Département de génie informatique et génie logiciel

INF3995

Projet de conception d'un système informatique

Proposition répondant à l'appel d'offres
no. H2018-INF3995 du département GIGL.

Système mobile de télémétrie en temps réel

Équipe No 3

Charles Hosson	
Erica Bugden	
Fabrice Charbonneau	
Félix Boulet	
Justine Pepin	
Patrick Richer St-Onge	

Février 2018

Table des matières

1. Vue d'ensemble du projet

1.1 But du projet, portée et objectifs (Q2.1 et Q4.1)

Note : Dans l'ensemble de ce document de réponse à la proposition, les termes «notre équipe», «nous», «nos développeurs», «les membres du consortium/de l'équipe» ainsi que «notre groupe» fera référence à l'ensemble des membres de l'entité collaborative de travail réunissant Charles Hosson, Erica Bugden, Fabrice Charbonneau, Félix Boulet, Justine Pepin et Patrick Richer St-Onge.

En réponse à l'appel d'offres no. H2018-INF3995 émanant du département GIGL et latéralement de la société technique Oronos, notre équipe aimerait offrir ses services afin de développer un système mobile de télémétrie en temps réel.

Le produit final sera composé d'un serveur Python et d'une application mobile Android, qui devront répondre à chacun des critères spécifiés dans le document «Exigences techniques - Système mobile de télémétrie en temps réel», qui est le complément au présent document d'appel d'offres no. H2018-INF3995.

Le but du projet peut être découplé en deux parties distinctes. La première peut se résumer en les objectifs du cours INF3995, qui sont les suivants [3] :

- Intégrer les concepts techniques des disciplines du génie informatique et du génie logiciel vus dans quelques cours en 3ème année.
- Appliquer les méthodes de réalisation d'un système embarqué en général.
- Réaliser l'importance et les particularités de l'aspect logiciel dans un système embarqué.
- Offrir à l'étudiant une situation concrète permettant l'application de méthodes de gestion de projet pour le suivi d'un projet informatique.

Cette partie du but est implicite au projet et vise le développement d'une expertise chez le contracteur. Le moyen de son accomplissement passe par la réalisation de l'autre partie du but, c'est-à-dire du développement du système mobile de télémétrie en temps réel tel qu'entrevu par la société technique Oronos.

Les biens livrables attendus peuvent se résumer de façon générale à l'ensemble des fichiers pertinents à la bonne exécution et compréhension du système de télémétrie développé par notre équipe, qu'ils soient des fichiers de sources, de code, de configuration ou de documentation. Les services offerts par notre équipe concernent la production de ces biens livrables en accord avec l'échéancier et le niveau de fonctionnalité spécifiés dans le document d'appel

d'offres no. H2018-INF3995, ainsi qu'un rapport technique documentant ces biens.

1.2 Hypothèse et contraintes (Q3.1)

- Les spécifications du projet sont bien détaillées et facilement vérifiables.
- Les membres du consortium sont tous des étudiants en formation avec une expérience encore limitée dans bien des aspects qui seront mis à l'épreuve lors du développement du système de télémétrie.
    - Corollaire : L'évaluation du niveau de travail requis pour chaque tâche pourra s'avérer très approximative.
    - Corollaire : Le rythme du développement du projet pourrait s'accroître au fur et à mesure de l'expérience croissante des membres de l'équipe.
- Les membres de l'équipe en sont à leur premier projet en tant qu'unité collaborative.
- Les membres de l'équipe disposent de 2 périodes de 4 heures par semaine où ils peuvent collaborer au projet en étant réunis dans un même lieu physique.
- Il y a plusieurs contraintes techniques à respecter dans l'implémentation exigée.
    - L'application serveur doit être faite en langage Python, avec une interface graphique se servant de PyQt5, et doit s'exécuter sur une carte Zedboard de Digilent avec système embarqué Linux Ubuntu.
    - La communication des données peut être faite par socket UDP, mais il est requis d'utiliser l'API de sockets BSD de la librairie standard de Python.
    - L'application Android doit être écrite en Java en servant de l'environnement intégré de développement (IDE) Android Studio.
    - L'utilisation de librairies externes doit être approuvée par le client.
- Les membres de la société technique Oronos possèdent des connaissances techniques suffisamment développées pour que la documentation du programme possède du vocabulaire technique et ne soit pas d'un niveau trop abstrait.

1.3 Biens livrables du projet (Q2.3)

L'appel d'offres précise deux livrables importants du projet. Ces livrables doivent être fonctionnels et réaliser un certain pourcentage des requis de la spécification.

1. Livrable 1 pour le 29 mars 2018 (6 semaines de développement). Ce livrable correspond à la plus grande phase de développement. Toutes les fonctionnalités de base du serveur et de l'application Android seront implémentées.

2. Livrable 2 pour le 12 avril 2018. (2 semaines de développement). Ce livrable est développé sur une plus petite période et ajoute les fonctionnalités en lien avec l'expérience utilisateur et l'aspect visuel.

À la fin du projet, un rapport technique doit être remis et une présentation sera faite pour exposer le projet.

2. Organisation du projet

2.1 Structure d'organisation

La structure d'organisation de l'équipe est majoritairement décentralisée. Cependant, certains membres ont le rôle de responsable ou de spécialiste pour certains éléments du projet afin d'accélérer le processus de prise de décision et minimiser la possibilité de blocages. Étant donné la complexité du projet et la taille de l'équipe, chaque membre occupe plus d'un rôle.

Rôles techniques et de conception

- Responsable de l'architecture serveur : Charles
  - Développeurs de serveur Python : Charles, Erica et Félix
  - Spécialiste en assurance qualité (Serveur, Python) : Charles
- Responsable de l'architecture client : Patrick
  - Spécialiste en UI et organisation visuelle : Félix
  - Spécialiste en communication réseau : Justine
  - Développeurs de client Android : Patrick, Justine, Fabrice, Félix, Charles et Erica
  - Spécialiste en assurance qualité (Client, Java, Android) : Fabrice

Rôles de gestion et de support

- Gestionnaire des services Redmine : Erica
- Spécialiste en gestion des versions : Félix
- Consultant en contrôle de qualité : Félix
- Spécialiste en habiletés personnelles et relationnelles (HPR) : Charles

Les responsables du serveur et du client supervisent le développement de leur partie respective et ils auront le dernier mot quand vient le temps de déterminer si une fonctionnalité est terminée. Puisque selon notre équipe la partie client est plus complexe que la partie serveur, nous avons déterminé qu'il était pertinent d'avoir des rôles spécialisés pour la conception de l'interface utilisateur (UI) et pour l'établissement des connexions ainsi que la communication réseau. Afin d'assurer la qualité du projet, certains membres de l'équipe ont des rôles de contrôle de qualité et d'assurance qualité. De plus, les rôles de gestionnaire de Redmine et de gestion des versions assurent un processus de développement bien structuré. Finalement, le spécialiste en HPR nous permettra de gérer plus facilement les conflits internes qui pourraient survenir dans les mois qui suivent.

Tous les membres de l'équipe vont participer au développement du client puisque celui-ci représente une partie importante du travail à compléter. La quantité de travail qui est nécessaire pour compléter les fonctionnalités du serveur n'est pas assez grande pour justifier d'avoir des membres dédiés uniquement à cet élément du projet.

Le diagramme qui suit illustre la structure d'organisation et les rôles de l'équipe :

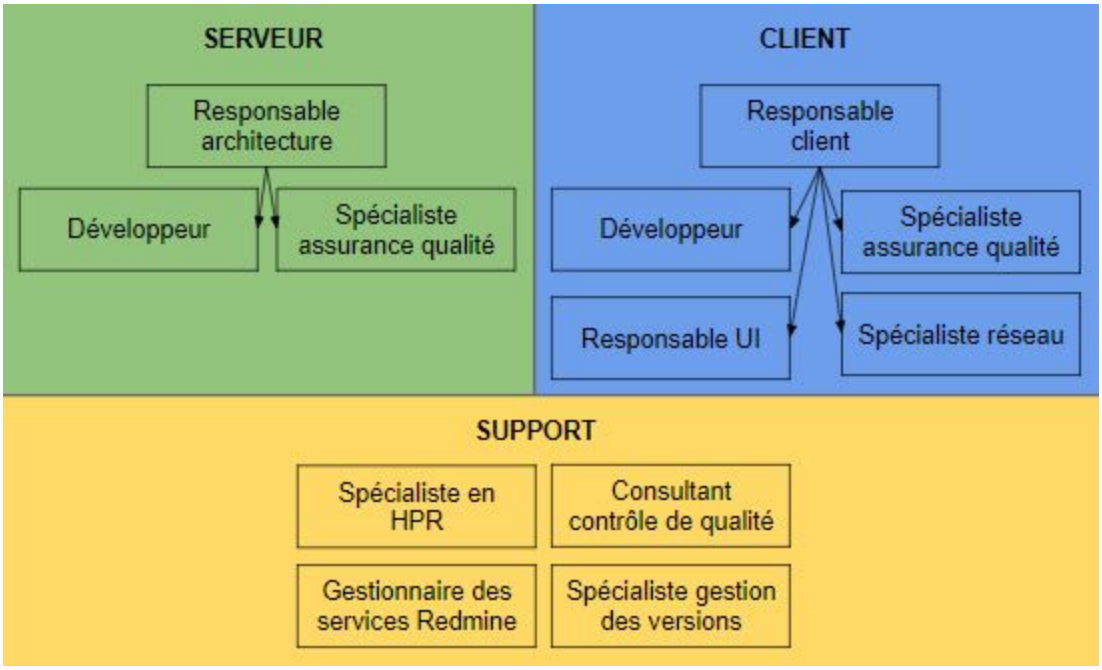

Diagramme 1 : Structure d'organisation de l'équipe

Puisque les responsables d'architecture ont le rôle d'évaluer et réviser les décisions prises par rapport au chemin des données, nous avons inclus des flèches pour représenter le droit de regard que ces individus vont avoir sur le serveur et le client respectivement. On entend par-là que les responsables vont être en mesure de donner les lignes directrices aux autres rôles impliqués dans chaque aspect du projet.

Ainsi, les spécialistes ont une priorité de décision sur les aspects du projet reliés à leur spécialité. Cela étant dit, chaque membre de l'équipe peut participer à la prise de décisions importantes.

2.2 Entente contractuelle

Une entente contractuelle est un document qui contient par écrit les détails relatifs à une entente, notamment ceux concernant les obligations, les engagements et les droits des signataires.

Notre équipe propose au département GIGL et à la société technique Oronos, afin de répondre à l'appel d'offres no. H2018-INF3995 du département GIGL, un contrat de type clé en main. Ce type d'entente engage le contracteur à livrer un produit fini au terme du contrat, et engage le client à effectuer le paiement final lorsqu'il acceptera et prendra possession du produit fini.

Notre équipe a fait le choix de ce type de contrat puisque les spécifications du produit fini concernant l'ensemble de sa qualité ont déjà été établies et sujettes à âpres négociations afin d'en changer la teneur, s'il y a possibilité de changement ou plus probablement de nuance d'interprétation légitime d'une spécification. De plus, le client détient la connaissance rigoureuse de ce pourquoi il a rédigé l'appel d'offres, ce qui l'assure d'ores et déjà de la pertinence de sa sélection d'un ou de plusieurs contracteurs. Quant à ce dernier propos, notre équipe est consciente que le client se réserve le droit de contracter plusieurs soumissionnaires au sujet de l'appel d'offres no. H2018-INF3995 du département GIGL, ce qui, sous un type de contrat octroyant une plus grande liberté aux contracteurs, serait dévastateur pour le client en termes de temps passé à suivre différentes solutions en parallèle.

Donc, de par le niveau élevé de détail requis pour les spécifications à l'émission de l'appel, de par la très grande inertie desdites spécifications, de par la maîtrise plus que précise, par le client, de cette vision que devra matérialiser le contracteur, ainsi que de par le faible niveau de suivi demandé par ce type de contrat, nous proposons un contrat clé en main au client concernant l'appel d'offres no. H2018-INF3995 du département GIGL.

3. Solution proposée

3.1 Architecture logicielle sur serveur (Q4.2)

Le diagramme qui suit résume les éléments principaux de l'architecture du serveur Python (page suivante) :

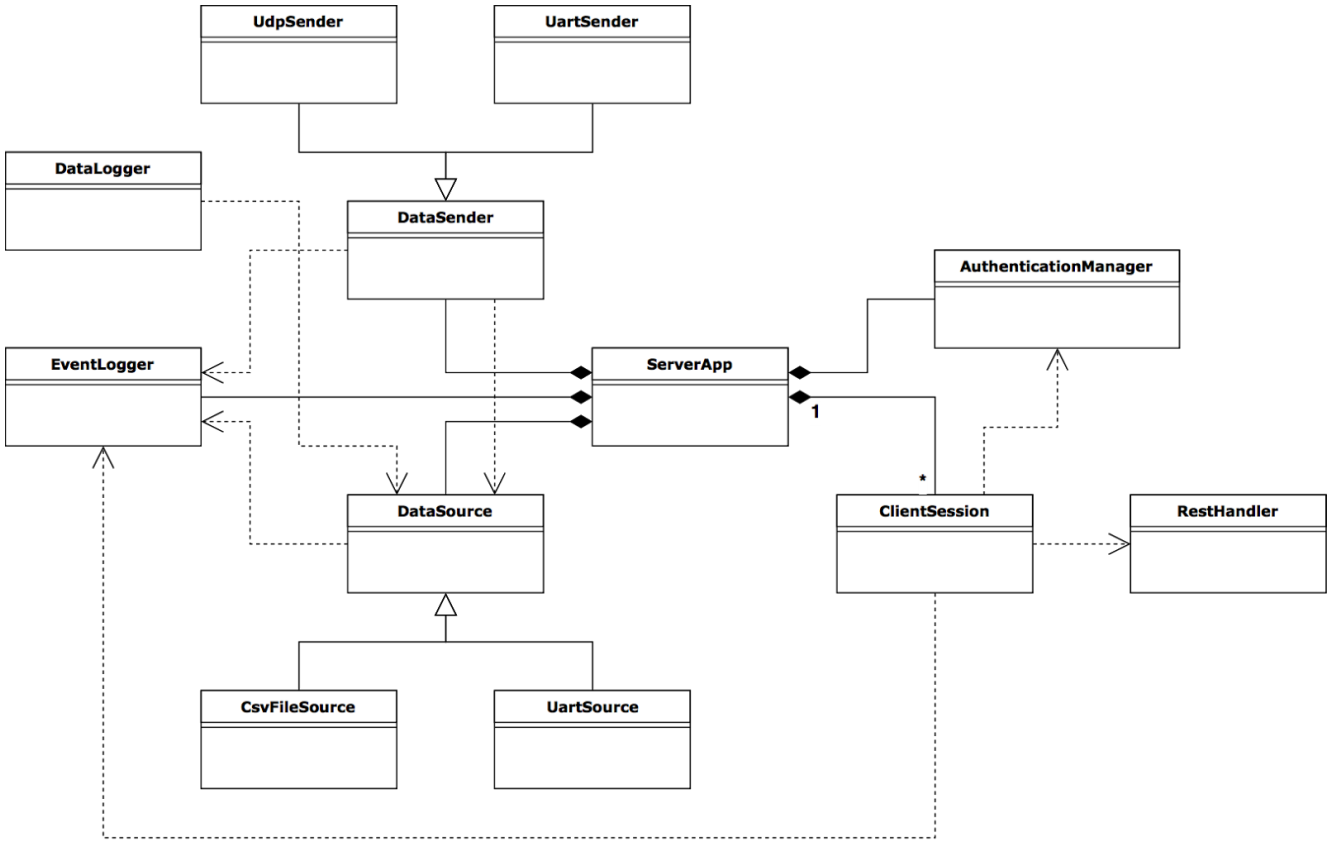

Diagramme 2 : Architecture du serveur

Il est bon de noter que dans le diagramme ci-dessus, les noms des classes ne sont là qu'à titre indicatif de la structure générale du logiciel et sont sujets à changement.

On peut voir sur le diagramme qu'il y a deux groupes principaux de fonctionnalités à implémenter : la gestion des requêtes de configuration et la réception/transmission des données.

Les tâches de réception (par UART ou à partir d'un fichier) et de transmission (par UDP ou UART) sont exécutées en parallèle. La tâche de réception possède un tampon (style FIFO) de sortie pour stocker ses résultats et les tâches d'envoi et d'enregistrement des données sont abonnées au tampon de la réception. La classe principale d'application s'occupe, à travers des *Factory*, de créer les instances des classes qu'elle possède.

L'authentification et la gestion des sessions de clients est exécutée en parallèle avec la réception/transmission. Un gestionnaire de *logging* s'occupe d'enregistrer les événements générés par les autres modules.

3.2 Architecture logicielle sur tablette (Q4.3)

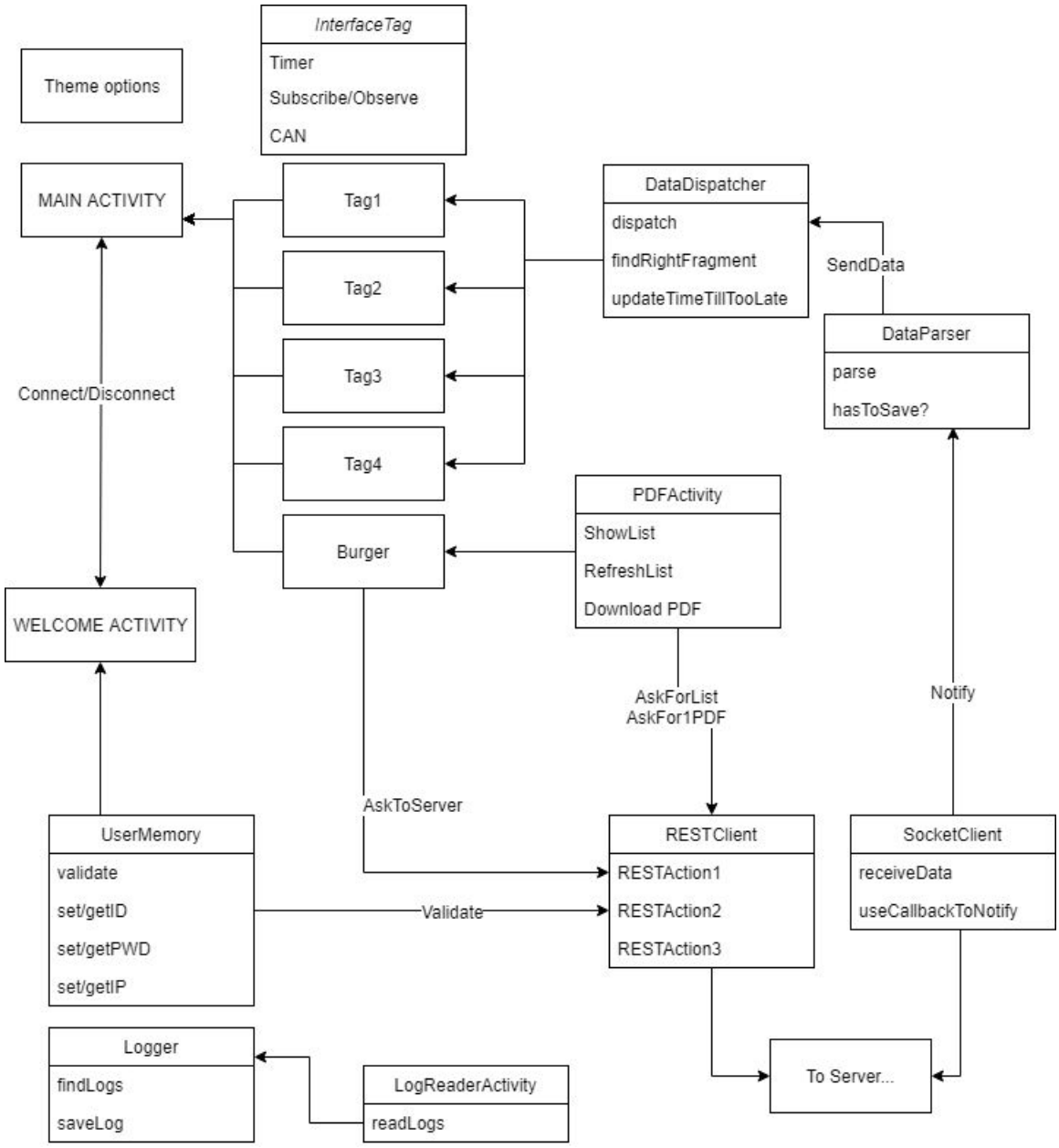

Diagramme 3 : Architecture du client

Le premier écran visible pour l'utilisateur est le *WelcomeActivity*, qui permet à l'utilisateur de se connecter au serveur (*ip*, *username*, *password*). Une fois connecté, l'utilisateur aura accès à l'écran principal de visualisation de données provenant de la fusée. L'écran principal sera le *MainActivity* et c'est à cet endroit que l'interface dynamique sera générée à partir des fichiers XML. Les tags des fichiers XML seront implémentés par des composants Android différents

(probablement des fragments imbriqués) qui serviront à la construction du UI. Le fonctionnement du client se base sur le patron de conception de type *Observer* où les composants visuels s'abonnent à certaines variables et reçoivent des notifications lorsque les données rentrent par le socket et modifient la valeur de ces variables. Afin que les données atteignent les bons tags sous la bonne forme de présentation, nous aurons besoin d'un *DataParser* et d'un *DataDispatcher* qui assureront le relai de tout ce qui réussira à parvenir au socket. Finalement, un *Logger* sera responsable d'enregistrer en mémoire tout message généré par l'application Android. Tout autre classe peut bien entendu signaler un événement à enregistrer au *Logger*.

4. Processus de gestion

4.1 Estimations des coûts du projet

Coûts liés au personnel :

Le projet se déroule sur deux périodes de quatre (4) heures par semaine, donc huit (8) heures par semaine. Nous prévoyons que l'équipe travaillera au total 329 heures, en tant que développeurs-analystes, au coût de 130$/h. Un coordonateur de projet travaillera également pour l'équipe, dont on évalue le temps total à 72 heures, au coût de 145$/h. Cela revient à un coût total de 53 210 $, soit 42 770$ pour les développeurs-analystes et 10 440 $ pour le coordonateur de projet.

Coûts liés à l'équipement :

L'équipement n'entraînera aucun coût, puisque l'on considère la tablette Android et le Zedboard comme une gracieuseté de l'école. De même, les ordinateurs personnels des développeurs ne sont pas comptés dans les coûts puisqu'ils servent autrement au consortium. Les outils de développement utilisés sont tous gratuits.

4.2 Planification des tâches (Q2.2)

Les tâches divisant le projet seront effectuées sur 4 plages de temps de deux semaines, exceptée la deuxième, qui sera de trois semaines (deux si on ne compte pas la semaine de lecture). Le diagramme de Gantt suivant présente la planification des tâches sur ces périodes de temps.

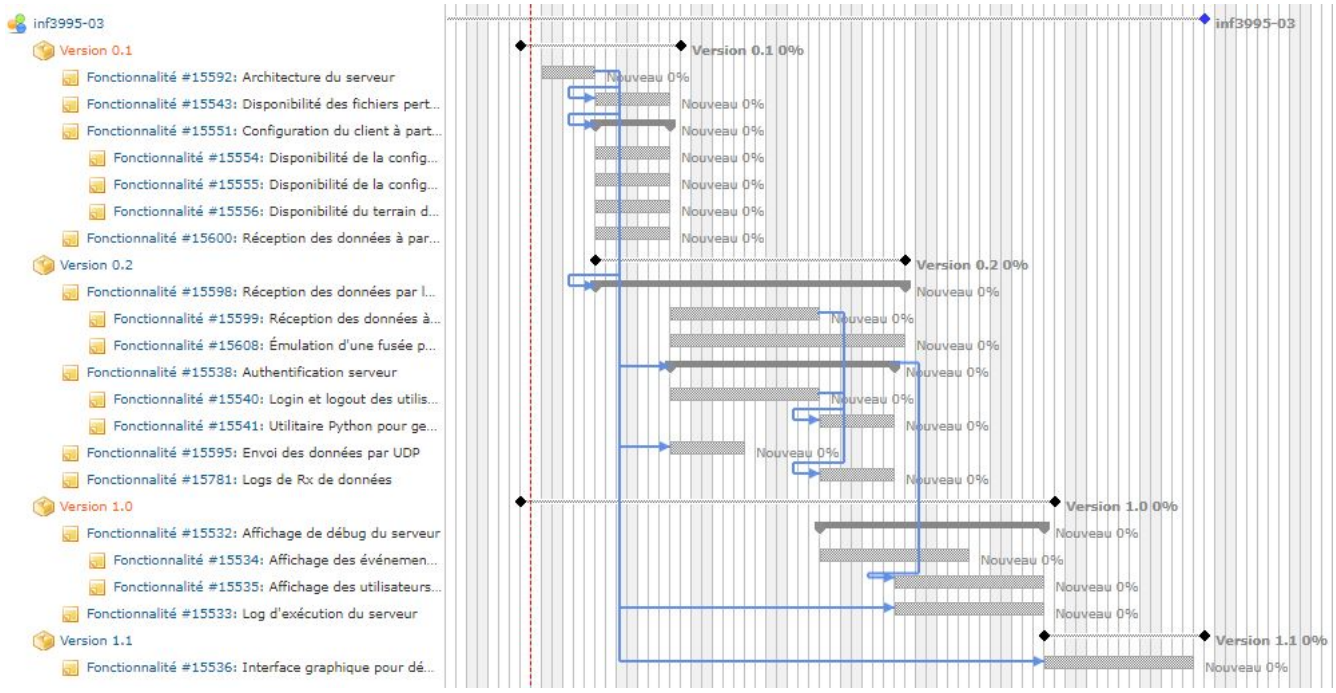

Diagramme 4 : Visualisation des échéances du serveur

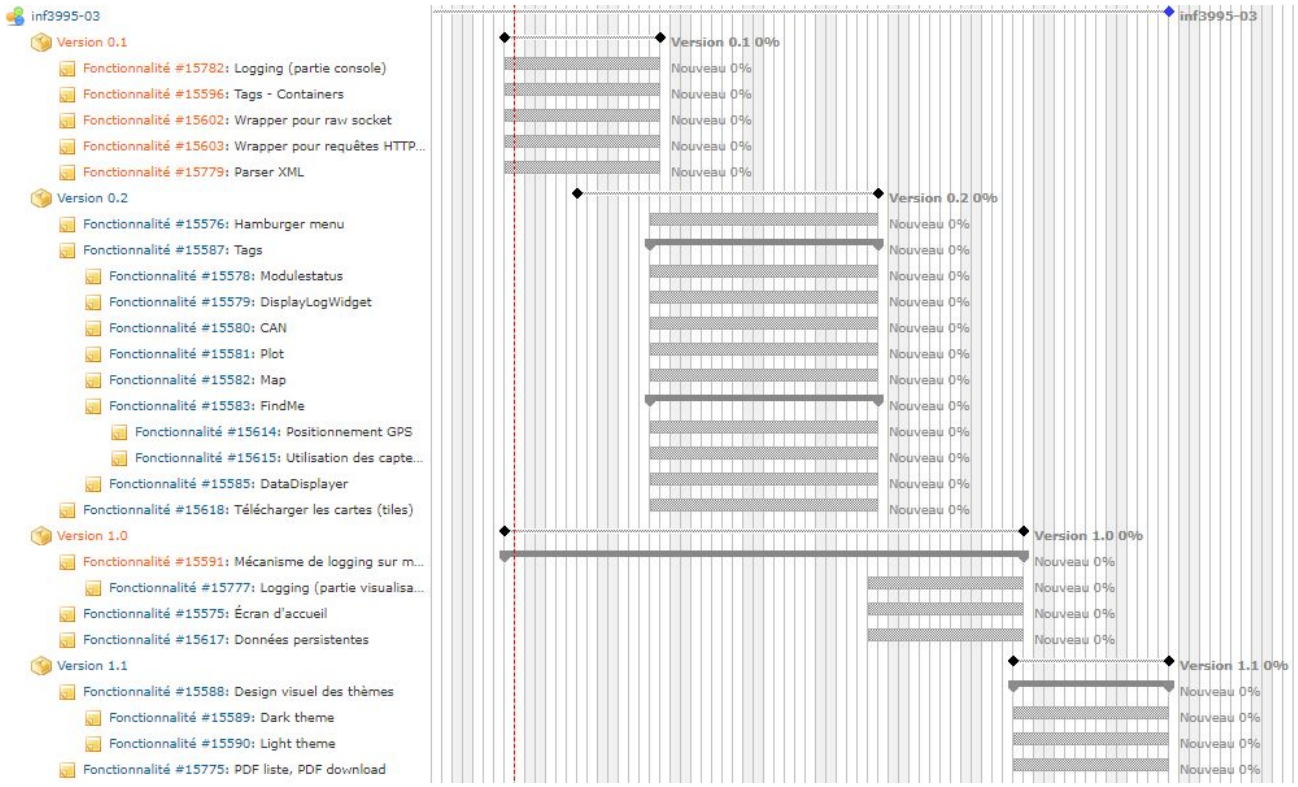

Diagramme 5 : Visualisation des échéances du client

4.3 Calendrier de projet (Q3.3)

Le tableau qui suit décrit les dates de terminaison de phases importantes dans le processus de développement. Nous avons choisi de sortir une nouvelle version fonctionnelle à chaque deux semaines.

Date	Description	Détails
22 février	Version 0.1	Serveur : Fonctionnalités de base du serveur (e.g. Interface REST, lecture d'un CSV) Client : S'occuper de l'interface vers le serveur (Interface REST & Socket de réception du flot), conception de base de l'UI, analyseur syntaxique.
15 mars	Version 0.2	Serveur : Envoi des données au client, authentification des clients Client : Fonctionnalités reliées aux différents tags.
29 mars	Version 1.0 (Livrable 1)	Serveur : Affichage débug & logging Client : Enregistrement en mémoire des logs, écran d'accueil.
12 avril	Version 1.1 (Livrable 2)	Serveur : Interface graphique pour démarrage du serveur Client : Arranger les thèmes et le visuel de l'application Android.
10 ou 12 avril	Présentation du projet	Exposer les réalisations de notre équipe

Tableau 1 : Résumé des versions et livrables

4.4 Ressources humaines du projet

Pour compléter ce projet, il sera nécessaire d'avoir une petite équipe formée de membres avec une expertise technique variée et pertinente selon les exigences techniques du projet. Nous proposons les individus suivants :

Nom	Adresse	Courriel	Téléphone
Charles Hosson	3995 avenue Dupuis, apt 9 Côte-des-neiges, QC,	charles.hosson@polymtl.ca	(819) 323-6673
Patrick Richer St-Onge	441A Lansdowne, Rosemère, QC, J7A 3G6	patrick.rst@polymtl.ca	(514) 705-5594

Justine Pepin	354 rue de Grenoble, Sainte-Julie, QC, J3E 1A2	justine.pepin@polymtl.ca	(438) 405-4190
Félix Boulet	41 rue du Blainvillier, Blainville, QC, J7C 5B1	felix.boulet@polymtl.ca	(450) 543-2090
Fabrice Charbonneau	7667 Avenue de Chateaubriand, Montréal, QC, H2R 2M2	fabrice.charbonneau@polymtl.ca	(514) 910-0464
Erica Bugden	4452 Avenue de Melrose, Montréal, QC, H4A 2H6	erica.bugden@polymtl.ca	(438) 863-4181

Tableau 2 : Membres du consortium et coordonnées

Expertises partagées par l'ensemble du consortium :

- Expérience avec Git et SVN pour la gestion de version.
- Expérience en développement web (HTML/CSS/JS, etc.).
- Expérience en programmation de FPGA (Vivado, VHDL).
- Expérience avec Python.

Expertises spécifiques

Charles Hosson :

- Expérience en design d'application de capture de mouvement et de contrôles multimédia.
- Ancien membre de la société technique Smart Bird dans laquelle un logiciel d'acquisition de données de vol et de vision par ordinateur a été développé (en C++ avec Qt).
- Membre de la société technique Poly eRacing où un système de télémétrie similaire est développé.
- Expérience en programmation parallèle et familiarité avec les concepts plus bas niveau de synchronisation non bloquante.
- Expérience en programmation d'acquisition de données et en programmation réseau.

Patrick Richer St-Onge :

- Membre de la société technique Poly eRacing où un système de télémétrie similaire est développé.
- Expérience en programmation d'applications Android lors d'un projet d'équipe.
- Très bonne connaissance des logiciels embarqués et de GNU/Linux.

Justine Pepin :

- Expérience en révision et en rédaction de documentation technique.

- Expérience en rédaction de documentation de tests.
- Expérience comme développeuse et comme chef de projet pour une équipe d'environ six membres au sein de la société technique Élikos.
- Expérience les avec quelques librairies Python les plus communes (numpy, matplotlib,...).
- Expérience en Java et en protocoles de télécommunication IP avec architectures serveurs-clients.

Félix Boulet :

- Expérience avec plusieurs librairies Python (numpy, TensorFlow + Keras, etc.).
- Expérience importante en Java (4 ans) : Android, LibGDX, Swing, Sockets, multi-threading.

Fabrice Charbonneau :

- Expérience en gestion de projet découlant des projets intégrateurs antérieurs et d'un stage.
- Expérience avec Python, bonnes connaissances de la conception logicielle.
- Bonne connaissance de l'utilisation de GNU/Linux.

Erica Budgen :

- Conception et développement de logiciel embarqué en modèle *multithread*.
- Connaissances plus approfondies en systèmes d'exploitation (e.g. allocation et gestion de mémoire).

5. Suivi de projet et contrôle

5.1 Contrôle de la qualité

Le contrôle de la qualité est primordial dans un projet d'une telle envergure. Celui-ci permet en effet de s'assurer que le produit sera facile à maintenir, facile à comprendre et exempt de problèmes et de bugs à la livraison.

Afin d'obtenir une façon concrète d'évaluer la qualité de chaque bien livrable en cours de développement et pour chacune des remises, un guide d'assurance-qualité a été développé et est disponible en annexe (annexes 1 et 2).

Un guide complet (annexe 1) sur la procédure à respecter pour intégrer une fonctionnalité fraîchement développée au projet est d'abord énoncé. Ce guide décrit chacune des étapes à respecter afin de se conformer au processus qui sera appliqué par les spécialistes de l'assurance-qualité. Tout manquement à

celui-ci devra être pris en charge par les membres concernés afin d'assurer une qualité irréprochable aux livrables et aux fonctionnalités sous-jacentes.

S'ensuit un autre guide (annexe 2), qui a pour but de définir des critères de codage clairs à respecter, ainsi que des suggestions pour aider chacun des membres à respecter les différentes normes établies par les spécialistes de l'assurance-qualité pour chacune des fonctionnalités du projet. Les suggestions incluent notamment des environnements de développement à utiliser et des astuces pour la gestion de versions.

5.2 Gestion de risque (Q2.6)

Ce tableau décrit les principaux risques potentiels que nous avons identifiés dans le processus de développement.

Description du risque	**Probabilité**	**Sévérité**	**Mitigation possible**
Manque de temps ou de motivation pour faire proprement l'assurance qualité (versionnage et testing) à l'approche des échéances	Moyenne	Basse	Faire confiance aux membres de l'équipe, user de rétroaction constructive en cas de manquement
Changement des requis du client (Adrien) ou du promoteur (Jérôme)	Basse	Moyenne	Toujours rester en contact pour ne pas que les changements restent sans analyse plus que quelques jours
Mauvaise estimation des temps de développement	Haute	Moyenne	Évaluer les incertitudes sur les prévisions (qualitativement) et prévoir des contingences de temps pour les estimations plus incertaines
Un développeur manque de temps ou d'expertise pour compléter une tâche	Basse	Haute	Toujours avoir deux développeurs au courant de chaque tâche ou en programmation par paires pour que la réponse soit rapide
Implémentation incorrecte d'une fonctionnalité (mauvaise compréhension des requis) ou utilisation d'une technologie non permise	Basse	Haute	Négocier directement avec le client pour toute question d'implémentation et faire des revues de design pour être sûr que tout le monde est sur la même page

Tableau 3 : Résumé des risques du projet

5.3 Tests (Q4.4)

L'objectif est d'implémenter des tests pour le plus d'éléments possibles. Des tests seront effectués pour suivre de près le développement logiciel. Des tests unitaires seront implémentés pour les fonctions pertinentes au niveau du serveur et au niveau du développement Android.

Les tests unitaires au niveau du serveur devront vérifier le bon fonctionnement de diverses opérations, par exemple:

- Recevoir correctement les données du port USB du Zedboard
- Traiter les données reçues (au besoin), comme les thresholds, ou la conversion en d'autres unités (par exemple la vitesse en m/s ou km/h)
- Envoyer correctement les données vers le client (format des données, débit des données, bon fonctionnement des sockets)
- Bien authentifier les clients (réagir correctement à l'ajout d'utilisateur et bien chiffrer les mots de passe)

Chez le client, des tests d'intégration devront être implémentés afin de vérifier la création de certains éléments du UI. Ceux-ci ne seront pas, dans la majorité des cas, des tests très élaborés puisque nous pourrons tester les éléments du UI directement en observant l'application sur un simulateur Android ou sur un appareil. Toutefois, il nous sera possible de rédiger une liste de type «à cocher» où tous les éléments à vérifier visuellement seront répertoriés et dont le développeur pourra se servir afin de n'oublier aucun élément. D'autres tests unitaires plus poussés seront faits afin de tester d'autres éléments du client qui sont plus ou moins en lien avec l'affichage. Les tests unitaires sur Android pourront être effectués en utilisant les plateformes JUnit et Mockito.

Afin de s'assurer que les tests et plus globalement que l'assurance qualité garde le bon cap et ne se perde pas au fil du temps qui passe, un spécialiste de l'assurance qualité sera nommé pour le serveur et un spécialiste de l'assurance qualité sera nommé pour le client. Ces deux spécialistes devront s'assurer que tous les développeurs de l'équipe contribuent à produire les tests unitaires, appliquent les tests avant de partager sur le gestionnaire de version tout code ajouté et/ou modifié et vérifient visuellement les éléments qui sont sur la liste de type «à cocher». Ils sont également responsables, le cas échéant, de rédiger et de tenir à jour cette liste. Les spécialistes sont désignés dans la section structures d'organisation.

5.4 Gestion de configuration

En conformité avec les contraintes du projet, le logiciel de gestion de versions utilisé par l'équipe est git. Un seul répertoire sera utilisé pour l'ensemble du projet. La branche principale *master* sera utilisée lorsque nous sortirons une

nouvelle version du serveur Python ou de l'application Android, donc lors de la fin d'une étape interne de développement et lors des livrables. Les branches *dev-server* et *dev-android* seront utilisées pour le développement actif des programmes et pour l'intégration des nouvelles fonctionnalités. Cette approche simplifiée est basée sur le modèle de GitFlow [5].

Un membre de l'équipe sera en charge de la gestion de configuration. Cette personne devra s'assurer que tous les membres contribuent à faire de l'entrepôt git un espace organisé et entretenu selon les conventions décidées au sein de l'équipe. Dans l'éventualité où un membre de l'équipe dévierait des conventions d'équipe, ce spécialiste en gestion des versions sera responsable d'avertir ce membre et de lui expliquer les bonnes procédures à suivre afin d'éviter l'accumulation d'erreurs. Il est désigné dans la section *Structures d'organisation.*

Afin de permettre la réutilisation du code et une transition en douceur entre nos développeurs et les développeurs d'Oronos, l'ensemble du code sera documenté. Le code Python du serveur sera documenté avec des *docstrings* selon le standard PEP 257 [1]. Le code Java de l'application Android sera documenté selon le standard de Javadoc [4]. Des commentaires supplémentaires pourront être ajoutés au code source afin de préciser certaines opérations.

6. Références (Q3.2)

[1] D. Goodger, G. v. Rossum. (2001) PEP 257 -- Docstring Conventions [En ligne].
Disponible : https://www.python.org/dev/peps/pep-0257/

[2] G. v. Rossum, B. Warsaw, N. Coghlan. (2001) PEP 8 -- Style Guide for Python Code. [En ligne].
Disponible : https://www.python.org/dev/peps/pep-0008/

[3] J. Collin. (2018) Projet de conception d'un système informatique - Plan de cours - Hiver 2018. [En ligne].

[4] Oracle. (2012) How to Write Doc Comments for the Javadoc Tool. [En ligne].
Disponible :
http://oracle.com/technetwork/java/javase/documentation/index-137868.html

[5] V. Driessen. (2010) A successful Git branching model. [En ligne].
Disponible : http://nvie.com/posts/a-successful-git-branching-model/

ANNEXES

Annexe 1 - Processus d'assurance-qualité

Les étapes énoncées ici décrivent comment devrait se faire une l'ajout d'une nouvelle fonctionnalité au projet. Le masculin et le singulier sont utilisés sans discrimination :

1	Après avoir été assigné à une fonctionnalité ou une tâche sous-jacente, le développeur met à jour sa copie locale du projet en effectuant un `git pull` sur la branche d'où il veut démarrer l'ajout de la fonctionnalité.
2	Avant de commencer à travailler, le développeur teste localement la version actuelle du projet afin de vérifier si aucun problème non-relevé précédemment ne survient sur sa machine locale.
3	Après cette vérification, le développeur crée une branche à partir de celle qu'il a choisie pour sa fonctionnalité, en utilisant `git checkout -b leNomDeLaBranche`.
4	Il peut à présent débuter son travail, et doit notifier le spécialiste de l'assurance-qualité (s'il y en a un assigné à la fonctionnalité) de la branche qui sera à vérifier à la fin du travail. Si aucun responsable n'a été assigné au préalable, le développeur doit en assigner un avant de commencer.
5	Une fois la fonctionnalité complétée, le développeur s'assure d'avoir respecté toutes les normes d'assurance-qualité établies pour le projet (énoncées à l'annexe 2) et pour la fonctionnalité en question (détaillées sur la demande Redmine pour la fonctionnalité). Il peut alors notifier le spécialiste de l'assurance-qualité que sa branche de fonctionnalité est valide pour évaluation, et en changer le statut sur Redmine.
6	Le spécialiste de l'assurance-qualité effectue un `git checkout` et un `git pull` sur la branche mentionnée afin d'obtenir les modifications effectuées par le développeur.
7	Le spécialiste de l'assurance-qualité lance les tests automatisés créés par le développeur, et en ajoute au besoin (si un cas de figure non-décelé

	auparavant survient, par exemple). Si tous les tests passent, il procède à la prochaine étape. Sinon, le développeur est notifié des tests qui échouent. Le spécialiste de l'assurance-qualité peut assister le développeur pour résoudre les différents tests, au besoin.
8	Le spécialiste de l'assurance-qualité vérifie que toutes les normes d'assurance-qualité sont respectées par rapport au projet et à la fonctionnalité en question. De petits manquements (du *lint*, par exemple) pourront être corrigés à la discrétion du spécialiste, alors que de gros manquements devront être signalés au développeur, et provoquer un refus de l'intégration de la fonctionnalité.
9	Si le spécialiste de l'assurance-qualité juge que la fonctionnalité répond aux critères de qualité du code et que tous les tests passent, il peut demander au reste de l'équipe de vérifier si la fonctionnalité fonctionne de façon appropriée sur les différents postes de travail et appareils. Ce n'est toutefois une obligation que pour les fonctionnalités majeures (qui provoquent des *breaking changes*).
10	Le spécialiste peut alors intégrer la fonctionnalité à la branche de développement de départ, en effectuant au préalable un `git rebase` si nécessaire, puis un `git merge`. Comme le dépôt sur Redmine ne supporte pas les *pull requests*, le développeur est responsable de vérifier le suivi de son travail avec le spécialiste de l'assurance-qualité.
11	Le développeur ou le responsable doivent mettre le statut de la fonctionnalité ou de la tâche sous-jacente à "Complété" sur Redmine, et comptabiliser le total d'heures passées en développement comme des activités "Implémentation" et "Tests". Au besoin, des heures peuvent également être comptées en "Conception".

Annexe 2 - Normes techniques d'assurance-qualité pour le projet : Python et Android

Environnement de développement

- Pour Python, il est fortement suggéré d'utiliser un éditeur avec auto-complétion (Intellisense) et intégration de PyLint. De bons choix sont *PyCharm* de *Jetbrains* (https://www.jetbrains.com/pycharm/) ou encore

Visual Studio Code de *Microsoft* (https://code.visualstudio.com/). Il est à noter que l'extension Python (officielle, de Microsoft) et le module *PyLint* doivent être installés pour que VS Code soit conforme aux normes d'assurance-qualité requises :

- `pip install pylint`

Tout autre éditeur (Vim, Notepad++, Netbeans, etc.) est accepté pour autant qu'il soit correctement configuré et réponde aux normes susmentionnées.
Le serveur Python devrait être testé régulièrement sur le Zedboard afin de vérifier son bon fonctionnement sur l'architecture spécifique de celui-ci. Bien sûr, il est recommandé de développer sur vos machines locales plutôt que directement sur le Zedboard.

- Pour Android, il est **obligatoire** d'utiliser Android Studio en version 3.0, car c'est un requis de la spécification du projet. Par défaut, Android Studio utilise Intellisense pour la complétion de code. À des fins de test, il est recommandé d'utiliser un émulateur qui correspond aux requis du projet, soit une tablette **Nexus 9** sur l'**API 23 (Android 6.0)**. Il peut également être bénéfique de tester sur un appareil physique (la tablette ou un téléphone équipé d'Android 6.0 ou plus récent) afin de vérifier tout problème de compatibilité éventuel. Le projet doit être développé en utilisant **Java 8 (JDK 1.8)**.

- Il est recommandé d'utiliser un client graphique (GUI) pour la gestion de versions avec Git. De bons choix sont :
  - GitKraken (https://www.gitkraken.com/)
  - SourceTree (https://www.sourcetreeapp.com/)
  - L'outil intégré à Visual Studio Code
  - Les outils intégrés aux IDEs Jetbrains (PyCharm, Android Studio).

  Si vous êtes à l'aise avec la ligne de commande, vous pouvez également utiliser Git de cette façon. Toutefois, il est primordial de **savoir ce que vous faites** si vous décidez de procéder ainsi, car des critères stricts sont associés à la révision de code et aux actions nécessaires pour qu'une fonctionnalité soit validée. Tout manquement entraînant une destruction de l'historique du projet ou une perte quelconque de fonctionnalité est en violation des règles du contrôle de la qualité.

Qualité du code Python

- Utilisation obligatoire de PyLint (intégré à de nombreux éditeurs, dont PyCharm et VS Code). Ce linter se conforme aux normes PEP8, mais pourra être ajusté au besoin.

- Prioriser une programmation orientée-objet, avec des classes et tous les principes annexes :

    - Bien définir l'encapsulation des variables et des fonctions.
        - `variable` → public
        - `_variable` → protected
        - `__variable` → private

    - Faire usage d'héritage et de polymorphisme lorsque possible.

    - Utiliser un ou plusieurs patrons de conception approprié(s). Dans le cadre d'un serveur, cela pourrait inclure :
        - Command
        - Mediator
        - Observer
        - Strategy
        - Composite
        - Proxy
        - Facade
        - etc.

    - Respecter les principes SOLID :
        - Single responsibility principle : Une classe n'a qu'une seule responsabilité
        - Open/closed principle : On devrait pouvoir ajouter des parties à un objet, mais éviter d'avoir à le modifier
        - Liskov substitution principle : On devrait pouvoir remplacer l'instance d'une superclasse par une classe fille sans problème (exemple : RocketComponent devrait pouvoir être remplacé par RocketEngine)
        - Interface segregation principle : On doit pouvoir bien séparer ce qui relève de différentes interfaces.

        - Dependency inversion principle : On doit dépendre des abstractions, pas des implémentations (une classe abstraite devrait se servir d'interfaces).

Qualité du code Android (Java)

- Se fier aux normes de Java, utiliser régulièrement l'outil de formatage de Android Studio (Ctrl+Alt+L).

- Obligatoirement se servir d'une programmation orientée-objet, avec des classes et tous les principes annexes :

    - Bien définir l'encapsulation des variables et des fonctions.
        - public variable
        - protected variable
        - private variable

    - Faire usage d'héritage et de polymorphisme lorsque possible.

    - Utiliser un ou plusieurs patrons de conception approprié(s). Dans le cadre d'une application Android, beaucoup de patrons sont déjà implémentés à même le fonctionnement du code (Activity, Intent, etc.). Toutefois, on devrait pouvoir faire usage des suivants :
        - Command
        - Singleton
        - Composite
        - Facade
        - Proxy
        - Chain of Responsability
        - Command
        - Mediator
        - Observer
        - State
        - Template method
        - Visitor

    - Respecter les principes SOLID, tout comme en Python.

- Prioriser l'utilisation des layouts XML pour construire les interfaces visuelles de l'application. Tout écrire en Java directement (donc construire l'interface à partir du code) surcharge le code inutilement alors qu'il y a de nombreux outils intégrés à Android Studio pour nous faciliter la vie avec tout cela.