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
A2020-INF3995 du département GIGL

***Conception d'un système* de prédiction de données sur l'usage du BIXI**

Équipe No 6

Anastasiya Basanets (193929)
Samuel Charbonneau (1897953)
Jean-Olivier Dalphond (1873653)
Olivier Dion (1927844)
Mariam Sarwat (1928777)

24 Septembre 2020

Table des matières

1. Vue d'ensemble du projet

1.1 But du projet, portée et objectifs

Le développement de l'intelligence artificielle (IA) apporte de plus en plus d'outils qui permettent à des programmeurs, même moins expérimentés, de concevoir des logiciels à l'aide de cette technologie. Dans cette optique, l'entreprise derrière le système de vélo libre-service BIXI de la Ville de Montréal aimerait pouvoir bénéficier de cet avancement : on cherche à fournir une solution qui permettrait de consulter les données BIXI et de prévoir plus efficacement l'utilisation éventuelle des vélos. Étant donné que BIXI est en service depuis quelques années, le volume des données recueillies est considérable, et elles devraient permettre de fournir certaines prédictions. Le présent projet a pour but de développer un système informatique complet pouvant afficher de l'information pertinente aux utilisateurs et aux administrateurs, mais également prédire certaines données.

Cette plateforme interactive comprendra une application Android disponible pour tous. Cette dernière contiendra la présentation de données associées au système de vélo en libre-service BIXI ainsi que les prédictions faites à partir de ces données. Le système comprendra un serveur d'accessibilité des données. De plus, d'autres logiciels pour accélérer l'analyse et la prédiction des données, décrites par l'appellation « engins de données », seront mis sur pied et seront développés en Python. Ce langage de programmation est d'ailleurs très répandu dans le domaine de l'intelligence artificielle. Finalement, un léger logiciel administratif devra être conçu afin de permettre à un administrateur de voir les allées et venues des utilisateurs de l'application mobile et d'assurer la disponibilité de l'ensemble du système.

Afin de faciliter le portage des engins de données et du serveur, il sera nécessaire d'utiliser des outils « conteneurs ». La technologie des conteneurs Docker sera notre outil de prédilection, avec Docker Compose pour arrimer ces composantes entre elles. Cela permettra également de limiter les interactions entre les entités du système et ainsi renforcer la sécurité.

1.2 Hypothèses et contraintes

Comme mentionné dans la section précédente, le projet, soit la conception d'un système informatique de prédiction de données sur l'usage du BIXI, est composé de plusieurs parties. Effectivement, elle repose sur la communication client-serveur, où le client est représenté une application Android et une application PC. Les engins de données, quant à eux, feront une analyse en temps réel des données, de même qu'un auto-apprentissage en utilisant l'intelligence artificielle. Dans la section suivante, les

composantes du système seront détaillées, et les hypothèses et les contraintes qui leur sont associées seront mentionnées.

La première composante du système informatique correspond au serveur web central. Ce serveur, roulant sur Linux, devra répondre aux requêtes qui lui sont envoyées par les applications PC et Android. De plus, il doit pouvoir communiquer avec les engins de données et surveiller leur service. Pour répondre à ces besoins, trois volets sont à prendre en compte : la sécurité, les communications interprocessus et la gestion du stockage des données. En effet, bien que la sécurité soit gérée par un proxy qui redirige les requêtes déchiffrées vers le serveur, celui-ci est néanmoins capable de faire le chiffrement et le déchiffrement en utilisant le protocole TLS au besoin. Au niveau des communications interprocessus, ces dernières sont entièrement faites à l'aide du protocole HTTP et d'une interface REST. Ensuite, la gestion des données sera faite à l'aide d'une arborescence de fichiers, étant donné sa simplicité. Enfin, puisque le serveur web ne doit jamais tomber hors service, ce dernier sera conçu de façon à être tolérant le plus possible aux pannes.

Les engins de données constituent la deuxième partie du projet. Trois engins seront construits afin de simplifier le fonctionnement du système. Le premier s'occupera de la recherche et de l'information reliée aux stations BIXI. Le deuxième transmettra les données récoltées via Kaggle et les utilisera pour faire des prédictions. Le dernier, quant à lui, exécutera l'algorithme de prédiction, soit celui des forêts aléatoires qui utilise l'intelligence artificielle. Tous ces engins seront codés en Python 3 et seront stockés avec Docker. Cependant, pour la plupart des membres de l'équipe, l'apprentissage du langage Python, de l'outil Docker et Docker Compose ainsi que des bases de l'IA sera nécessaire afin d'assurer la réalisation des tâches. De plus, les données existant sur Kaggle seront stockées sur une base de données avec la technologie SQLite, qui est plus légère que MySQL ou PostgreSQL, par exemple, mais qui convient parfaitement aux besoins du système.

La troisième partie du projet concerne l'application Android qui permet aux usagers de consulter les données du système. Effectivement, cette application contient des graphiques qui affichent les prédictions fournies par le serveur et les engins de données. De plus, les stations seront affichées sur une carte de Montréal fournie par Google Maps et des sondages seront effectués auprès des utilisateurs. Il est important de concevoir un design d'application qui soit intuitif et simple à utiliser. Cette application, suivant les couleurs de BIXI (rouge et blanc), va respecter le modèle MVVM qui permet de faire un lien direct entre les données et la vue, et des services pour gérer les différentes fonctionnalités. Au niveau technique, le développement mobile de la plateforme représentera un défi pour l'équipe. En effet, personne dans l'équipe n'a déjà construit une application Android à l'aide du langage Kotlin. De plus, l'interaction avec Google Maps peut s'avérer complexe en termes

de programmation. Cependant, l'expérience de l'équipe en programmation web avec Angular permettra l'apprentissage rapide de ces nouvelles connaissances, car Google a l'habitude de bien organiser l'interaction entre ses différents services et ses différentes technologies. Cela devrait faciliter l'intégration de la composante Google Maps pour l'application.

La dernière partie, quant à elle, est l'application PC qui servira à l'administrateur du système. Elle permettra de changer le mot de passe du système, présenter les résultats du sondage et voir les messages du journal des trois engins de données. Cette application sera disponible sur Linux et Windows. Elle sera conçue à l'aide du langage Python 3 et de PyQt5, qui est une nouveauté pour la plupart des membres de l'équipe. Il faudra donc allouer du temps pour l'apprentissage de ces technologies. Cependant, l'expertise en Python d'un membre de l'équipe permet aux autres d'avoir recours à un support rapide spécifique au projet.

Il est crucial de mettre en place une architecture cohérente et solide. Effectivement, cette considération est importante pour assurer une interaction logique entre les différents composants du système. Nous détaillerons l'architecture générale du système, ainsi que celles des composants dans une section dédiée du présent document. Il faut prévoir du temps en début de projet pour penser à une bonne architecture afin de bien planifier le travail et surtout éviter d'avoir à restructurer de grandes portions de code.

Bien sûr, le contexte particulier de Covid-19 pourrait freiner la mise sur pied d'un tel projet, notamment par le retard d'accès au matériel (tablette Android), et par l'impossibilité de réunir l'équipe sur une base régulière. Toutefois, une bonne organisation des tâches et du temps et une grande disponibilité de toute l'équipe permettront un travail à distance productif. Notre équipe a l'habitude de synchroniser les efforts de chacun à distance, donc nous sommes confiants que cette situation ne freinera pas la réalisation du projet de façon significative.

1.3 Biens livrables du projet

Comme mentionné précédemment, le système informatique sera composé d'une application mobile, d'un serveur REST, de trois engins de données et d'une application PC. Des conteneurs seront ajoutés pour limiter les accès aux composantes du système par d'autres pour les engins de données. Par ailleurs, la température sera utilisée comme élément de base pour développer une intelligence artificielle capable de prédire des données dans le futur. Ainsi, une fois l'expertise acquise, les données de BIXI pourront être utilisées pour permettre la prédiction de certaines données dans le futur.

Le 24 septembre 2020 sera la date de présentation du prototype du système informatique. Ce prototype sera composé d'un serveur web REST dans un conteneur Docker, une application mobile de base pour Android et un premier engin de données. L'application mobile doit être en mesure de faire une communication avec le serveur REST. À ce stade, un simple message de succès suffira. L'engin de données est composé simplement de l'implémentation de l'algorithme de forêt aléatoire qui sera utilisé pour la prédiction de température. Pour le moment, dans le prototype, il n'y aura pas de communication entre le serveur REST et l'engin de données.

Le 29 octobre 2020, un livrable intermédiaire sera transmis. Dans ce livrable, le serveur se verra ajouter la gestion du sondage qui sera disponible sur l'application mobile. Les engins de données devront être dans des conteneurs Docker distincts, permettre le retour des résultats à l'application mobile et produire des messages « logs » en sortie standard locale. L'application Android aura donc l'ajout de l'affichage des résultats de prédiction ainsi que l'état de fonctionnement des engins de données, en plus du sondage. Finalement, pour ce livrable, une application PC sera ajoutée au projet; celle-ci permettra à un administrateur de consulter les résultats des sondages mobiles. De plus, une interface usager permettant la consultation des logs des engins de données sera disponible, mais sans la transmission de ces derniers depuis le serveur.

En dernier lieu, le 1er décembre 2020 sera la date du livrable final. Le serveur web pourra vérifier l'état des engins de données. Ces engins auront maintenant la fonctionnalité de recherche des stations BIXI et les prédictions seront raffinées. De plus, Docker Compose sera mis en place pour permettre la communication entre le serveur web et les trois engins de données, et leur déploiement sur toute machine qui détient Docker. Le visuel de l'application mobile sera perfectionné et les correctifs suite aux commentaires provenant du livrable intermédiaire seront ajoutés. L'application PC, quant à elle, se verra ajouter la réception des logs des engins de données et le changement de mot de passe du compte administrateur.

2. Organisation du projet

2.1 Structure d'organisation

L'équipe de projet sera composée de cinq professionnels : quatre développeurs-analystes et un coordonnateur de projet. Afin de permettre une communication directe et constante entre les membres, un réseau de communication décentralisé sera employé au sein de l'équipe. Effectivement, pour bien assembler les différentes composantes du projet, des échanges réguliers sont nécessaires entre les membres de l'équipe. Ce type d'organisation permet à chaque membre de travailler seul sur ses propres tâches tout en ayant la possibilité de communiquer avec autrui, et de suivre l'avancement de celles des autres membres.

Les différentes parties du projet seront séparées parmi les membres en fonction des connaissances de chacun, afin d'assurer l'efficacité de l'équipe. L'objectif est de mettre à profit, autant que possible, les forces de chacun. Olivier, détenant de bonnes connaissances sur les serveurs et la programmation en C, s'occupera de la conception et la réalisation du serveur web. De plus, il sera en charge de la configuration du système de contrôle de version, soit le Git. Jean-Olivier, avec l'aide de Samuel qui possède de l'expérience dans la conception d'application mobile, s'occupera de la mise en place de l'application Android, en plus de la configuration des conteneurs Docker. Finalement, Mariam, coordonnatrice de projet, et Anastasiya, voulant relever des défis, s'occuperont de l'application PC. Les engins de données seront développés par Anastasiya, dans le cas de l'engin de prédiction, et Samuel, pour les engins de gestion de la carte, et des données statistiques. Bien que certains membres travaillent avec certaines technologies pour la première fois, ils ont tous derrière eux plusieurs projets logiciels et informatiques. Ce n'est donc pas leur premier projet d'envergure et ils connaissent le déroulement de ce genre de travail.

Chaque membre contribuant au projet affichera ses travaux sur la plateforme Redmine. Chaque modification devra être révisée par un autre membre de l'équipe avant que ces derniers soient acceptés sur la branche principale, soit le « master », assurant ainsi la qualité du logiciel et limitant la possibilité de régression ou de bris de fonctionnalité. La qualité du code est une priorité pour notre équipe.

Sur un plan plus informel, nous utiliserons le logiciel Discord pour nos réunions, et pour le partage de documents importants. Un serveur personnalisé et accessible seulement aux membres de l'équipe a été créé. Il inclut un salon de discussion pour les conversations orales et les réunions, puis des canaux écrits pour l'envoi de documents, liens, informations ou autres.

2.2 Entente contractuelle

L'entente contractuelle que nous proposons est un contrat à terme. Bien que l'estimation du coût final ne soit pas forcément précise et définitive, le projet, en soi, n'est pas très long et ne devrait pas impliquer de grands excès de coûts. Pour les projets de nature informatique, il peut être difficile de prévoir le temps exact nécessaire à l'implémentation du système. Ce type de contrat fournit alors au promoteur ainsi qu'au contracteur plus de flexibilité, et réduit les coûts reliés aux spécifications. En effet, plusieurs imprévus peuvent survenir lors de la conception d'un tel projet. Un contrat à terme nous engage à livrer le produit à une date future selon les conditions prédéfinies du client, ce qui a pour effet de réduire les négociations. La conception des fonctionnalités n'aura pas de date précise, mais plutôt des dates de livraison estimées.

Ce type d'entente permet également de nous mettre plus rapidement au travail, ce qui est important considérant la durée réduite du projet. Bien qu'en conséquence les coûts ne soient pas parfaitement fixes, la courte durée du projet nous permettra de garder aisément le contrôle sur cet aspect. Nous souhaitons avancer rapidement dans la réalisation des composantes du système, de façon à nous garder le plus de temps possible pour assurer la qualité du produit que nous remettons. Par ailleurs, nous savons que ce type d'entente donne l'opportunité au client de choisir l'équipe qui présente la meilleure qualité de système, et nous sommes confiants d'être en mesure de présenter un projet fiable, innovateur et performant.

3. Solution proposée

La Figure 1 montre l'architecture globale du système. On constate que les clients sont les applications Android et le programme d'administration. Ces derniers communiquent à l'aide des protocoles HTTPS et TCP/IP, avec un serveur proxy qui est Traefik. Ce proxy redirige les requêtes soit vers les engins de données soit vers le serveur web. Tous les deux sont exécutés dans un environnement contrôlé à l'aide de Docker Compose. Enfin, les engins de données se partagent une base de données en SQlite, tandis que le serveur web utilise plutôt une arborescence de fichiers.

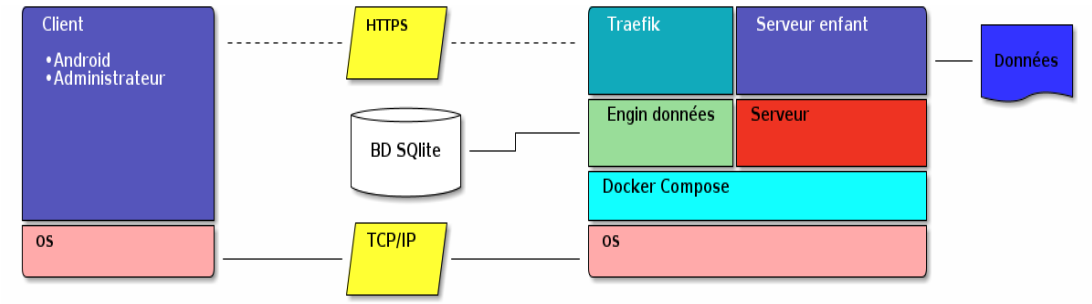

Figure 1 - Architecture globale du système ©Olivier Dion

3.1 Architecture logicielle sur serveur

Au commencement de son exécution, le serveur fera l'analyse grammaticale des options de la ligne de commandes. Ces options permettent, entre autres, de sélectionner le port d'utilisation du serveur (par défaut 443), de faire ou non le chiffrement avec TLS et de connecter le serveur à une adresse globale ou locale (loopback).

Par la suite, le serveur initialise la librairie OpenSSL et charge en mémoire le certificat autosigné qui sera utilisé pour chiffrer les futures communications, si l'option de chiffrement a été activée. Le serveur est donc

capable de faire par lui-même le chiffrement, dans le cas où l'on ne désire pas utiliser un proxy.

Ensuite, le serveur attend indéfiniment pour de nouvelles connexions. Pour chaque nouvelle connexion, le serveur délègue celle-ci à un processus enfant. Bien que cette solution ne soit pas très évolutive, elle permet d'isoler les connexions entre elles. Ainsi, si l'une d'entre elles cause un bogue, les autres connexions ne seront pas affectées.

Un processus enfant commence par faire la négociation TLS (*handshake*) si le serveur a été lancé avec le chiffrement activé. Puis, la requête HTTP est scannée à l'aide d'un scanneur lexical généré par l'utilitaire Flex. Cette requête est auditée pour s'assurer qu'elle soit valide. Puis, une recherche binaire est faite pour trouver la méthode à appeler pour la route dans la requête. Si cette route est inexistante, ou que cette dernière n'accepte pas la méthode demandée, une réponse de type BAD REQUEST est retournée au client. Sinon, la méthode est exécutée avec deux objets JSON. Le premier objet correspond au corps de la requête et le second correspond au corps de la réponse. Une fois la méthode terminée, la réponse est envoyée au client et la connexion est fermée, tuant ainsi le processus enfant. La Figure 2 résume l'exécution du serveur web.

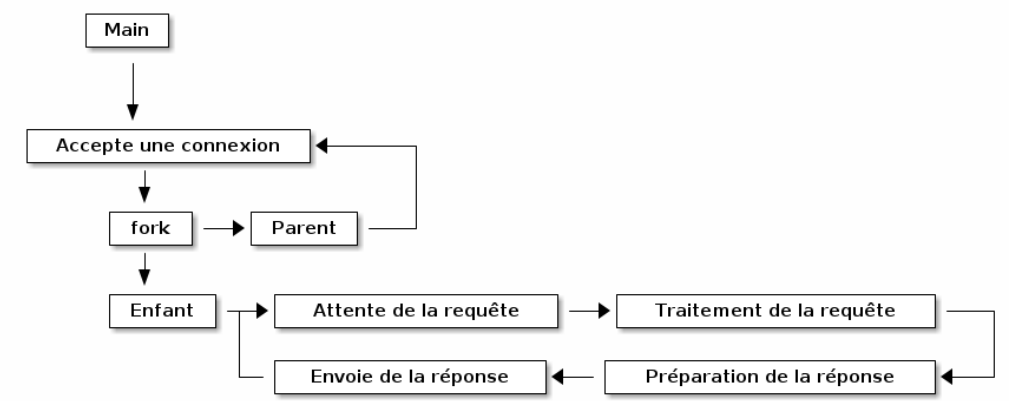

Figure 2 - État machine du serveur web ©Olivier Dion

Cela étant dit, une architecture alternative est proposée. Celle-ci n'est pas mise en place, étant légèrement plus susceptible à des bogues. De plus, le système doit être en marche en permanence, alors pour ces raisons, la première architecture est privilégiée. Cependant, cette seconde architecture pourrait être choisie dans le cas où la demande en requêtes au serveur surpasse sa capacité. La Figure 3 présente cette architecture secondaire. On constate qu'il s'agit d'une architecture en pipeline. Le gestionnaire de connexions, qui est le processus principal, se charge d'accepter les connexions et de mettre en place le chiffrement TLS au besoin. Puis, il délègue le travail de la lecture des requêtes à un fil d'exécution (*thread*). Ce

dernier, après avoir lu la requête, passe la main à un autre fil d'exécution qui en effectue le traitement. Le fil responsable de ce traitement peut accéder à la base de données sur le disque du système pour transférer de l'information. Par la suite, la charge de travail est déléguée au fil d'envoi des réponses, responsable d'envoyer la réponse au client. Enfin, la connexion est renvoyée à un fil d'exécution de lecture de requêtes, si la connexion est persistante. Dans la Figure 3, on remarque qu'il y a deux fils d'exécution à chaque niveau. Cela est pour illustrer que sur un nœud possédant plusieurs cœurs, il est possible de distribuer chaque thread par processeur.

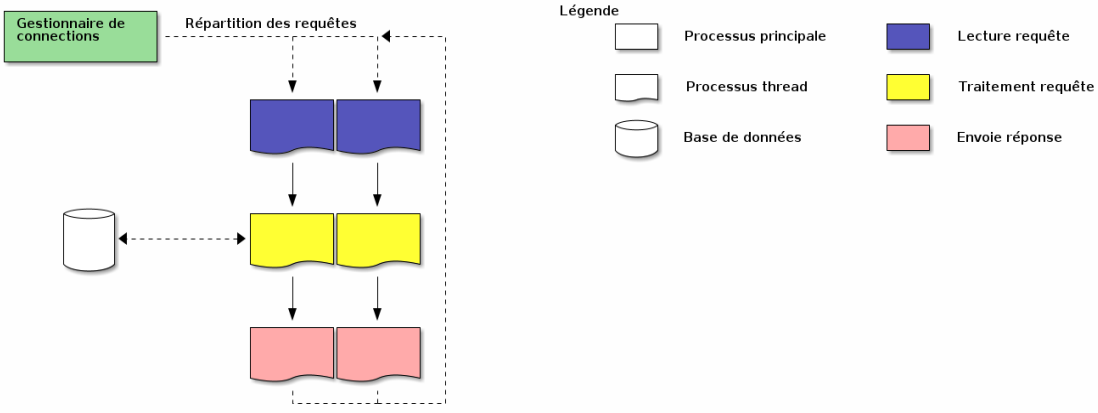

Figure 3 - Architecture secondaire du serveur web ©Olivier Dion

3.2 Architecture logicielle générale des engins de données

Les engins de données seront découplés pour simplifier le code et permettre la réutilisation des différents modules. À ce compte, nous aurons un module qui s'occupera de la gestion des requêtes HTTP. Il sera réutilisé par tous les engins de données. L'outil Traefik sera en charge de rediriger les requêtes entrantes vers le serveur qui, à son tour, s'adressera aux engins de données et le module HTTP s'occupera de traiter le tout. De plus, chacun des engins aura un processus principal qui fera la tâche demandée. Ils seront reliés à un module qui récupérera les données de BIXI. De plus, un module de journalisation et d'informations système sera aussi mis en place afin de conserver des traces des processus. Le troisième engin qui fera usage de l'intelligence artificielle aura à interagir avec un module qui génèrera des graphiques pour une présentation visuelle efficace et intéressante. Cela permettra de faire le suivi sur le processus d'apprentissage machine de l'engin de données. Chaque engin sera encapsulé dans un conteneur Docker. Nous orchestrerons la synchronisation des différents conteneurs avec la technologie Docker Compose. Finalement, pour gérer la communication entre les conteneurs du serveur web et des engins de données, nous utiliserons la fonction Linux *poll()*, puisque les

identificateurs de processus des conteneurs seront présents dans l'hôte Linux (dans le répertoire /proc). Si besoin est, une alternative serait de créer un dossier pour y placer les sockets Unix. La Figure 4 démontre l'exécution des engins de données.

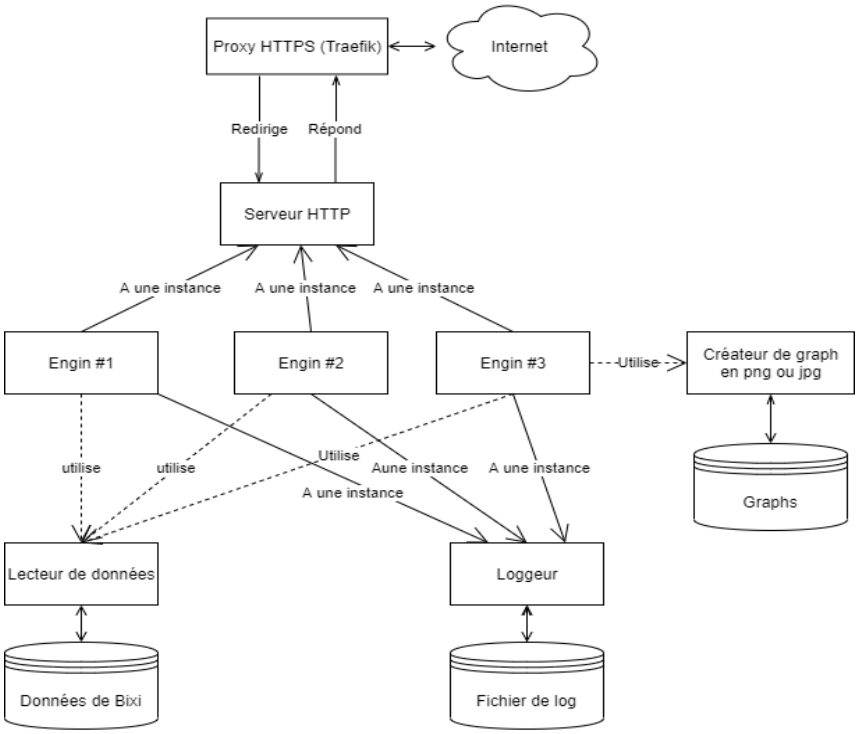

Figure 4 - Architecture des engins de données ©Samuel Charbonneau

3.3 Architecture logicielle sur tablette

Pour faire la conception de l'application mobile, nous allons suivre le modèle MVVM (Modèle – Vue – Vue-modèle). Cette architecture permettra d'associer les éléments des différentes vues directement aux variables correspondantes dans le modèle. Ceci augmentera la rapidité de transmission des données puisque les vues ne changeront pas seulement lors d'un évènement. [1] Ainsi, nous aurons des vues (nos activités), des Vues-modèles qui vont gérer et structurer les données à afficher, et finalement les modèles qui seront responsables de stocker les informations et faire les appels nécessaires aux services pour les requêtes. Nous aurons un service responsable des communications avec le serveur web, associé à un module Backend. Ce module ne sera présent qu'en une seule instance, de façon à gérer efficacement et sécuritairement les requêtes. Nous créerons également une interface pour faire le lien entre les autres services et celui-ci, ce qui aura pour effet notamment de faciliter les tests. Chaque activité associée aux engins de données se verra attribuer un service afin de bien encapsuler les rôles et respecter une cohérence dans l'architecture

globale de l'application. Celle-ci est présentée sur la figure suivante (Figure 5) :

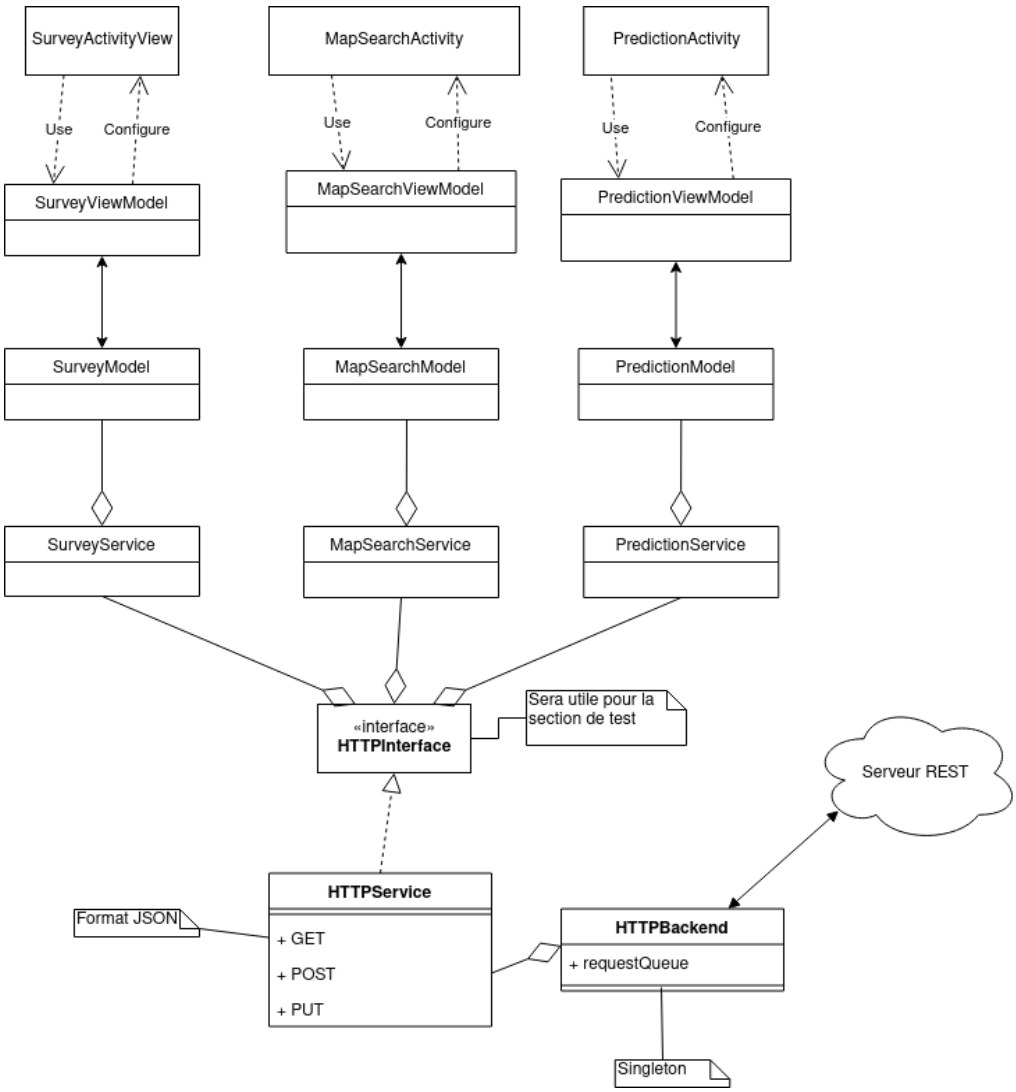

Figure 5 - Architecture de l'application Android ©Jean-Olivier Dalphond, Samuel Charbonneau

Afin d'améliorer l'expérience usager, la librairie Material Design sera utilisée afin de générer des graphiques conviviaux et visuellement attrayants. L'intégration de celle-ci se fera facilement, car nous avons choisi d'utiliser un produit de Google et nous savons que ceux-ci interagissent généralement bien entre eux. De plus, pour afficher de courts messages à l'usager lorsque ce sera pertinent, nous utiliserons la librairie Toast, qui s'intègre très bien à l'écosystème Android. Pour la présentation des graphiques, nous avons l'intention d'utiliser notamment la librairie MPAndroidChart, qui s'intègre aisément à Gradle, l'outil d'intégration et d'assemblage d'Android Studio. [2] Pour la communication avec le serveur web, nous utiliserons la librairie Volley de l'écosystème Android, qui est bien

documentée. D'ailleurs, nous planifions nous inspirer de certains concepts bien établis dans l'industrie, comme les patrons de conception Singleton et Controller pour centraliser les requêtes vers le serveur. [3]

3.4 Considérations logicielles de l'application sur PC

L'application PC, suivant un modèle MVC, sera conçu à l'aide du langage Python et PyQt un module libre qui permet de créer facilement des interfaces graphiques (en Python). Effectivement, comme le langage Python est déjà utilisé pour la conception des engins de données, il sera plus avantageux de l'utiliser aussi pour l'application PC. Ajoutant à cela, certains membres de l'équipe ont une longue expérience avec Python. De plus, l'équipe a déjà travaillé avec Qt dans le passé, donc l'apprentissage de PyQt se voit facilité. Dans l'ensemble, l'utilisation de Python et de PyQt est alors un bon choix, car nous connaissons assez bien la technologie et le temps passé à apprendre de nouvelles notions est minimisé.

Python est connu comme étant un langage facile à utiliser, notamment pour les débutants, en raison de sa lisibilité. [4] Il s'agit d'un langage gratuit et open source avec une multitude de tutoriels et de documentations présents en ligne. En ce qui concerne PyQt, il est connu ayant des bibliothèques standards étendues et un typage dynamique. [5] Ainsi, il n'est pas nécessaire de définir le type de données. De plus, il a un gestionnaire de packages et une grande collection de librairies libres d'utilisation (*open source*). Toutes ces raisons rendent le développement d'un tel projet plus rapide et plus simple.

4. Processus de gestion

4.1 Estimations des coûts du projet

Outre la planification des tâches, le temps total estimé pour ce projet est de 514 heures. Parmi ces heures, 86h sont attribuées à notre coordinatrice de projet, Mariam, et 428h sont attribuées aux quatre développeurs-analystes. Afin de prévenir le manque de temps en cas d'imprévus, nous attribuons un 10% de marge de manœuvre. Ainsi, 94,6h sont assignées au coordinateur et 470,8h au restant de l'équipe pour un total de 565.4 heures. Étant donné qu'il y a quatre développeurs-analystes à un taux de 130$/h et un coordonnateur de projet à 145$/h, le budget salarial est de 74 921$. De plus, il faut considérer l'utilisation d'une tablette Android au coût de 350$. Le budget total planifié pour le projet est donc de 75 271$. Il faut prendre en compte que ce total est seulement une estimation.

4.2 Planification des tâches

La figure suivante (Figure 6), démontre le diagramme de Gantt qui spécifie la planification des différentes tâches pour ce projet.

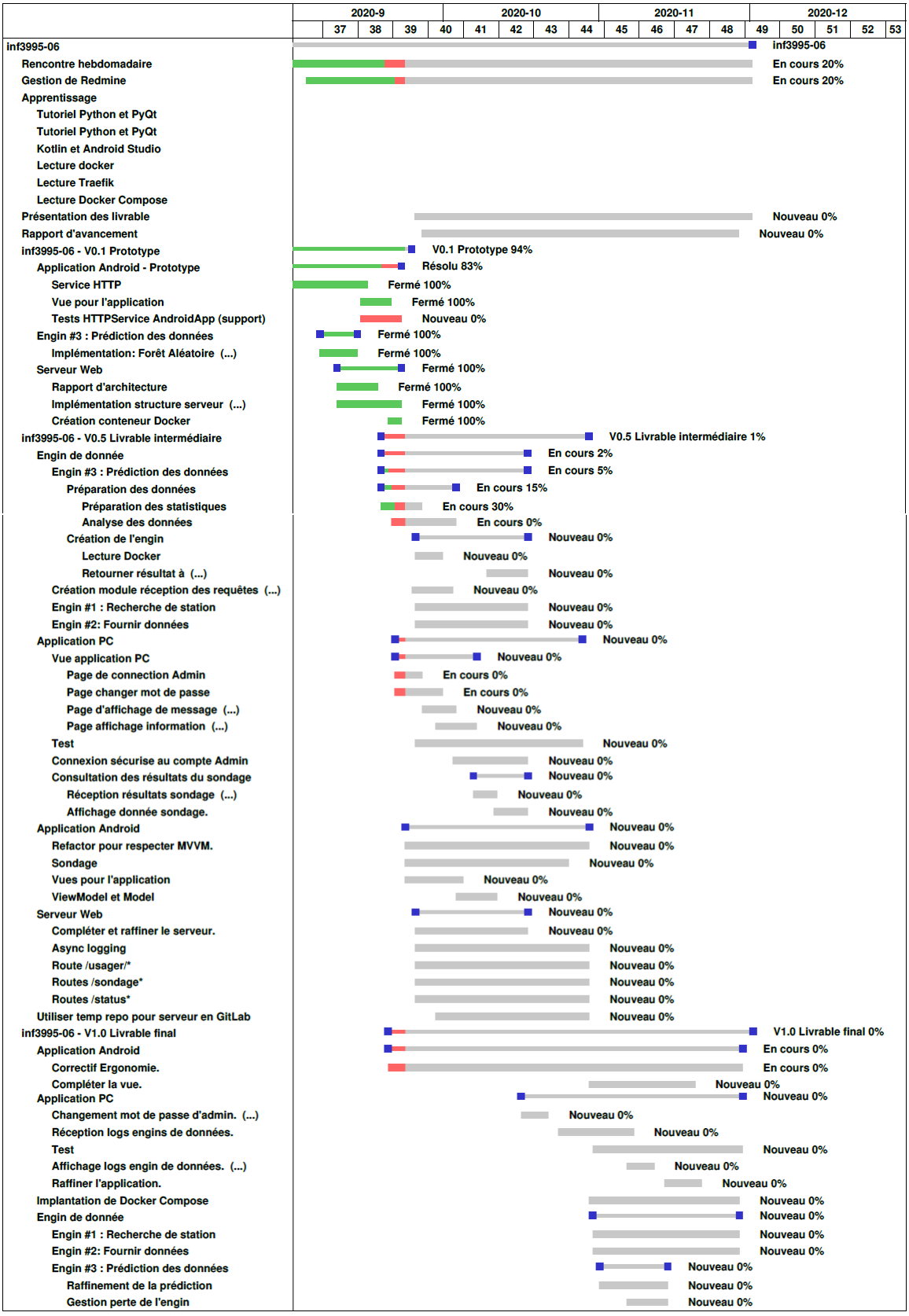

Figure 6 – Diagramme de Gantt ©Redmine

Pour plus de détails pour la répartition des tâches, nous vous invitons à vous référer aux sections 2.1 (Structure d'organisation) et 4.4 (Ressources humaines).

4.3 Calendrier de projet

Le tableau suivant (Tableau 1) indique les dates cibles de terminaisons des phases importantes ainsi que des dates de versions.

Déploiement version 0.1 (prototype) – 24 septembre 2020	
Serveur Web	12 septembre 2020
Application Android	15 septembre 2020
Déploiement version 0.5 (livrable intermédiaire) – 29 octobre 2020	
Serveur Web	17 octobre 2020
Engin de données	
Application PC	28 octobre 2020
Application Android	29 octobre 2020
Déploiement version 1.0 (livrable final) – 1er décembre 2020	
Engin de données	28 novembre 2020
Application Android	29 novembre 2020
Application PC	

Tableau 1 - Calendrier de projet ©Mariam Sarwat

4.4 Ressources humaines du projet

Pour la réalisation du projet, il sera nécessaire d'avoir la contribution de cinq personnes. La gestionnaire d'équipe, Mariam, veillera à l'exécution des tâches et motivera l'équipe. Plus spécifiquement, elle fera la mise à jour hebdomadaire de Redmine. Le choix de la gestionnaire s'appuyait sur le fait qu'elle avait déjà de l'expérience lors d'autres projets de programmation. Ces projets se sont déroulés calmement et les équipes ont apprécié le déroulement des projets. De plus, elle fera l'implémentation de l'application PC.

Le premier développeur, Olivier, qui s'occupera de la composante serveur, a plusieurs années d'expérience en programmation et il est spécialiste dans l'architecture d'un serveur. De plus, il a d'innombrables connaissances dans divers langages informatiques, a déjà complété deux stages et est chargé de laboratoire pour un cours d'informatique à Polytechnique Montréal.

Le deuxième développeur, Jean-Olivier, s'occupera de l'application Android. Voulant relever des défis, il a décidé de réaliser cette tâche pour la

première fois. De plus, il a des connaissances avec Docker. Détenant une plus grande expérience sur le marché du travail, il sera en mesure de donner confiance aux autres membres de l'équipe et d'être disponible en support pour des tâches variées.

Toujours soucieux d'accroître son éventail de connaissances et d'expertises, le troisième développeur Samuel s'occupera de deux des engins de données, soit celui de la recherche des stations et celui de la fourniture des données. Cependant, il détient une expérience dans la conception d'applications Android et agira comme ressource de soutien pour le développement Android. Samuel a plusieurs années de programmation d'expérience et est un grand atout pour l'équipe.

Finalement, Anastasiya s'occupera de l'engin de données de prédiction. Ses expériences de projets intégrateurs et son stage chez Bombardier lui ont permis de parfaire ses connaissances. C'est donc avec confiance et expertise qu'elle relèvera le défi de la conception de l'intelligence artificielle du système.

5. Suivi de projet et contrôle

5.1 Contrôle de la qualité

Afin de suivre le progrès du projet, chaque membre de l'équipe va devoir signer ses commits utilisant Git. Dans le cas où plusieurs personnes ont contribué, la signature de ces derniers sera appliquée. Ces signatures sont un engagement auprès des développeurs que les changements associés à un commit sont authentiques et respect les licences du projet. Nous nous engageons d'ailleurs à respecter le Developer Certificate de Linux Foundation. [6]

Une série de commits, chacun signé, est ensuite envoyée au serveur GitLab en attente de révision par un autre membre de l'équipe. Plus précisément, un membre qui n'a pas participé dans la conception des commits. Le membre effectuant la révision ajoute une signature de révision au commit. Lorsque la totalité des commits de la série possède une signature de révision, la série est fusionnée avec la branche principale par le mainteneur. Si la nouvelle branche passe les tests de l'intégration continue, cette dernière est acceptée. Aucune régression dans les tests n'est acceptée.

L'intégration continue se fait en trois étapes reliées entre elles en pipeline. En effet, si une des étapes échoue, les autres ne sont jamais exécutées. La première étape consiste à compiler les différents logiciels. Elle permet simplement de vérifier l'intégrité des moteurs de production et du code source. La deuxième étape effectue divers tests sur chaque logiciel. Elle permet d'assurer la qualité et la stabilité des logiciels. La dernière étape fait un déploiement des logiciels, soit en utilisant Docker ou un émulateur. Elle

permet de vérifier les dépendances des données, la compatibilité des librairies. Enfin, si ces trois étapes passent, alors les logiciels sont prêts à être déployés en production.

5.2 Gestion de risque

Dans un premier temps, ce projet entraîne des risques techniques. Le serveur devra gérer une grande quantité de requêtes. Pour cela, nous avons conçu deux architectures distinctes, de façon à pouvoir passer vers la seconde en cas de problèmes majeurs. Notre objectif est de fournir une solution évolutive, qui peut s'adapter sous pression, et nous estimons que cette approche est la meilleure, compte tenu du temps qui nous est alloué.

Dans un second temps, il y a également des risques liés à l'organisation. Le changement ou l'ajout de fonctionnalités du client au courant du projet peut entraîner des retards. De plus, il faut considérer que les membres de l'équipe auront de nouvelles connaissances à acquérir. Cela peut entraîner une augmentation sur l'estimation du temps total. Pour éviter les retards, il faut s'assurer d'avoir une marge de manœuvre. Les tâches seront planifiées de telle manière pour être terminées une à deux semaines avant les livrables. Cela laissera le temps de réviser et améliorer les fonctionnalités dans le besoin.

Par ailleurs, afin de sauvegarder les données relatives à BIXI, aux usagers ainsi qu'à l'authentification dans l'application PC, une basse de données SQL sera utilisée. Ainsi, il faut éviter les risques de fuite de données, par exemple des attaques de type « SQL injection ». Nous allons mettre en place des moyens de défense contre ce type d'attaque. Ainsi les données contenues dans la base de données ne pourront pas être volées.

Dans un dernier temps, l'environnement de travail est aussi un risque à prendre en compte. La disponibilité des membres, sachant qu'ils ont aussi d'autres occupations, peut entraîner des retards dans la remise des livrables. Des périodes sont fixées pour travailler sur le projet. De plus, pour éviter les conflits d'équipe, des rencontres hebdomadaires sont organisées pour discuter de l'avancement du projet et des problèmes rencontrés. Aussi, des activités de cohésion sont prévues pour assurer un bon esprit d'équipe.

5.3 Tests

Notre équipe se démarque dans ce domaine. Avec plusieurs expériences antérieures, nous arrivons toujours à livrer des produits robustes et ayant été testés.

En ce qui concerne le serveur web, des scripts sont lancés en parallèle. Ces derniers vont faire différentes requêtes HTTPS au serveur ce qui permettra de faire un test de stress, sur le temps de réponse. Cela aura

aussi comme but de faire une couverture du code le plus complet possible à l'aide de ces requêtes. Cette couverture est faite à l'aide de Gcov. [7] De plus, le serveur sera lié dynamiquement à ASAN durant les tests, afin de pouvoir intercepter les fuites de mémoire et autres. [8]

Par rapport à l'application Android, il faudra s'assurer de tester l'intégrité des données envoyées vers le serveur, et faire quelques tests de performance. Ainsi, afin de tester les performances et l'intégrité des données, des tests unitaires seront créés et seront exécutés chaque fois qu'une nouvelle fonctionnalité sera ajoutée, les tests seront lancés afin d'éviter une régression dans l'application. Par ailleurs, des tests au niveau interface seront exécutés afin de s'assurer que l'expérience utilisateur n'a pas de problèmes. Un plan de test d'interface sera rédigé afin de ne pas oublier des cas à vérifier lors des relâches de nouvelles versions.

Plus concrètement, on devra vérifier les données qui sont envoyées lors de la soumission du sondage par l'application, de même que leur représentation en objets JSON. De plus, il faudra gérer les cas problème, soit les cas où le serveur ne répond pas correctement, ou si l'utilisateur entre des données qui ne sont pas valides. Les outils de test utilisés pour l'application Android seront les scénarios d'activités et Espresso, qui permettent de simuler des actions par l'usager sur l'interface graphique et tactile. Il y aura également des tests unitaires plus traditionnels avec des Mock et des Spy, ce qui sera utile notamment pour tester les réactions aux réponses du serveur web et donc des engins de données.

Afin de tester l'interface utilisateur de l'application PC, le module de test unitaire de Python et le module QTest seront utilisés. Effectivement, cela permettra de simuler les clics de souris et les entrées du clavier. Pour tester la communication entre l'application PC et le serveur, on va effectuer des tests d'intégrités des données qui seront envoyées vers le serveur. Pour ce faire, des tests unitaires seront produits. Ils seront exécutés lors de l'ajout de nouvelles fonctionnalités. Cela permettra de vérifier le comportement de l'application si le serveur ne répond plus ou si l'application plante lors de l'exécution d'une tâche.

5.4 Gestion de configuration

La gestion du code source se fait à l'aide du logiciel de gestion de versions décentralisé Git. Ce dernier permet en effet à tous les membres de l'équipe de travailler de façon locale et indépendante des autres. Pour éviter le plus possible d'avoir des problèmes au niveau des fusions des branches, les différents projets sont séparés entre eux. La Figure 7 démontre l'arborescence principale du projet. Le dossier **server/**, **AI/**, **android/**, **pc/** contiennent respectivement tout le code pour le serveur web, les engins de données, l'application mobile et le programme d'administration. Enfin, le fichier **config.lua** permet de configurer le serveur web.

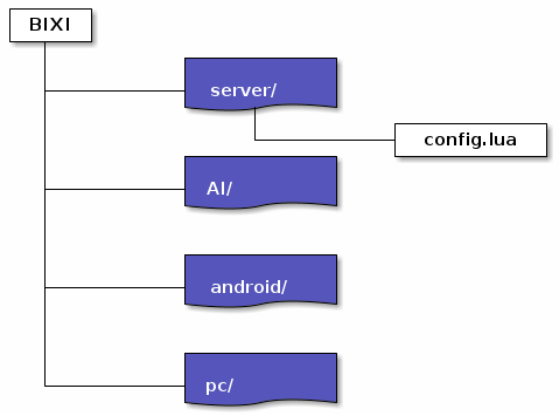

Figure 7 - Arborescence des dossiers du projet ©Olivier Dion

Ensuite, la Figure 8 présente la topologie entre les différents nœuds dans le processus de développement. Le nœud Git local représente une instance locale du projet d'un développeur. Ce dernier communique à l'aide de HTTPS ou SSH avec le nœud de GitLab qui entrepose le projet et qui agit comme nœud central. Ce dernier envoie au serveur Redmine les différentes branches de Git de façon unilatérale. Enfin, le nœud GitLab communique avec un démon, gitlab-runner, qui se situe sur un Linode. Ce démon fait l'intégration continue qui assure la stabilité des logiciels.

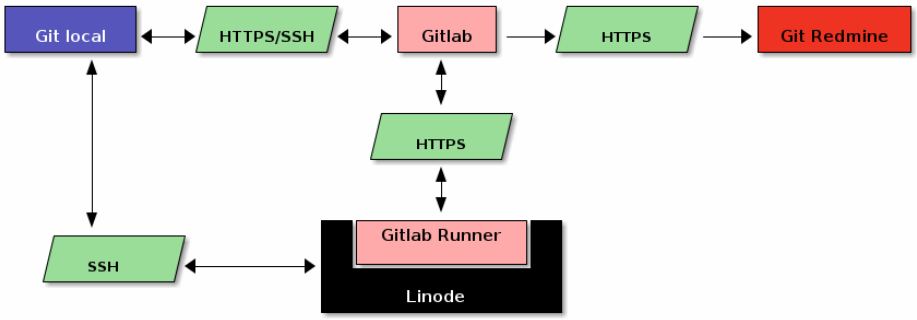

Figure 8 - Topologie des différents nœuds dans le processus de développement ©Olivier Dion

Enfin, les membres de l'équipe travaillent sur leurs propres branches. Pas de convention est en place, mais de façon générale, le nom de la branche devrait inclure le stade, le nom de la développeuse et le nom du logiciel ou fonctionnalité. Par exemple, **devel/old/server** représenterait la branche de développement du serveur web appartenant au développeur Olivier, et **server/jo/master** la branche maître du serveur pour le développeur Jean-Olivier.

6. Références

[1] A. Bisht, « MVVM (Model View ViewModel) + Kotlin + Google Jetpack | by Ankit Bisht | Medium » Medium, 2 mai 2019. [En ligne]. Disponible : https://medium.com/@er.ankitbisht/mvvm-model-view-viewmodel-kotlin-google-jetpack-f02ec7754854.

[2] P. Jahoda, « Github - PhilJay/MPAndroidChart: A powerful Android chart view / graph view library, supporting line- bar- pie- radar- bubble- and candlestick charts as well as scaling, panning and animations. » 2020. [En ligne]. Disponible : https://github.com/PhilJay/MPAndroidChart.

[3] D. Algar, « Kotlin with Volley » Varvet, 4 avril 2017. [En ligne]. Disponible : Controller.

[4] S. Veeraraghavan, « Best Programming Languages to Learn in 2020 » 12 septembre 2020. [En ligne]. Disponible : https://www.simplilearn.com/best-programming-languages-start-learning-today-article.

[5] Machine Koder, « Choosing Qt for Python or Qt C++? » 2019. [En ligne]. Disponible : https://machinekoder.com/choosing-qt-for-python-or-qt-cpp/.

[6] The Linux Foundation, « Developer Certificate Of Origin » 2004, 2006. [En ligne]. Disponible : https://developercertificate.org.

[7] Free Software Foundation, « Gcov (Using the GNU Compiler Collection (GCC)) » 2018. [En ligne]. Disponible : https://gcc.gnu.org/onlinedocs/gcc/Gcov.html.

[8] Google, « Github - google/sanitizers: AddressSanitizer, ThreadSanitizer, MemorySanitizer » 2009-2015. [En ligne]. Disponible : https://github.com/google/sanitizers.