour clôturer le sprint en cours et ainsi commencer le suivant.

Celles-ci nous servaient à communiquer sur les points importants qui nécessitaient une discussion plus longue (rapports sur l'avancée, prises de décisions . . .), en plus des stand-ups meetings quotidiens qui servaient eux à traiter les petits problèmes (question(s) sur la partie d'un autre membre, demande de *code review* . . .) .

De plus, il nous arrivait de nous réunir physiquement afin de porter une réflexion commune sur un sujet plus important que le développement d'une fonction (principalement un changement de structure ou un ajout important, ex : les projections) et qui ne pouvait attendre une réunion planifiée.

Chapitre 11

Organisation du travail

11.1 Kanban

11.1.1 *Issue board*

Puisque nous avons choisi d'adopter une approche agile, nous avons donc planifié nos tâches pendant nos réunions d'amorçage de sprint (catégorie *To implement*), puis nous les avons réparties entre nous par compétence et affinité, sur le tableau Kanban (*Sprint Backlog*) que propose GitLab (qui est normalement le "tableau de problèmes" (*issue board*) pour les communautés de plus grande ampleur).

11.1.2 Évolution incrémentale

Afin d'assurer un suivi du tableau au fur et à mesure de l'avancement des tâches, celles-ci étaient déplacées dans la catégorie correspondante à leur statut actuel de façon incrémentale : lorsque le développement était fini, elle se devait d'être testée (dans la limite du possible) puis ensuite être revue pour affirmer que son intégration dans le code resterait une évolution (sans *continuous integration*).

11.1.3 Branches

Dès la première réunion, nous avons débattu afin d'adopter un mode de fonctionnement pour la mise en commun du travail en utilisant GitLab : nous avons retenu un fonctionnement par branches, avec comme contrainte d'ajouter une fonctionnalité supplémentaire ou d'améliorer le projet en général, puis, lorsque celle-ci était fusionnée avec la branche mère (`master`), la branche était automatiquement supprimée avec l'option disponible lors de la création des *Merge Requests*.

11.2 Aperçu des sprints réalisés

Voici un schéma récapitulatif de nos sprints :

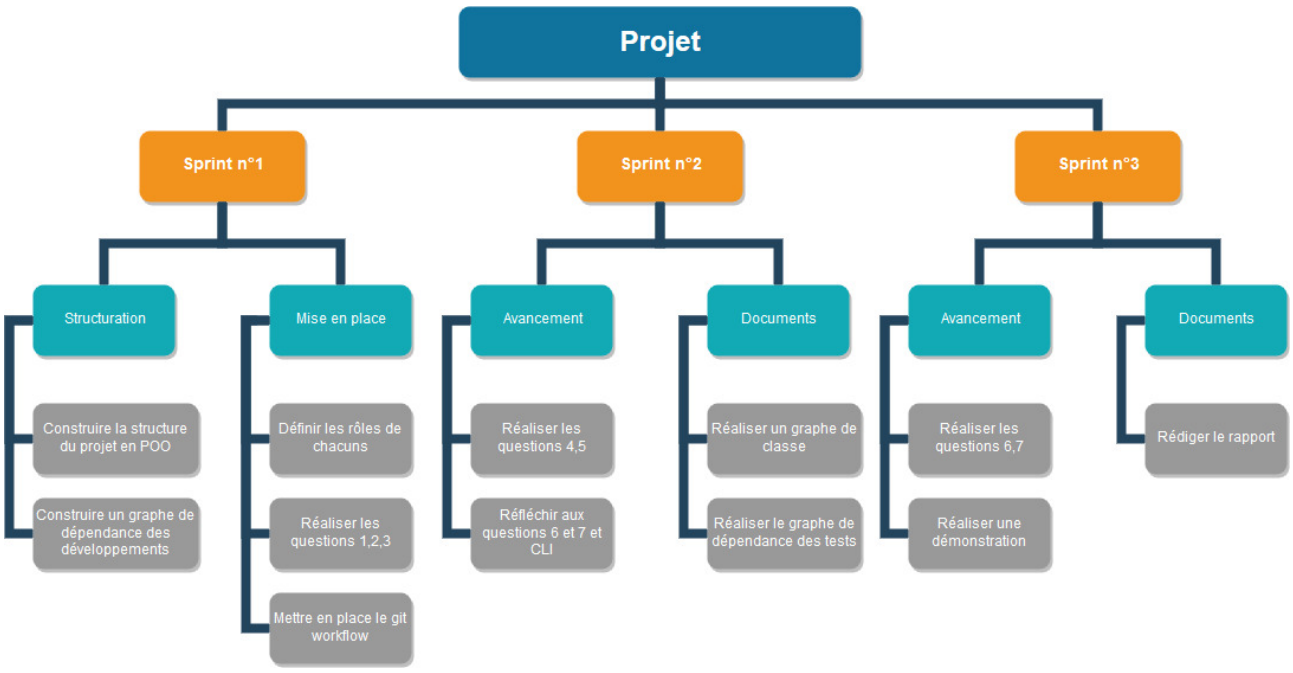

FIGURE 11.1 – Répartition des sprints

Toutefois, les tâches ne se limitaient pas au développement, il nous a fallu rédiger quelques documents annexes afin de mener à bien ce projet, tel qu'un diagramme de classe dans le but de se retrouver parmi nos objets et nos méthodes.

Chapitre 12

Difficultés rencontrées

12.1 Niveaux des membres du groupe

La première difficulté qui s'est révélée au début du projet est le niveau non homogène des membres du groupe, nous avons donc choisi de résoudre ce problème lors de la répartition des tâches : les fonctions simples se sont donc plus réparties sur les personnes plus faibles de l'équipe tandis que les points plus complexes se sont vus réalisés à plusieurs (*pair programming*) afin que chaque membre du groupe puisse progresser dans ces domaines.

12.2 Gestion du temps

Nous avons également eu des difficultés à maintenir un rythme constant, car certains points indispensables du projet nous ont pris plus de ressources que prévu,

De plus, le problème du temps s'est accru lorsque nous avons essayé d'implémenter des fonctionnalités qui se sont avérées difficiles à implémenter sans être primordiale, en particulier la CI/CD (*Continous Integration / Continuous Deployment*) sur GitLab.

Chapitre 13

Communication interne

L'une des notions les plus importantes à la réalisation d'un projet est la communication. Nous avons choisi de ne pas négliger cela et avons mis en place plusieurs moyens de dialogue :

13.1 TELECOM Nancy

Notre moyen d'échange le plus utilisé a été l'enseigne même de l'école TELECOM Nancy. En plus de nos réunions régulières, nous utilisions notre temps libre afin de parler du projet et de nos différents problèmes rencontrés durant nos sprints.

13.2 Discord

Nous avons beaucoup utilisé Discord, un logiciel gratuit optimisé pour la communication rapide. Nous avons créé des salons textuels portant des noms spécifiques permettant de garder une certaine rigueur et d'optimiser notre partage d'informations. Grâce à cela nous étions capables contextualiser la demande très rapidement. Nous avons également mis en place des salons vocaux principalement consacrés au *pair programming*.

13.3 GitLab

GitLab nous a permis de suivre le projet dans son ensemble et de lire le code en son intégralité à travers les différentes *Merge Request*s. GitLab nous a permis également de laisser une trace de la répartition des tâches dans le groupe. ;

Cinquième partie

Bilan et évolutions

Chapitre 14

Bilan

Arrivé à la date butoir du rendu de projet, nous avons un code fonctionnel, performant et testé. De plus, il couvre l'ensemble des problématiques du sujet, sans aucun comportement anormal ou défaut de fonctionnement.

L'utilisation de GitLab et du *VCS* git s'est également très bien déroulée, avec un *git history* propre, cohérent, illustrant bien l'implication et les évolutions du projets tout au long de son développement.

Pour finir, la réalisation ponctuelle mais efficace de réunions d'avancements et de stand-up meeting nous a maintenu sur la bonne voie en renforçant à la fois la cohésion mais aussi la visibilité globale du projet et de ses difficultés par tous les membres du groupe.

Il en résulte un livrable fini, de qualité ainsi que des documents l'accompagnant fournis. A défaut peut-être d'une gestion du temps parfois un peu trop approximative, l'expérience tirée par les membres du groupe est à la hauteur de la qualité du projet ainsi réalisé.

Chapitre 15

Évolutions possibles

Durant la réalisation du projet nous nous sommes rendu compte que certains points pouvaient être améliorés :

— Ne plus faire appel a un fichier sous forme csv pour le chargement des données, mais faire directement appel a une Api ce qui permet une mise à jour des données. Exemple, si un pays change de nombres d'aéroports cela changera aussi dans l'API.

— Avoir une vue satellite de notre carte pour une meilleure visualisation. Nous avions essayé mais la taille de l'image était trop importante.

— Approfondir encore plus loin les projections proposées par notre programme.

— Remplacer notre CLI par une API. Donc de ne plus avoir à passer par des commandes mais par une interface graphique routée par un serveur web.

Annexes

Chapitre 16

Compte-rendus de réunions

16.1 Compte rendu de réunion n°0

Membres présents :

— Bagrel Thomas
— Bouillon Pierre
— Thouvenin Axel
— Vogt Florian

Sujet : *Structuration et mise en place du projet*

Ordres du jour :

— Critique et retour sur le WBS
— Point sur le git workflow
— Réflexion sur les tests et la structure du code

Général

WBS

— Raffinement de la répartition des tickets
— Satisfaction globale du groupe : clair, efficace et synthétique

Workflow

— Utilisation du branching workflow (une fonctionnalité = une branche séparée)
— Présence d'un wiki, le centraliser sur le GitLab
— Quelques zones d'ombre sur l'utilisation du workflow ont été redétaillées
— Mise en place de pair-programming pour accompagner les membres dans l'utilisation du workflow

Tests et structure

- Réflexion sur un framework de test pour effectuer des tests unitaires
- Déconstruction des tests sous la forme AAA* évoquée
- Difficulté pour rédiger les tests unitaires à cause de parties interdépendantes pas encore implémentée => Construction d'un graphe de dépendances pour prioriser les implémentations
- Workflow type à terme : implémentation -> documentation -> test -> review
- AAA testing

16.2 Compte rendu de réunion n°1

Membres présents :

- Bagrel Thomas
- Bouillon Pierre
- Thouvenin Axel
- Vogt Florian

Sujet : *Avancement*

Ordres du jour :

- Priorisation des tâches
- Mise en commun de la progression de chacun

Général

Priorisation des tâches

- Graphe de dépendance des tests finis : organiser la rédaction des tests autour dudit graph
- Implémenter les tests tels que la personne les ayant fait ne les codes pas mais review la PR les concernant
- Au choix du développeur parmi le label `to test`

Mise en commun de la progression de chacun

- Bagrel Thomas
  - Fin des issues à développer
  - Début du testing dès l'ajout d'un template de test

    - Implémentation de la distance entre les aéroports
- Bouillon Pierre
    - Fin des tâches sur les Airports
    - Modifier l'unité du `.csv` des `countries` et implémenter sa lecture
    - Ajout de ressources dans le wiki
    - Voir pour l'implémentation GitLab de l'intégration continue
- Thouvenin Axel
    - Pair-programming
    - Prise en main du projet, prise de conscience des issues
- Vogt Florian
    - Issues avancées mais pas encore push (sera fait dans la soirée)
    - Graphe de classes à faire
    - Diagramme de classes à faire

16.3 Compte rendu de réunion n°2

Membres présents :

- Bagrel Thomas
- Bouillon Pierre
- Thouvenin Axel
- Vogt Florian

Sujet : Revue du sprint n°1

Points positifs

- Gitlow, système de board et de marge requests très fluides et appréciés ;
- Code et architecture propres, agréable à review et explicite ;
- Diagramme UML qui permet de bien appréhender le projet dans son ensemble.

Points négatifs

- Tracking des issues à tester peut-être pas assez rigoureux ;
- Attention à garder une avance vis-à-vis de la deadline.

Améliorations suggérées

- Possibilité de générer un manuel technique du logiciel grace à la ScalaDoc ;
- Prévenir dans les channels de communication intra-groupe pour prévenir des MR.

Tâches reportées

— Continuer à tester les différentes classes et atteindre une plus grande couverture de code testé.

Autre

— Evocation d'une évolution du workflow en branches avec différents niveaux de maturité (non suivie à cause de l'impact en terme de temps et de complexité que ça impliquerai)

16.4 Compte rendu de réunion n°3

Membres présents :

— Bagrel Thomas
— Bouillon Pierre
— Thouvenin Axel
— Vogt Florian

Sujet : Amorçage du sprint n°2

Ordre du jour :

— Planification des tâches

Général

Planification

— Répartition des différentes tâches dans le board ;
— Implémentation des questions 6/ et 7/ ;
— Continuation des tests ;
— Evocation d'une interface (?) ;
— Tâches annexes :
  — Tests intégration en corrélation avec le Readme.md ;
  — Etudier la mise en place de la CI ;
  — Implémenter une méthode qui appellerait une API pour les `Countries` ;
  — Nourrir le wiki ;
  — Etudier l'utilisation de lazyvals ;
  — Refactoring de certains noms d'attributs.

16.5 Compte rendu de réunion n°4

Membres présents :

— Bagrel Thomas
— Bouillon Pierre
— Thouvenin Axel
— Vogt Florian

Sujet : *Avancement*

Ordres du jour :

— Remarques et ressentis sur l'avancement du sprint

Général

Remarques négatives

— De gros ralentissement liés à un problème coté GitLab qui tuais la fluidité du workflow à cause de l'impossibilité de merge les branches (résolu via console) ;
— Petit inquiétude quant aux délais, l'implémentation des projections des données sur une carte prends plus de temps que prévu.

Remarques positives

— La testabilité et la couverture de test sont très satisfaisantes ;
— Le code est très lisible et écrit avec une grande consistance ;
— La communication autour du GitLab et des merge requests s'est beaucoup améliorées.

16.6 Compte rendu de réunion n°5

Membres présents :

— Bagrel Thomas
— Bouillon Pierre
— Thouvenin Axel
— Vogt Florian

Sujet : Revue de sprint n°2

Points positifs

— Avancement correct et satisfaisant du sprint ;
— Rapidité et fluidité d'execution du code y compris API ;
— Projet dans sa finalisation.

Points négatifs

— Avancement parfois disparate.

Améliorations suggérées

— Raffinement des salons textuels de notre plateforme de communication (Discord) ;
— Propositions pour la démo.

Tâches reportées

— Finalisation de la démo ;
— Complétude des tests.

Autre

— Abandon définitif de la mise en place d'une CI, trop chronophage pour l'apport que cela aurait.

16.7 Compte rendu post mortem

Objectifs initiaux au projet :

Voir le sujet donné.En résumé, une application permettant diverses opérations sur une base de données d'aéroports ainsi que la création de projections de ces derniers.Accompagner le projet de différents documents et ressources sur la gestion de projet et son déroulement.

Résultat final du projet

Toutes les parties et opérations demandées ont pu être testées. Une interface pour utiliser l'application a été également mise en oeuvre. Les documents de gestion de projet sont présents sous forme de wiki et de ressources dans le GitLab.

Analyse et identification

Réflexions des membres du groupe sur les principaux points du projet :

Satisfaction du livrable produit

— Bagrel Thomas : Très satisfait de la partie classes, un peu déçu de n'avoir pu proposer une meilleure interface par manque de temps. Très content de l'extensibilité du projet (facile d'ajouter de nouvelles projections, de changer le style de la carte...)

— Bouillon Pierre : Extrêmement content de la qualité du code et de ses performances. Le code est très stable, n'est pas aux limites de ses capacités au sens où sa durée d'exécution est encore très correcte pour le volume de données chargé. Il est aussi robuste : bien structuré, documenté et en grande partie testé. Pour finir, nous n'avons aucun bug ou comportement anormal/pénalisant, ce qui est également un facteur de réussite selon moi.

— Thouvenin Axel : Très satisfait du résultat final, honoré d'avoir participé à ce projet. Un peu déçu de ne pas avoir eu le temps de réaliser toutes nos idées.

— Vogt Florian : Très satisfait dans l'ensemble, je suis ravis de l'homogénéité de notre code et de la discussion qu'il y a eu entre les membres du groupe.

Quels ont été les points négatifs, comment ne pas les reproduire ?

— Bagrel Thomas : Le manque de temps s'est surtout fait sentir sur la fin du projet. Nous aurions dû démarrer le projet encore plus vite dès le début, et commencer le rapport bien plus tôt / au fur et à mesure de l'avancée du code. Enfin, nous aurions pu essayer d'avoir des périodes de travail plus homogènes.

— Bouillon Pierre : Nous avons cruellement manqué de temps dans la finalisation du projet. Cela vient pour moi de deux facteurs. Premièrement, la semaine de partiel nous a paralysé pendant deux semaines au total (une de révision, une d'examen), avec en plus la période entreprise rendant difficile le travail et sur le projet, et sur nos missions professionnelles. Deuxièmement, le manque de prise en compte de la charge de travail représentée par les documents de gestion de projet a grandement impacté la fin du livrable. Comme nous devions accorder plus de temps nous nous retrouvions en retard sur le code, et comme nous ne pouvions pas passer la partie code sous silence nous avons également pris du retard sur la partie gestion de projet. Je pense qu'une meilleure pondération des tâches et une prise en considération plus importante de la partie gestion de projet aurait pu éviter ces problèmes.

— Thouvenin Axel : Le manque de temps, ainsi que mon manque de technique, je n'ai pas réussi à produire le travail que j'aurai aimé fournir. Mon manque de connaissances a impacté directement la bonne réalisation du projet. J'aurai du réagir plus vite et ne pas avoir honte de demander de l'aide à mon équipe dès le début du projet.

— Vogt Florian : Le manque de connaissances dans le langage Scala s'est fait ressentir au début du projet, puis lorsque j'ai acquis assez de connaissances en celui-ci, le manque de temps s'est fait lui ressentir à son tour.

Quels ont été les points positifs, comment reproduire les bons résultats ?

— Bagrel Thomas : La rigueur au niveau de la propreté du code a été très profitable tout au long du projet. La présence abondante de commentaires est également une réussite pour notre code. Pour un premier projet avec une gestion de projet clairement établie, je trouve que nous avons réussi à garder un bon équilibre entre production et management. Enfin, nous avons réussi à avancer dans un climat très agréable notamment grâce à l'absence de hiérarchie au sein du groupe

— Bouillon Pierre : L'élégance et la robustesse du code est un énorme point positif. Les structures, les tests, la documentation, tout est relativement bien proportionné, pertinent dans leurs utilisations. Travailler avec une base de travail saine était très stimulant et a permi, à n'en pas douter, de renforcer l'investissement dans le projet de certains. Une autre source de satisfaction est l'utilisation de GitLab avec un workflow qui s'est révélé très fluide et efficace, nous avons pu garder une structure équilibrée du git et ne jamais avoir rencontré de problèmes critiques avec le gestionnaire de version. Enfin, la communication dans le groupe était très aisée et a pu accélérer bon nombre de prises de décisions.

— Thouvenin Axel : Nous avons fait preuve de professionnalisme, le code est documenté, propre et bon. La communication a été au centre de notre production ce qui apporte un vrai plus à la réalisation d'un projet. Je n'ai pas été mis à l'écart malgré mon manque de niveau bien au contraire toute l'équipe m'a soutenu et m'a aidé dans la réalisation de mon code et de mes différentes tâches au sein du groupe.

— Vogt Florian : Le fait de fonctionner en Agile a pour moi été un bon point, puisqu'au début nous n'avions pas énormément de connaissances vis-à-vis du projet, et celle-ci s'est affinée par la suite ce qui a permis de créer plus d'idées lors des derniers sprints.

Temps prévu v/s temps passé

— Bagrel Thomas : Nous avons beaucoup avancé les premières semaines, et nous avons donc eu le sentiment que la fin du travail était proche. Cependant, le nombreuses améliorations que nous avons mises en place dans le dernier sprint ont demandé un temps considérable, et nous avons donc eu du mal de finir à temps. Je pensais au démarrage du projet que la partie code prendrait beaucoup moins de temps (mais c'était sans compter sur le serveur pour les fonds de carte ainsi que les nombreuses autres améliorations que nous avons réalisé).

— Bouillon Pierre : Nous nous attendions à un projet peu important dans le temps qu'il allait exiger pour le programmer, mais avions beaucoup trop négligé et les aléas pouvant intervenirs, et les blocages possibles sur une tâche pour un autre membre, et la partie gestion de projet, et enfin l'impact de notre emploi du temps scolaire et professionnel (partiel et missions d'entreprise). Pour cela, nous avons fini par manquer de temps pour tout produire, même si, pour la partie développement, nous avons plutôt bien respecté nos estimations et délais.

— Thouvenin Axel : Les premiers sprints ont été très concluants, nous avons donc poussé les améliorations au maximum ce qui, au final, a augmenté la masse de travail à fournir et qui, par le fait, nous a ralenti.

— Vogt Florian : Nous avions débuté le projet en réalisant les tâches prévues par notre planning, puis des améliorations n'ont fait que surgire ce qui nous a considérablement ralenti. Je pense donc que nous avons sous-estimé la partie développement du projet qui nous a pris beaucoup plus de temps que prévu.

Plan d'action futur

Modification des calculs estimés

Lors de la répartition des tâches et de l'évaluation de ces dernières, nous avions largement anticipé et integré les tâches de développement. En revanche, nous avons trop passé sous silence le rapport et les documents de gestion de projet à fournir et produire. En conséquence, nous avons été retardé tant dans ces documents que dans les tâches. Il faudra donc, à l'avenir, pondérer aussi bien les tâches de développement que les tâches plus "administratives".

Modification du processus projet

Pour gagner en rigueur, en temps et en efficacité, nous pourrions mettre en place des périodes de sprint plus courtes, elles-même régulées par une CI (continuous integration). Cela permettrai de s'assurer de l'évolution de notre projet, de maintenir son exactitude mais d'en précipiter un peu le rythme.

Apprentissages et formations

— Bagrel Thomas : Durant ce projet, j'ai appris pour la première fois à travailler véritablement en groupe, et ce dans un climat très agréable. J'ai également pu mesurer l'importance de la gestion de projet, qui paraît très superflue au début du projet, mais qui s'avère être une ressource de taille au fil des semaines. Enfin, j'ai pu m'améliorer dans le développement en Scala et en particulier en POO.

— Bouillon Pierre : Grâce à ce projet, j'ai pu éprouver mes connaissances en git et surtout les parfaire : aider les autres membres, gérer le workflow et le mettre en place m'a permis de creuser un peu plus des notions que je ne connaissais parfois que trop vaguement. Pour le développement, j'ai pu approfondir mes compétences en tests (structure, notion et utilisation du coverage, etc). J'ai également pu mettre en place la gestion de projet dans un cadre plus pratique et exigeant ce qui fait que j'ai pu mieux appréhender sa portée et aussi mettre à l'épreuve ce que j'avais appris lors de cours plus théoriques. Enfin, et surtout, j'ai beaucoup appris sur Scala et pu augmenter mes compétences.

— Thouvenin Axel : J'ai eu l'occasion de travailler plusieurs fois en groupe afin de réaliser, mais je n'avais jamais travaillé avec des personnes aussi fortes en code. J'ai pu améliorer ma technique ainsi que ma gestion de projet.

— Vogt Florian : Ce projet a, pour moi, montré la nécessité d'une gestion de projet stable et clair pour tous les membres du groupe et m'a permis d'apprendre un nouveau langage qui m'était inconnu (Scala) et de m'améliorer sur mes notions de POO.

Sources

this is sources !

Martin ODERSKY & al., 2016, *Programming in Scala* (3rd edition), artima