Rapport Projet Semestre 7

AL Langage dédié

IAM Smart Spaces

Décembre 2020 - Janvier 2021

Équipe AL-IAM3

LEBRISSE David

LOSCIALE Vivian

POUEYTO Clément

USHAKA KUBWAWE Kevin

Introduction

De nombreuses villes ont vu leur centre-ville perdre en attractivité. Ces raisons sont principalement dues à la création de nouveaux centres commerciaux, à la forte expansion de l'E-commerce, ainsi qu'au problème de mobilité urbaine.

Beaucoup de "villes moyennes" cherchent des moyens pour redynamiser leur centre-ville. Le produit conçu, PolyVilleActive, est une solution proposée pour répondre à ce problème. PolyVilleActive se focalise sur le problème des transports urbains de la ville.
Cependant, certaines questions se posent. Comment fluidifier et adapter le trafic urbain en prenant en compte les aléas quotidiens de la ville ? Comment peut-on amener les citoyens à adopter une démarche plus écologique ?

Produit

Besoins

Pour répondre à ce premier problème, nous nous sommes penchés sur la question des transports en commun. En effet, ils sont toujours moins utilisés que la voiture et les villes encouragent de plus en plus les personnes à les prendre en mettant en place des parkings relais par exemple car le trafic et l'accessibilité au centre-ville jouent aussi sur son attractivité.
Cependant ils peuvent tout aussi bien connaître des incidents qui viennent perturber la circulation. C'est pourquoi notre produit permet à la fois de fluidifier le trafic et de trouver un itinéraire optimal avec les transports en commun tout en gérant ces incidents en fonction des besoins de la ville.
Nous avons donc défini deux personas. Chacun de ces personas remplit un rôle bien distinct, ce qui couvre les fonctionnalités de l'application.
Ces personas sont les suivants:

Le premier est un expert en mobilité urbaine, employé par la ville qui s'occupe de créer des règles pour fluidifier les réseaux de transports. Celui-ci pourra, à travers un langage, préciser différentes règles de gestion de transport. Il lui sera donc possible de prioriser un moyen de transport dans la ville.

Le second correspond à un utilisateur de l'application. Ce dernier est l'utilisateur final de notre produit. Il va demander à l'application un trajet en indiquant un lieu de départ et d'arrivée. Par la suite, le persona se contentera de suivre le trajet que l'application lui propose.

Vision

Le trafic dans les villes peut être variable. Il n'est jamais possible de connaître à l'avance les aléas qui se trouvent sur le trajet choisi. On peut trouver des transports en communs publics remplis, en retard, ou bien des pannes sur le réseau Tramway de la ville. Tous ces aléas peuvent impacter, de manière importante, la vie des citoyens de la ville.
Notre vision s'est portée sur les incidents ainsi que sur les moyens de transports les plus fréquemment rencontrés dans les grandes villes.
Lorsqu'un aléa important survient sur le trajet d'un utilisateur, il est nécessaire de le rediriger vers un autre itinéraire afin qu'il soit le moins impacté.
Plus la ville possède de transports en communs, plus les possibilités d'itinéraires sont étendues. On souhaite donc donner le pouvoir à la ville de réagir par des règles, à certains aléas qu'un utilisateur rencontre.
Répondre à ce problème pour que les habitants puissent effectuer leur trajet en rencontrant le moins d'aléas possibles est l'axe principal du produit.

Axes et mineures

Le produit conçu s'oriente sur deux axes technologiques.
Le premier axe est un langage dédié: il met l'accent sur la définition d'un langage qui permet de spécifier des règles de transport en fonction de la situation de la ville.
Le second axe est *Smart Space*: cet axe se focalise sur des objets connectés présents dans la ville. Ces objets peuvent être des abris de bus, des bornes de métro, des capteurs… Ces objets peuvent communiquer entre eux et informer l'application pour optimiser le trajet des utilisateurs.

Architecture logicielle - Langage dédié

L'axe du langage dédié permet à des personnes non initiées à l'informatique de développer des interactions en fonction d'événements de plus hauts niveaux liés à l'application.
Cet axe correspond au persona de Stefanie. Celle-ci peut inscrire des règles de priorités parmi les différents moyens de transports en communs via une console dédiée à l'application.
L'utilisateur de la console peut ainsi choisir de prioriser, d'éviter ou d'interdire des moyens de transport, et ce de manière générale ou bien en réponse à des aléas prédéfinis. Ce qui modifiera le résultat des requêtes des prochains utilisateurs.
Ces aléas correspondent à une surfréquentation, un retard ou une panne qui affectent une ligne ainsi qu'à la variation de la météo.
Les moyens de transports pouvant être ciblés sont : le bus, le tramway, le métro et le train.
Dans ce cas, le langage dédié est nécessaire car il est difficile de développer ces règles via un simple formulaire. En effet, les paramètres de déclenchement des règles sont variables au niveau des moyens de transports ainsi que des aléas, ce qui permet de nombreuses combinaisons. Il faut par la suite décrire les conséquences des règles. Aussi, l'utilisation d'un formulaire aurait été longue et fastidieuse pour l'usager.

```
global eviter train fin global
local
quand metro panne alors eviter metro prioriser train 1
quand neige alors prioriser metro 1 interdire bus
quand bus rempli alors prioriser tramway 2
fin local
```

Capture d'écran de règles créées à partir du langage

Intelligence ambiante - "Smart Spaces"

Dans l'axe *Smart Spaces*, les citoyens cherchent à tirer parti de nouveaux objets connectés répartis dans la ville dans le but de rendre l'espace de vie plus intelligent.
Dans le cadre de notre projet, nous avons choisi les abribus (les stations de métro et tramway) connectés qui récoltent des données et les traitent afin d'envoyer des informations de plus hauts niveaux à l'application. En effet, la notion de *Smart Spaces* vise à coordonner les services logiciels et les objets connectés.

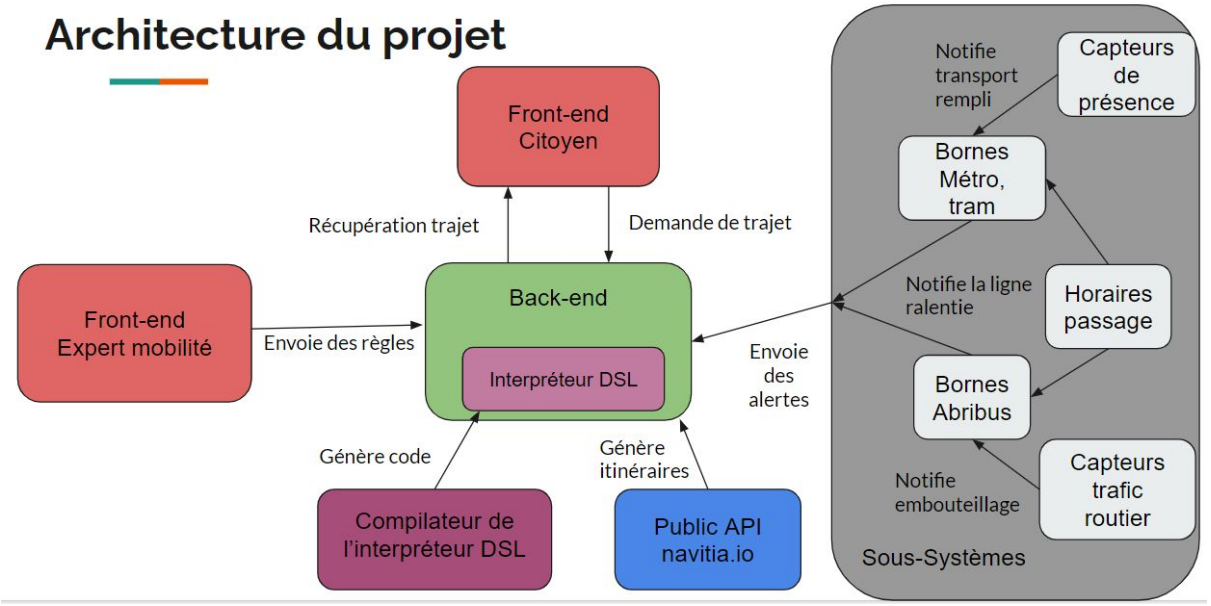
C’est pourquoi nous utilisons des capteurs de présence par infrarouge sur les routes afin de détecter des embouteillages sur une ligne. Lorsque les capteurs détectent une présence prolongée, l'information est envoyée aux abribus de la ligne impactée. Ainsi lorsque plusieurs capteurs de présence envoient une information, le programme de l’abribus détermine s’il s’agit d’un embouteillage ou non.
Au sein de l’abribus, il y a aussi un capteur d’appareils bluetooth permettant de donner une approximation sur le nombre de personnes qui attendent les prochains bus.
Chaque abribus dialogue avec les trois prochains arrêts d’une ligne pour connaître le nombre de personnes qui attendent.
Dans chaque bus sont placés des capteurs infrarouge à l’entrée et à la sortie afin de savoir si la capacité d’accueil du bus atteint sa limite.
En recoupant ces informations, le programme de l’abribus est capable d’estimer si les personnes pourront entrer dans le prochain bus.
Enfin, lorsqu'un bus atteint une station, il notifie l’abribus correspondant. Celui-ci sait alors si le bus est passé en avance, à l’heure ou en retard.
Ainsi chaque abribus notifie l’application lorsqu’un bus est en retard, en avance ou rempli.
Nous avons étendu cette configuration aux stations de métro et tramway en ajoutant une information de panne, et nous comparons également les données météorologiques d’un service météo externe avec les données prises sur le terrain pour savoir si une ligne est impactée.

Croisement des axes

Dans l’application PolyVilleActive, les deux axes se complètent de part le fait que les abribus connectés permettent de détecter des aléas impactant le trafic urbain et envoient les informations à l’application. Celle-ci utilise les règles de la ville définies grâce au langage dédié pour rediriger l’utilisateur au mieux en fonction de sa position actuelle.
Ces actions sont donc complètement transparentes pour l’utilisateur final de l’application mobile car celui-ci n’est alerté que lorsque son itinéraire est replanifié.

Conception

Approches et langages utilisés



Architecture du projet PolyVilleActive et utilisation des services

Serveur parent - Backend

Pour notre serveur parent, notre choix s'est tourné vers le *framework SpringBoot* car le langage Java était connu par toute l'équipe et un membre connaissait les bases du framework.
Nous avons utilisé une approche API REST avec un format de réponse en JSON car il s'agit d'un standard pour l'échange de données entre les clients Web, mobiles et les services *back-end* et qui fait preuve d'une grande interopérabilité. Il était donc facile de se fournir en documentation.

Ce serveur est le centre de l'application car elle reçoit les alertes des abribus, stations de métro et tramway. Elle reçoit aussi les demandes des utilisateurs via l'application mobile ainsi que les changements de règles du DSL.
L'application ne dépend pas des abribus et inversement. Les deux applications fonctionnent séparément mais communiquent ensemble.

Service externe : API Navitia

L'API Navitia permet d'obtenir plusieurs trajets qui utilisent des transports en communs d'un point A à un point B en renseignant les paramètres nécessaires. Elle permet la récupération des informations de chaque transport, arrêts, stations ...
Nous avons conclu que la période d'essai était suffisante pour couvrir la période du projet, néanmoins, dans un contexte plus long cette solution aurait été reconsidérée.

Architecture Logicielle-Langage Dédié

Nous avons recherché plusieurs outils existants nous permettant de mettre en place un DSL (Domain Specific Language), les solutions que nous avons trouvées sont répertoriées dans le tableau suivant.

	DSL	Lexical	Syntaxique	Autre remarque
Groovy	interne	✓	✓	
xText	externe	✓	✓	pour Eclipse
ANTLR	externe	✓	✓	
JFlex	externe	✓	X	Semblable à Lex
BYACC/J	externe	X	✓	Semblable à Yacc
Java_cup	externe	X	✓	

Notre langage devant s'adresser à des non programmeurs, nous nous sommes tournés vers des DSL dit externes qui permettent de dissocier l'interpréteur de notre langage de notre environnement de développement. En effet, ils permettent de générer un interpréteur que l'on peut solliciter dans notre application au besoin.

La gestion des fonctionnalités lexicales et syntaxiques du DSL par ANTLR a été un critère de sélection prépondérant. Les autres solutions nécessitaient d'être combinées entre elles, et apportaient plus de complexité au projet.

Site Web - expert en mobilité urbaine

Le site web permet à un expert en transport employé par la mairie, d'utiliser une console afin d'envoyer de créer des règles liées aux incidents et au trafic urbain. Pour cela nous avons développé une page web en Angular permettant d'envoyer une requête au serveur contenant le code écrit. Angular était une solution évidente pour notre équipe, car tous les membres avaient eu une expérience avec ce *framework* et il suffisait de créer une simple page avec une console.

Application mobile - Citoyen

L'application mobile est destinée aux citoyens de PolyVille, ces derniers peuvent rechercher un trajet en désignant un point de départ et d'arrivée et ajouter les moyens de transport souhaités. Ils peuvent créer un compte, se connecter, accéder à leur profil et récompenses.
L'application mobile constituait donc une partie majeure de notre projet, c'est pourquoi nous avons choisi Flutter comme langage de programmation. En effet, notre équipe n'étant pas à l'aise avec le langage Android, il était plus judicieux de choisir un langage qu'un membre de l'équipe connaissait. Avec les contraintes de temps qui nous étaient imposées, cette solution était idéale et nous a permis de gagner beaucoup de temps afin de développer d'autres parties du projet. De plus, dans le cadre ou l'application aurait été déployée, Flutter aurait permis de générer une version Android et IOS du code l'application, évitant ainsi de devoir utiliser 2 langages (et donc 2 projets) différents.

Intelligence ambiante - "Smart Spaces"

Dans les conditions sanitaires actuelles, il n'était pas possible d'utiliser des objets connectés. Nous avons simulé les connexions et le fonctionnement de nos appareils en JavaScript. Les abribus fonctionnent avec un serveur nodeJS qui reçoit, traite et envoie des informations à l'application et aux autres abribus. Les connexions et l'envoie de données se font avec des requêtes HTTP quand les objets sont distants et les données doivent être *"mockées"* pour déclencher les différents scénarios expliqués dans nos axes. Les informations provenant des capteurs sont envoyées à un intervalle de temps régulier configurable pour éviter qu'ils analysent les données trop vite et déclenchent l'incident dès le début du programme.
De plus, pour exécuter simultanément ces programmes qui sont normalement exécutés sur des machines différentes, nous avons utilisé les libraires de threading tel que *concurrently*. Enfin, nous avons testé au mieux nos simulations et le déclenchement de nos scénarios avec la libraire de test *mocha.* La possibilité de tester des fonctions asynchrones a permis de tester les événements qui se produisent après un certain délai.

Outils utilisés

Planification des tâches

Nous avons utilisé l'outil Cacoo afin de planifier chaque *Sprint* de ce projet. Cacoo a notamment été utile pour discuter des idées, les représenter avec des personas, scénarios puis de créer les *User Stories* associées. Pour chacune d'entre elles nous leur avons donné une taille ainsi qu'une priorité (méthode *Sizing* et *MoSCoW*). Pour chaque *Sprint* nous avons attribué des *User Stories* à réaliser en fonction de notre capacité à développer ces fonctionnalités sur une semaine.
Une fois les *User Stories* créées, nous avons défini la liste des tâches à faire sur GitHub et détaillant ce qui était attendu pour chacune d'entre elles afin d'être validées.

Postman

Durant la phase de développement, nous avons été amenés à utiliser l'outil PostMan afin d'essayer de connecter les différents services entre eux au serveur parent (*backend* application). Ainsi nous avons pu vérifier que chaque requête était correctement formée et qu'elle renvoyait bien la réponse attendue.
Cependant nous n'avons pas réalisé de tests automatisés de nos routes avec Postman, car nous n'avions pas prévu le temps d'apprentissage dans notre liste des tâches. Et cela aurait représenté une surcharge de travail pour l'équipe car aucun membre ne maîtrisait assez cet outil.

Tests

Afin de s'assurer de la qualité du code déployé et de la non régression à chaque release, nous avons réalisé des tests unitaires ainsi que des tests d'intégration.
Au sein de notre serveur de l'application, nous avons réalisé des tests unitaires avec Junit pour chaque fonctionnalité ajoutée afin de tester son bon fonctionnement.
De plus, l'utilisation de Pitest nous a permis d'évaluer des cas limites et de trouver les parties qui avaient besoin de meilleurs tests.
Des tests unitaires ont aussi été réalisés sur l'application de l'abribus avec Mocha.
C'est la bibliothèque de référence de test en JavaScript, c'est pourquoi nous l'avons choisie. Elle contient tous les outils pour réaliser des tests et sur tout type de fonctions, notamment les fonctions asynchrones.

Afin de s'assurer du bon comportement de la mise en relation de plusieurs composants et de l'enchaînement des processus complets, nous avons mis en place des tests d'intégration en utilisant Cucumber.
Avec ce *framework* nous avons testé les routes ainsi que les relations entre les services à travers des scénarios.

Sonar

Sonar nous a permis avant la fin de chaque *Sprint* de savoir si la qualité du code attendue était atteinte ou non. Nous nous étions fixé un objectif de couverture de 75% car cela correspondait à ce que nous avions obtenu à la fin du premier *Sprint* alors que l'application fonctionnait correctement. A la fin du 3e *Sprint*, notre coverage atteignait les 84%.

Innovation

Besoin

Le but de notre innovation est de sensibiliser et de changer les comportements concernant les personnes dans les transports publics car selon la manière dont nous utilisons les transports publics, cela peut avoir un impact sur notre environnement et les interactions sociales entre les utilisateurs. Nous voulons que nos utilisateurs soient plus respectueux de l'environnement et plus sociables.

Solution

Pour ce faire nous avons ajouté de la *gamification*. Le principe est d'ajouter des éléments de jeu pour améliorer l'engagement des utilisateurs. Nous utilisons des récompenses pour les joueurs qui accomplissent certains gestes. Les types de récompenses incluent des points et des badges de succès. Les points cumulés conduisent à de vraies récompenses comme des codes promotions ou bien des tickets gratuits. Ces récompenses proviennent des magasins de PolyVille afin d'inciter les utilisateurs à s'y rendre.

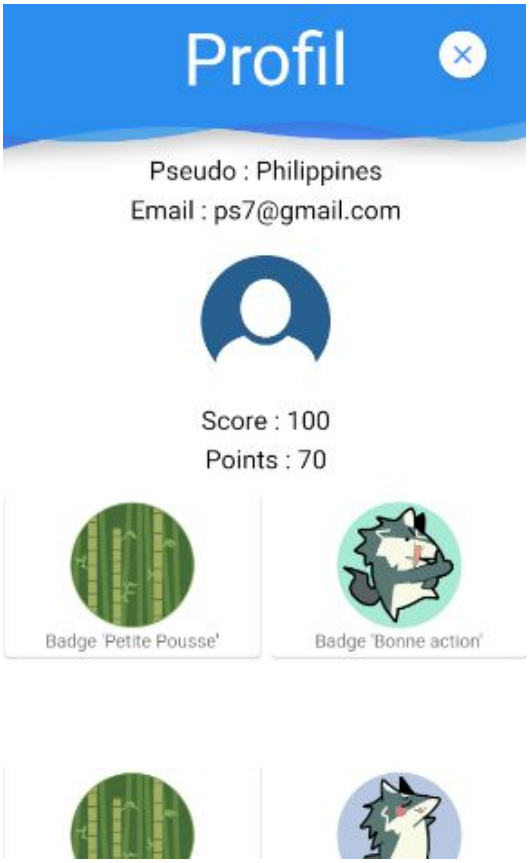


Image 1 *Profil du jeu pour un utilisateur*

Image 2 *Badge vert "Petite Pousse" (à gauche), Badge civique "Bonne Action" (au milieu), Badge civique "Bon citoyen" (à droite)*

Actuellement, il existe deux moyens pour gagner des points. Le premier cas est le trajet vert. L'utilisateur a le choix d'emprunter le trajet qui a le moins d'émission CO2 en activant le mode vert. S'il réalise ce trajet, on lui confère un bonus de points.
Plus l'utilisateur a de moyens de transports sélectionnés au début, plus il cumulera de points.
Dans le deuxième cas, il cède sa place dans les transports. S'il est dans un transport rempli et qu'une personne souhaite prendre ce bus durant son itinéraire, une demande lui est envoyée et il lui est possible de céder sa place. Ce geste sera récompensé par l'ajout de points supplémentaires.

Par la suite, on pourrait imaginer un système de classement dans l'application qui mettrait en avant les utilisateurs les plus assidus, ainsi qu'un système de titres (touriste, citoyen, maître ...).

Originalité

D'autres applications similaires utilisent la *gamification*. Waze est l'un d'entre eux: cette application accumule des points et encourage l'utilisateur à se surpasser mais il n'y a pas de récompense particulière, elles ne sont que fictives.
Il y a également Street Life Project: ce projet encourage les trajets verts par un système de points sans conséquence. C'était un projet fait en 2013 pour regarder l'impact d'une *gamification* dans la vie des gens. Enfin, on a Bart Perks. Dans cette application, les abonnés reçoivent des points s'ils prennent les transports publics en dehors des heures de pointe. Accumulés, ces points permettent aux voyageurs d'accéder à des récompenses : des trajets gratuits, de petites récompenses, de l'argent, l'accès à une loterie avec des lots plus importants, etc.
Notre projet est différent car il intègre la gestion des incidents dans les transports en commun. La *gamification* de ces projets n'apporte rien de concret à l'utilisateur.
Nous utilisons directement les informations de l'application et de la ville afin de proposer ces services uniquement aux citoyens concernés.
Lorsque les capteurs du bus détectent une sur fréquentation du bus, les utilisateurs ayant ce bus dans leur trajet recevront une notification et se verront proposer de changer d'itinéraire si cela est possible. Notre système de récompense permet d'accorder des réductions dans les différents commerces de proximités ou bien d'avoir des trajets gratuits (selon le nombre de points que l'on possède). Cela permet de fluidifier les trafics, et aussi de redynamiser/inciter les personnes à se rendre dans les commerces de proximité de la ville.

Rétrospective du projet

Analyse et critique

A long terme, le serveur de notre application devra permettre à toute une population de trouver et de suivre un itinéraire sûr, cependant, elle n'est pas en mesure de gérer un grand nombre de personnes en plus de toutes les connexions aux abribus. En effet, il n'y pas de mécanisme pour gérer la surcharge du serveur et l'application mobile fonctionne en lui envoyant régulièrement des requêtes pour mettre à jour ses données. L'utilisation simultanée de milliers de smartphones augmentera le risque de saturation du serveur. Nous avons utilisé cette solution car elle paraissait être la plus efficace à une petite échelle et déployable plus rapidement. Cependant, avec du temps supplémentaire, le serveur actuel aurait été

transformé en un serveur de push. Ainsi, le serveur pourrait notifier l'application plus facilement de cette manière sans appels inutiles.

De plus, les objets connectés ont été simulés. Nous n'avons pas fait face aux réelles contraintes techniques ou physiques que l'on peut rencontrer en utilisant ces objets. Il est possible que notre solution soit modifiée ou adaptée selon son fonctionnement en conditions réelles comme les interactions entre les capteurs et les abribus.

Enfin, nous avons utilisé la dernière version d'ANTLR (4) et avec du recul, il aurait été préférable d'utiliser la version antérieure (3), pour laquelle la documentation était bien détaillée sur internet, et qui présentait déjà les fonctionnalités dont nous avions besoin. Cependant, la version 4 ne présentait pas de documentation à jour, et cette documentation était de plus consignée dans un livre qu'il fallait se procurer moyennant finances. Nous avons donc dû chercher en plus des extraits gratuits, des tutoriels et sources annexes d'exemples afin de comprendre le fonctionnement de cette version.

Méthodes de travail

Dans le groupe, nous nous sommes spécialisés par axe et par technologies. Une partie du groupe a travaillé sur la partie langage dédié avec ANTLR pour traiter le langage et l'application Web Angular pour écrire les règles de priorités. Un membre s'est focalisé sur la partie application mobile avec Flutter, et application des règles de priorités, et un autre membre sur la partie simulation des objets connectés et déclenchement des scénarios.
Nous avons mis en place un kanban pour savoir chaque jour quelles étaient les fonctionnalités à implémenter et celles qui étaient déjà faites. De cette manière, et avec le daily meeting, nous avons gardé une vision de ce qu'il nous restait à faire pour préparer une démonstration dans les temps.
Nous avions également mis en place une stratégie de branches, la stratégie *gitflow*. Nous avions au début de projet, en plus de la branche principale (*main*), une branche *develop* dans laquelle nous avions commencé à mettre en place l'architecture du projet puis nous avons commencé à créer des branches *features* pour les différentes fonctionnalités. A chaque *push* dans la branche *develop*, la CI se lançait pour vérifier le build dans *develop* et *main*.

Definitions of Ready and Done

Nous considérons que nous sommes prêts (Ready) à commencer le développement pour une issue, lorsque le tag de sizing et de MoSCoW ont été

attribués et que le contenu est bien défini. On considère une issue bien définie lorsque la description de la fonctionnalité à implémenter est comprise par tous les membres de l'équipe,que les critères d'acceptation sont bien définis et qu'au moins un test d'acceptation est explicité.

Nous considérons qu'une issue est terminée (Done) lorsque les tests d'acceptation passent, que le build de la CI est validé, que les tests unitaires et de comportements sont vérifiés et qu'elle est intégrée à la branche commune.

Auto-évaluation

LOSCIALE Vivian: 90 points
POUEYTO Clément: 120 points
LEBRISSE David: 90 points
USHAKA Kévin: 100 points

Conclusion

PolyVilleActive nous a permis de découvrir des technologies différentes et d'avoir un aperçu d'un projet qui doit répondre aux besoins d'un client.
Nous avons rencontré des difficultés lors de la définition de notre sujet et avons été agréablement surpris lorsque les deux axes se sont révélés compatibles entre eux.

Du côté des technologies utilisées, nous nous sommes initiés aux *frameworks* Spring pour le *back-end*, Flutter pour l'application mobile ainsi qu'Antlr pour le DSL. Nous nous sommes également améliorés avec Angular pour la partie *front-end*, avec JavaScript pour les objets connectés et Cucumber, Junit ainsi que Mocha pour les tests.
Nous avons également appris à faire des choix de conception, à créer une architecture adaptée à un projet et à faire communiquer des *front-end* et des services externes au *back-end*.

La définition des *User stories* nous a montré l'importance de communiquer avec le client afin de livrer de la valeur et de cerner les points les plus importants à développer en priorité.
Enfin, le découpage en sprints hebdomadaires associé à la planification est une méthode que nous avons trouvé efficace pour connaître nos capacités de développement.