POLYTECHNIQUE MONTRÉAL

UNIVERSITÉ D'INGÉNIERIE

Département de génie informatique et génie logiciel

INF3995

Projet de conception d'un système informatique

Proposition répondant à l'appel d'offres
No. A2019-INF3995 du département GIGL.

Chaîne de blocs pour dossiers d'étudiants

Équipe No 1

Bernard Meunier
Charles Marois
Francis Granger
Gabriel Pollo
Rose Hirigoyen

Septembre 2019

Table des matières

1. Vue d'ensemble du projet

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

Le but du projet est de créer un registre contenant l'information sur les étudiants de l'école Polytechnique Montréal ainsi que la liste des cours auxquels ils assistent au fil du temps. Ce système sera partagé avec un ensemble d'employeurs potentiels d'étudiants et de gradués de Polytechnique Montréal pour qu'ils puissent avoir accès à l'information. L'utilisation d'un registre distribué rendra l'information inviolable et fiable pour tous.

Le système à concevoir sera composé de plusieurs éléments. Premièrement, l'utilisation d'un registre distribué nécessite l'utilisation d'une chaîne de blocs pour garder en mémoire chaque modification, ainsi que trois mineurs utiles à agrandir cette chaîne. Ces mineurs devront communiquer entre eux, mais aussi avec un serveur central. Celui-ci devra s'occuper de faire travailler les mineurs et servira d'interface entre la chaîne de blocs et les clients du système.

L'information contenue dans le système sera accessible par une application Android de différentes façons. Deux types de comptes pourront y avoir accès, des comptes d'édition et des comptes de consultation. Le type d'édition sera conçu pour le personnel de Polytechnique, qui devra entrer les informations sur les dossiers que l'on retrouve sur la chaîne de blocs. Il pourra aussi consulter l'information qui se trouve dans la chaîne. Les comptes de consultation seront utilisés par les employeurs potentiels et les étudiants, et ils ne pourront que consulter l'information.

Un client administrateur sur desktop est requis pour faire la gestion des comptes. Il pourra créer et supprimer les deux types de comptes. De plus, le client desktop aura accès aux logs du serveur qui consisteront en des messages d'erreur, de modification de la chaîne de blocs et autres.

1.2 Hypothèse et contraintes (Q3.1)

Plusieurs contraintes sont imposées sur le projet. Pour commencer, le serveur et les mineurs devront être conçus en C/C++. Ils seront installés sur une carte de développement ZedBoard et seront exécutés sur Arch Linux. L'application Android devra être développée en Java ou Kotlin sur Android Studio et devra fonctionner avec les tablettes fonctionnant avec Android 7 et plus. Pour l'application desktop, aucun langage n'est imposé, mais elle ne peut pas être réalisée avec des technologies web. Elle doit être nativement compilée sur Windows ou Linux.

Le projet doit être complété en un maximum de 480 heures-personnes. On estime qu'il sera réalisé par une équipe de 5 développeurs. Les librairies importantes choisies pour la réalisation du projet devront être mentionnées, et

l'utilisation de logiciels supplémentaires devra être validée par le promoteur. De plus, toute documentation nécessaire pour l'utilisation du système, telles les commandes d'installation et les fichiers de configuration, devront être fournis avec le projet.

1.3 Biens livrables du projet (Q2.3)

Trois livraisons sont prévues lors de la session :

1. Le jeudi 26 septembre 2019. Livraison d'un premier prototype comprenant une application Android de base, un serveur web avec l'interface REST ainsi qu'un mineur. L'application Android et le mineur devront être en état de communiquer avec le serveur web. Les communications peuvent consister uniquement de messages logs à ce point.

2. Le jeudi 7 novembre 2019. Un livrable intermédiaire doit être remis. Celui-ci doit comprendre un serveur web capable de gérer les transactions et contrôler le minage des mineurs. Il doit être aussi en mesure de communiquer avec un seul compte d'édition de sur la tablette Android. Les trois mineurs doivent être capables de miner une chaîne fonctionnelle sans perte possible d'un nœud et doivent produire localement des logs sur leur sortie standard. L'application Android doit avoir son mode d'utilisateur éditeur complètement fonctionnel. Finalement, l'application PC doit pouvoir recevoir les logs du serveur web dans son mode administrateur, mais sans la fonctionnalité de création de comptes.

3. Le jeudi 28 novembre 2019. Le livrable final doit être finalisé pour cette date. Celui-ci rajoutera la fonctionnalité de démarrage automatique du serveur web avec *systemd* dans ArchLinux ainsi que le fonctionnement complet des comptes éditeur et de consultation. Les mineurs devront être capables de gérer la déconnexion d'un ou de deux mineurs, d'envoyer des logs à l'application PC et de s'amorcer automatiquement avec le systemd de ArchLinux. L'application PC devra avoir un mode de consultation sans possibilité d'édition en plus de la possibilité de création et suppression de compte. Elle doit aussi pouvoir voir tous les logs émis par le système.

Des instructions d'installation et les fichiers de configuration nécessaires au fonctionnement du système doivent être fournis à chaque livraison.

2. Organisation du projet

2.1 Structure d'organisation

Notre équipe est composée de 5 développeur-analystes, dont un membre qui prendra en charge à temps partiel la coordination du projet. La structure d'organisation de l'équipe sera majoritairement décentralisée. Toutefois, le coordinateur du projet sera en charge de faire les suivis avec chacune des équipes de travail et avec le client lors des différentes remises. Il aura aussi la tâche d'accélérer la prise de décisions en réunion et de minimiser les blocages possibles.

Responsabilités techniques et de conception

- Architecture logicielle sur serveur et chaîne de blocs
    - Développement du serveur web en C++
    - Développement de la chaîne de blocs et de la communication réseau
    - Suivi de l'assurance qualité (serveur, réseau)
- Architecture logicielle sur Android
    - Développement du client Android en Kotlin
    - Développement du UI et de la présentation visuelle
    - Suivi de l'assurance qualité (client, Kotlin, Android)
- Architecture logicielle sur PC
    - Développement du client PC avec JavaFX
    - Suivi de l'assurance qualité (client, Java)

Pour les responsabilités techniques et de conception, nous n'avons pas dédiés de responsables spécifiques. Nous proposons plutôt de changer de tâche de façon hebdomadaire pour que chaque membre puisse travailler sur chaque section du projet et que tous les membres soient sur le même longueur d'onde. Toutefois, on s'attend à ce que chaque membre fasse un suivi serré de l'avancement des fonctionnalités. Pour qu'une fonctionnalité soit acceptée et ajoutée au projet, elle doit être obligatoirement approuvée par deux autres membres. Ceci nous assure une couverture adéquate de nos fonctionnalités par la majorité des membres de l'équipe.

Selon notre équipe, le développement du serveur C++ et de la chaîne de blocs seront les parties les plus complexes du projet, c'est pourquoi nous y mettons davantage de ressources humaines. La charge de travail du développement des architectures sur Android et sur PC n'est pas à négliger, mais requerra moins de ressources en temps et en personnes.

Rôles de gestion et de soutien

- Scrum master et responsable des services Redmine (Francis Granger)
- Responsable de l'interface et de l'expérience utilisateur (UI/UX) (Rose Hirigoyen)
- Responsable de l'intégration et du développement continu (CI/CD) (Gabriel Pollo)
- Responsable de la qualité du code (Charles Marois)
- Responsable en habiletés personnelles et relationnelles (HPR) (Bernard Meunier)

Afin d'assurer la qualité du projet, nous proposons également de dédier des rôles spécifiques de gestion et de soutien. La responsable du UI/UX veillera au bon développement de la présentation visuelle des applications Android et PC et de l'expérience des usagers. Le responsable du CI/CD s'assurera de développer des protocoles d'automatisation du code pour le déploiement et les tests du code. Le responsable des services Redmine et Scrum Master veillera au bon déroulement du projet et au respect des échéanciers et tâches. Il s'assurera également de la bonne tenue des rencontres de travail. Le responsable en HPR veillera à la bonne communication interne et aux relations entre les développeurs, afin d'éviter tout conflit potentiel au cours des mois de développement. Il aidera aussi à pallier tout autre souci de gestion ou technique. Enfin, le responsable en contrôle de qualité du code supervisera la qualité du travail technique des autres membres et instaura des règles claires d'assurance qualité à suivre lors des périodes de développement.

2.2 Entente contractuelle

Pour la réalisation de ce projet, nous suggérons une entente contractuelle à coût fixe, puisque nous nous engageons à livrer un produit final clé en main selon les exigences de Polytechnique. Les spécifications du projet sont clairement énoncées et ne risquent pas de changer en cours de route. Le paiement final est attendu lors de la livraison du produit final et de son acceptation par le client.

Ce contrat stipulera les coûts totaux estimés pour la réalisation du registre distribué, soit le salaire de 5 ingénieurs en informatique pour 480 heures-personne.

Nous nous engageons à livrer 3 étapes du projet au contractant. Un prototype sera proposé au client dès le jeudi 26 septembre 2019 et comprendra une communication de base entre l'application mobile Android et le serveur web central, et entre le serveur web central et un mineur. Un livrable intermédiaire sera proposé le jeudi 7 novembre avec davantage de fonctionnalités dont une application mobile complète disponible en mode usager et une application PC disponible en mode administrateur affichant l'historique des événements. Ce livrable contiendra aussi l'ensemble du contrôle, de la gestion et de la

communication avec la chaîne de calcul de blocs via le serveur web central. Enfin, le produit final sera livré le jeudi 28 novembre 2019. Ce produit sera entièrement portable et comprendra toutes les fonctionnalités requises. Un contrat fixe est donc le plus approprié pour le produit demandé.

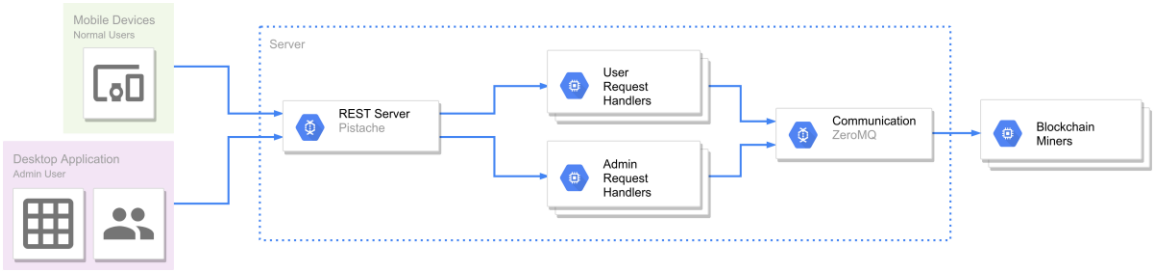
3. Solution proposée

3.1 Architecture logicielle sur serveur (Q4.2)

Le serveur a comme rôle principal d'être une interface entre les utilisateurs et les données enregistrées dans la chaîne de bloc.

Dans les requis, il était demandé d'interfacer avec l'application mobile et l'application bureau à l'aide d'un API REST. Il existe une grande panoplie de librairies C++ actives permettant d'établir un serveur HTTP pour traiter les requêtes tel que *pistache* ou *cpprestsdk.* Nous avons décidé d'utiliser pistache, car elle offre une grande quantité d'exemples, dont un serveur REST, et permet aussi de traiter les requêtes par HTTPS.

Afin de communiquer avec les mineurs, on utilise la librairie *ZeroMQ*. Celle-ci nous permet d'abstraire plusieurs types de transactions de systèmes distribués. Dans notre cas, le serveur et le mineur doivent pouvoir facilement communiquer entre eux efficacement afin de se synchroniser. Il existe aussi des alternatives comme *RabbitMQ* ou *ActiveMQ*, mais elles sont plus volumineuses et offrent beaucoup de fonctionnalités que l'on ne compte pas utiliser. Dans tous les cas, il est facile d'utiliser ces méthodes de transport avec une variété de langages dans le cas où l'on voudrait interfacer avec d'autres systèmes plus tard.

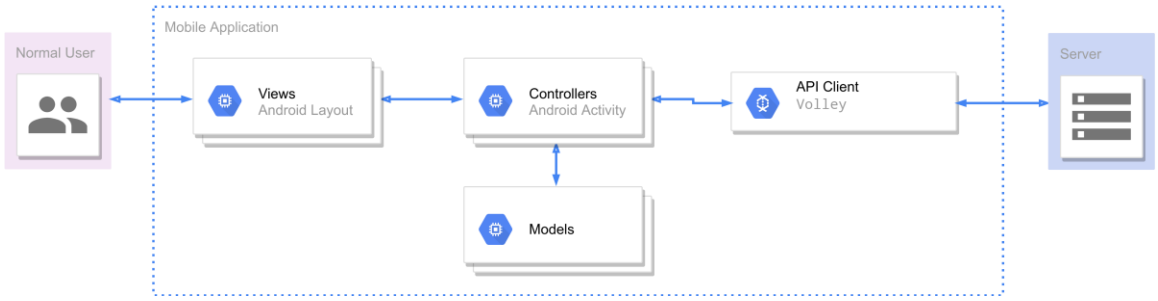


3.2 Architecture logicielle sur tablette (Q4.3)

L'application mobile a pour rôle principal de permettre à un utilisateur de consulter et modifier certaines données accessibles par le serveur. En tant qu'application graphique, elle utilisera une architecture modèle-vue-contrôleur afin de bien dissocier les données de l'interface.

Dans le cas du développement d'une application Android, les vues sont décrites avec les *Layout* Android. Il est facile de modifier une vue sans avoir à changer ou écrire une ligne de Java/Kotlin. On peut tout simplement utiliser une interface utilisateur. Lorsqu'une action est effectuée sur les vues, un contrôleur est appelé, et celui-ci est en charge de gérer la requête, de modifier des données et d'actualiser la vue de l'utilisateur.

Dans une grande majorité d'actions, le contrôleur devra faire des requêtes au serveur REST. Pour ce faire, on utilisera la librairie *Volley* qui est le standard pour construire des requêtes HTTP(S) sur Android. Par conséquent, il y a une grande quantité de documentation et d'exemples afin de nous aider dans le développement. La figure suivante résume l'architecture de l'application mobile.

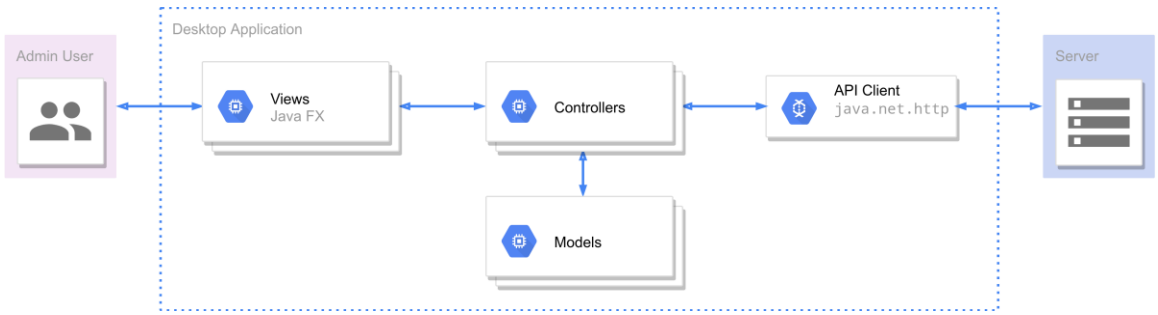


3.3 Architecture logicielle de l'application sur PC

L'application de bureau a pour rôle principal de permettre à un administrateur de gérer les comptes utilisateurs et éditeurs, ainsi que de voir des logs. Comme l'application Android, en tant qu'application graphique, elle utilisera une architecture modèle-vue-contrôleur.

En ce qui concerne les vues de l'application bureau, nous utiliserons le *JavaFX Scene Builder,* qui rend la création d'interfaces on ne peut plus simple. Il est en effet possible de simplement glisser et déposer des éléments afin de construire l'interface de notre choix. Comme pour l'application Android, chaque action sur une vue déclenche un contrôleur qui modifiera les données ou changera la vue, entre autres.

Dans une grande majorité d'actions, le contrôleur devra faire des requêtes au serveur REST. Pour ce faire, on utilisera la librairie de base *java.net.http*, introduite officiellement dans Java 11 après son incubation dans Java 9. Puisqu'elle fait partie du langage, il y a une documentation officielle exhaustive et beaucoup d'exemples. La figure suivante résume l'architecture de l'application de bureau.



4. Processus de gestion

4.1 Estimations des coûts du projet

Nous justifions les coûts d'allocation des services techniques selon le temps estimé fourni par les ressources humaines déployés et par l'achat de matériel informatique.

Cartes SD de 32Go pour mineurs	5cartes * 50$/carte = 250$
Salaire des développeurs-analystes	4dev. * 130$/h/dev * 9sem. * 8h/sem. = 37 440$
Salaire du chargé de projet	145$/h/dev * 9sem. * 8h/sem. = 10 440$
Total	48 130$

4.2 Planification des tâches (Q2.2 et Q11.2)

Livrable intermédiaire :

v1.0 - Semaine 1

Échéance dans 20 jours (03/10/2019)

0%

5 demandes (0 fermée — 5 ouvertes)

Demandes liées

- Fonctionnalités #4705: Création des points d'accès REST
- Fonctionnalités #4706: Créer une base de données serveur REST
- Fonctionnalités #4707: Addition d'un bloc sur un mineur
- Fonctionnalités #4708: Affichage simple de logs fictifs
- Fonctionnalités #4709: Affichage simple de résultats fictifs

v1.0 - Semaine 2

Échéance dans 27 jours (10/10/2019)

0%

5 demandes (0 fermée — 5 ouvertes)

Demandes liées

- Fonctionnalités #4710: Créer une transaction simple serveur REST
- Fonctionnalités #4712: Expérience de login fictive sur Android
- Fonctionnalités #4714: Expérience de transaction fictive sur Android
- Fonctionnalités #4715: Créer un bloc et le valider avec les 3 mineurs
- Fonctionnalités #4716: Système d'authentification pour usagers Android

v1.0 - Semaine 3

Échéance dans 34 jours (17/10/2019)

0%

5 demandes (0 fermée — 5 ouvertes)

Demandes liées

- Fonctionnalités #4711: Ajouter un fichier PDF serveur REST
- Fonctionnalités #4713: Expérience de login fictive sur PC
- Fonctionnalités #4717: Système d'authentification pour admin PC
- Fonctionnalités #4718: Création de transactions via Android
- Fonctionnalités #4719: Créer un bloc avec les données de transactions

v1.0 - Semaine 4

Échéance dans 41 jours (24/10/2019)

0%

5 demandes (0 fermée — 5 ouvertes)

Demandes liées

- Fonctionnalités #4720: Système de logs pour le serveur REST
- Fonctionnalités #4721: Implémenter les GET serveur REST
- Fonctionnalités #4722: Expérience fictive des GET sur Android
- Fonctionnalités #4723: Déposer un fichier PDF via Android
- Fonctionnalités #4724: Créer une base de données mineurs

v1.0 - Semaine 5

Échéance dans 48 jours (31/10/2019)

0%

5 demandes (0 fermée — 5 ouvertes)

Demandes liées

- Fonctionnalités #4725: Affichage de logs PC
- Fonctionnalités #4726: Fin du mode usager sur Android
- Fonctionnalités #4727: Intégration finale entre le serveur web et les mineurs fonctio
- Fonctionnalités #4728: Complétion des logs sur la console pour les mineurs
- Fonctionnalités #4729: Fin de la vue des logs sur PC

v1.0 - Semaine 6

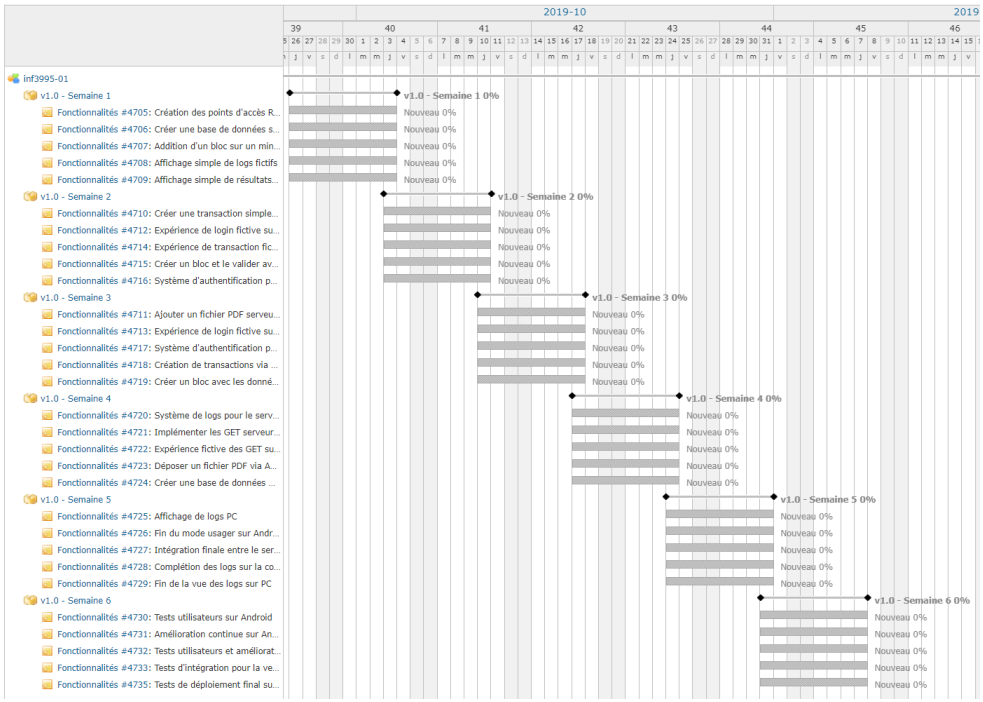
Échéance dans 55 jours (07/11/2019)

0%

5 demandes (0 fermée — 5 ouvertes)

Demandes liées

- Fonctionnalités #4730: Tests utilisateurs sur Android
- Fonctionnalités #4731: Amélioration continue sur Android
- Fonctionnalités #4732: Tests utilisateurs et amélioration continu sur PC
- Fonctionnalités #4733: Tests d'intégration pour la version 1.0
- Fonctionnalités #4735: Tests de déploiement final sur la carte



Livrable final :

v2.0 - Semaine 7

Échéance dans environ 2 mois (14/11/2019)

0%

5 demandes (0 fermée — 5 ouvertes)

Demandes liées

- Fonctionnalités #4736: Amorçage avec systemd serveur REST
- Fonctionnalités #4737: Amorçage avec systemd mineurs
- Fonctionnalités #4738: Comptes avec édition ou consultation serveur REST
- Fonctionnalités #4739: Système de logs pour les mineurs
- Fonctionnalités #4740: Le mode consultation seulement PC

v2.0 - Semaine 8

Échéance dans environ 2 mois (21/11/2019)

0%

5 demandes (0 fermée — 5 ouvertes)

Demandes liées

- Fonctionnalités #4741: Afficher les logs mineurs sur PC
- Fonctionnalités #4742: Création et suppression de compte PC
- Fonctionnalités #4743: Gèrer la perte d'un ou plusieurs mineurs serveur REST
- Fonctionnalités #4744: Gèrer la perte d'un ou plusieurs mineurs pour les mineurs
- Fonctionnalités #4746: Finir tout ce qui est en lien avec des comptes

v2.0 - Semaine 9

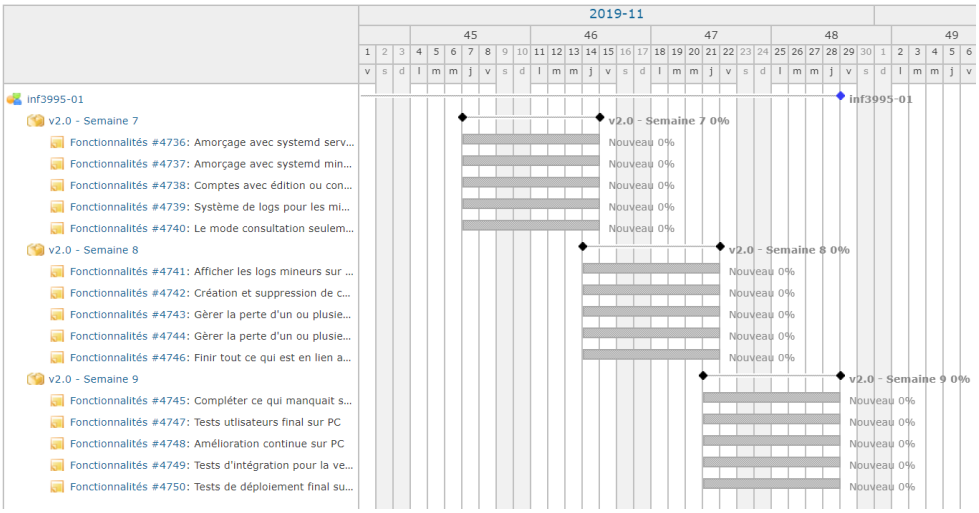
Échéance dans environ 3 mois (28/11/2019)

0%

5 demandes (0 fermée — 5 ouvertes)

Demandes liées

- Fonctionnalités #4745: Compléter ce qui manquait sur Android (si le cas)
- Fonctionnalités #4747: Tests utlisateurs final sur PC
- Fonctionnalités #4748: Amélioration continue sur PC
- Fonctionnalités #4749: Tests d'intégration pour la version 2.0
- Fonctionnalités #4750: Tests de déploiement final sur les cartes



4.3 Calendrier de projet (Q3.3)

Livrable intermédiaire, v1.0 : 7 novembre	Semaine 1 : 3 octobre
	Semaine 2 : 10 octobre
	Semaine 3 : 17 octobre
	Semaine 4 : 24 octobre
	Semaine 5 : 31 octobre
	Semaine 6 : 7 novembre
Livrable final, v2.0 : 28 novembre	Semaine 7 : 14 novembre
	Semaine 8 : 21 novembre
	Semaine 9 : 28 novembre

4.4 Ressources humaines du projet

4 développeurs-analystes :

- Charles Marois : Il à trois stages à son actif dont un en France où il a travaillé sur une application mobile, un serveur GraphQL/Elixir et un système embarqué pour véhicule.
- Bernard Meunier : Il a complété deux stages dans le domaine bancaire dans des compagnies de différentes grosseurs. Ses expériences sont principalement dans la création et manipulation de base de données et l'utilisation de python et de C#.
- Rose Hirigoyen : Elle à quatre stages à son actif, trois stages complétés chez Microsoft. Un de ceux-ci était dans l'accélérateur de projet, le Garage. On note beaucoup d'expérience avec le C++ et la sécurité informatique.
- Gabriel Guilbert : Il a déjà été impliqué dans Poly Orbite et est maintenant très impliqué dans la formule Poly eRacing. Il a une excellente

connaissance des systèmes embarqués en plus d'avoir de bonnes bases en C. Il est aussi sous Arch Linux sur son PC personnel.

1 chargé de projet :

- Francis Granger : Il a aussi quatre stages à son actif, dont deux chez Microsoft et un chez Google. Il a aussi fait partie de l'accélérateur du Garage où il était le leader de son équipe. Il a une bonne expérience en serveurs REST ainsi qu'en C++.

5. Suivi de projet et contrôle

5.1 Contrôle de la qualité

Tout au long de ce projet, nous voulons nous assurer de la qualité des livrables remis via un processus de révision. Puisqu'il y a trois livrables, chacun plus important que l'autre, il sera important de partir sur une bonne base au fur et à mesure que le projet prend forme.

Pour le prototype de base, nous souhaiterons assurer la qualité de chaque application individuellement, soit l'application Android, le serveur web, l'application sur PC et le mineur. Cependant, puisque c'est à cette étape que nous établirons la base du projet, nous voudrons aussi s'assurer que chaque application est bien intégrée avec les autres. Tout d'abord, nous mettrons un place un pipeline de développement à l'aide de GitLab, qui nous permettra de rouler des tests unitaires automatiquement chaque fois que nous souhaitons ajouter du code au projet. Le code sera également révisé par deux membres de l'équipe. Nous fonctionnerons avec des branches de développement afin de protéger l'intégrité du projet de base, ce qui assurera la qualité individuelle de chaque partie du projet. Nous écrirons également des tests d'intégration, afin de s'assurer que les changements réalisés dans une partie du projet n'ont pas d'impact sur les autres et que la communication se fait bien entre chaque morceau du projet. Finalement, nous ferons aussi des tests de déploiement, afin de s'assurer que le projet fonctionne avec le matériel informatique utilisé. Le plus important dans cette étape sera d'établir une base solide qui nous permettra de tester le projet en profondeur et d'attraper les erreurs à chaque étape.

Pour le livrable intermédiaire, nous souhaiterons encore une fois assurer la qualité de chaque application individuellement. Plus spécifiquement, pour le serveur web, nous devrons nous assurer du fonctionnement de l'authentification dépendant du type de compte, ainsi que de la communication du serveur avec les mineurs. Pour les mineurs, nous voudrons tester la synchronisation entre eux, ainsi que de s'assurer qu'ils génèrent des logs. Pour l'application Android, nous voudrons tester la communication avec le serveur et le bon fonctionnement du compte éditeur, ainsi que de tester toutes les fonctionnalités requises avec des tests unitaires. Ce sera aussi le bon moment pour s'assurer que l'interface est

intuitive à l'aide de tests avec des utilisateurs. Pour l'application PC l'écriture de tests unitaires et d'intégration nous permettrons de tester toutes les fonctionnalités que nous devons ajouter. À cette étape, il sera important de s'assurer que les applications communiquent bien ensemble et que nous finissons toutes les fonctionnalités requises avant de continuer à la prochaine étape. Il sera important de tester en profondeur afin d'éviter toute mauvaise surprise durant le sprint final.

Pour le livrable final nous voudrons non seulement nous assurer de la qualité de chaque partie du projet, mais aussi de la bonne communication entre chaque application, ainsi que de la robustesse du système. En effet, nous souhaitons obtenir un produit final agréable à utiliser et intuitif, il sera donc important de s'assurer que tout reste fonctionnel en cas d'erreur dans une des applications, à l'aide de tests d'intégration et de tests de déploiement sur les FPGAs. Il serait souhaitable de réaliser un *bug bash* au début de cette étape afin que le produit remis ait été testé en profondeur. Il sera important d'écrire plus de tests unitaires afin de viser une couverture maximale, ainsi que de déléguer une personne qui s'assurera de la bonne intégration de toutes les branches alors que nous ajoutons toujours plus de code. Bien sûr, il faut que tous les membres de l'équipe prennent la responsabilité de maintenir la qualité de la branche principale, mais à cause de la quantité de fonctionnalités à ajouter durant ce dernier sprint, il sera crucial de s'assurer que tout est intégré de la bonne manière. Il sera finalement important de réaliser des tests sur la base de données afin de s'assurer de l'intégrité des données. Le plus important dans cette étape sera de remettre un produit final poli, robuste et utilisable.

5.2 Gestion de risque (Q2.6 et 11.3)

En ce qui concerne la partie serveur, l'un des principaux risques est que la communication se fasse mal avec les mineurs et avec les autres applications. C'est un risque important, car si nous n'arrivons pas à mettre en place un serveur robuste, notre application sera inutilisable. Il faut également s'assurer d'avoir une bonne structure dans notre API. Heureusement, les requis sont très clairs quant aux routes à implémenter, ce qui devrait nous éviter d'avoir à faire des changements majeurs en cours de projet.

Pour l'application Android, le principal risque est qu'elle ne soit pas intuitive ou agréable à utiliser. Bien que la partie design ne soit pas la plus importante, l'utilisabilité de l'application est cruciale. Afin d'éviter d'avoir des mauvaises surprises, il sera important que tous les membres testent l'application. Bien sûr, il est possible que des changements dans l'interface surviennent en cours de route, mais une bonne boucle de rétroaction évitera qu'ils soient trop grands.

Les mineurs représentent probablement la partie la plus à risque, puisque c'est la technologie avec laquelle l'équipe est la moins familière. En effet, il est plus difficile d'évaluer le temps requis pour réaliser cette portion du projet,

comment la tester et comment s'assurer que tout fonctionne de manière robuste. C'est la partie dans laquelle il y aura probablement le plus d'erreurs inattendues, ce qui signifie qu'il sera important de prévoir assez de temps pour chaque tâche.

5.3 Tests (Q4.4)

Pour le serveur web, il faudra s'assurer que chaque requête retourne l'information attendue. Il faudra également s'assurer qu'il répond rapidement aux requêtes. Il faudra qu'il gère bien les requêtes malformées ou non autorisées en retournant le bon code d'erreur, le tout à l'aide de tests unitaires et d'une fausse base de données.

Pour l'application Android, il faudra s'assurer que les bonnes informations sont présentées au bon moment et que l'application gère bien les erreurs à l'aide de tests unitaires. Il sera aussi utile de faire un peu de tests avec les utilisateurs, afin de valider le flot de l'application.

Pour l'application PC, il sera important de s'assurer que l'information présentée est la bonne. Pour ce faire, nous aurons besoin de tests d'intégration, puisque nous voulons voir si l'information se rend bien des mineurs au serveur à l'application PC. Il sera également important de réaliser des tests avec les utilisateurs afin de confirmer que l'application est intuitive.

Pour les mineurs, il faudra écrire beaucoup de tests unitaires afin de vérifier chaque aspect des transactions. Il sera utile mais pas obligatoire de tester la performance des mineurs afin de s'assurer de la fluidité de l'utilisation de l'application.

En général, il sera très utile de réaliser un *bug bash,* c'est-à-dire d'essayer de briser notre propre application en testant les limites de ses fonctionnalités avant la remise. Tout cela nous permettra d'assurer la qualité du projet tout au long de son développement.

5.4 Gestion de configuration

Nous utiliserons GitLab comme système de contrôle de version. Nous aurons une branche master, qui contiendra tout le code stable. Cette branche ne devra jamais être instable, c'est-à-dire que tout ce que nous y ajouterons devra avoir été testé et approuvé. En bref, tous les tests doivent toujours passer sur cette branche. Lorsque nous voudrons ajouter du code, il faudra créer une branche à partir de la branche master, et dont le nom suivra la convention suivante: nom/nom-de-la-branche. Nous mettrons en place un système qui vérifiera notre code localement avant que nous l'ajoutions au dépôt distant. Il vérifiera la syntaxe du code ainsi que le format, afin que tout soit uniforme. Une fois le code ajouté au dépôt, il devra

passer les tests définis et être approuvé par deux membres de l'équipe avant d'être ajouté à la branche master. Cela nous permettra d'assurer en tout temps la stabilité de la branche master.

6. Références (Q3.2)

[1] J. Collin. (2019) Demande de proposition no. A2019-INF3995. [En ligne]
[2] J. Collin. (2019) Exigences techniques pour la conception d'une chaîne de blocs pour dossiers d'étudiants. [En ligne]