AL1

Rapport final

Projet Semestre 7 – SI4

Membres de l'équipe :

João BRILHANTE

Charly DUCROCQ

Quentin LAROSE

Loïc RIZZO

UNIVERSITÉ CÔTE D'AZUR | POLYTECH NICE-SOPHIA

Table des matières

I. Produit

1. Description

PolyVilleActive est un projet de conception d'outil numérique à destination des collectivités locales, des habitants et des commerçants dont **le but est de dynamiser les villes** en mettant en relation les acteurs majeurs de la ville. L'essentiel est d'informer des activités, des promotions, des dates à ne pas rater, de la création de nouveaux commerces de proximité, des places de stationnement et des transports en commun.

2. Axes

a. [AL-2] Extensibilité & Interopérabilité

Premièrement, cet axe a pour objectif **que notre produit fonctionne avec le plus grand nombre d'applications différentes**. Pour cela, notre produit doit définir des points d'extensibilité pour permettre à d'autres utilisateurs d'étendre ses capacités et de l'utiliser de plusieurs manières que nous n'avions pas nécessairement imaginé au préalable.

Deuxièmement, cet axe a pour objectif que **notre produit fonctionne avec d'autres systèmes informatiques, existants ou futurs, sans difficulté particulière** de mise en place. Pour cela, notre produit doit être capable de communiquer avec d'autres systèmes de la ville. De même, les autres systèmes de la ville doivent être capables de communiquer avec notre produit sans problème.

b. [AL-3] Usine logicielle & Armée de clones

Premièrement, cet axe a pour objectif que **notre produit puisse produire un logiciel adapté aux besoins de différentes villes**. Pour cela, notre commercial doit être capable d'assembler un logiciel différent en fonction des besoins du client.

Deuxièmement, cet axe a pour objectif que **notre produit puisse être découpé en plugins qui ajoutent différentes fonctionnalités**. Pour cela, notre produit doit permettre le développement de plugins qui ajoutent des fonctionnalités au produit seulement lorsqu'ils sont activés.

3. Personas

a. Sylvain Dupond

Il s'agit du **maire d'une ville**. Il souhaite dynamiser le centre-ville, analyser les statistiques de fréquentation de la ville et analyser les statistiques économiques de la ville.

b. Carine Moulin

Il s'agit d'une **visiteuse de la ville**. Elle souhaite avoir accès aux informations publiques de la ville (les magasins, les promotions, les établissements, les événements, etc.).

c. Jean Martin

Il s'agit d'un **commerçant de la ville**. Il souhaite partager les informations et les promotions de son magasin. Il souhaite également être capable de faire des publications pour rester en contact avec ses clients.

d. Aline Robert

Il s'agit d'une **organisatrice d'événements**. Elle souhaite partager les informations liées à ses événements. Elle souhaite également être capable de communiquer à propos de ses événements à l'aide de publications.

e. Louis Girard

Il s'agit du **responsable d'un parking de la ville**. Il souhaite partager les informations de son parking. De plus, il souhaite communiquer à tout instant le nombre de places disponibles.

f. Robert Ducreux

Il s'agit d'un **agent de sécurité de la ville** ou d'une force de l'ordre. Il souhaite avoir accès à des informations de fréquentation de la ville en temps réel. En effet, il souhaite détecter rapidement les regroupements importants ou les événements nécessitant son intervention.

g. Jérémie Belpois

Il s'agit d'un **développeur pour une société de la ville**. Il souhaite compléter notre produit avec de nouvelles fonctionnalités. De plus, il souhaite utiliser notre produit pour concevoir une application pour son client.

h. Hugo Allen

Il s'agit du **responsable commercial de notre société**. Il souhaite commercialiser notre produit au plus grand nombre de collectivités locales. Pour cette raison, il souhaite que notre produit s'adapte facilement aux besoins des différentes villes.

II. Problématiques

1. Technologies

La première problématique que nous avons rencontrée est le **choix des langages et des technologies du projet**. Nous avons choisi de développer le backend en Java et le frontend en JavaScript. Ces choix se sont imposés d'eux-mêmes car tous les membres de l'équipe avaient de l'expérience sur ces deux langages de programmation. En effet, au vu du temps imparti à la réalisation du projet, la découverte d'un nouveau langage aurait fait perdre à l'équipe un temps considérable.

Nous avons ensuite décidé de développer le backend à l'aide du framework Spring Boot et le frontend à l'aide du framework Angular. Nous avons choisi Spring Boot et Angular pour des raisons bien précises. En effet, ce sont tous deux des frameworks à la fois modernes et matures. Spring Boot fait partie du framework Spring, développé depuis 18 ans, largement entretenu et documenté. Cela nous permet d'utiliser l'écosystème très riche de ce framework et d'augmenter drastiquement notre productivité. Angular, quant à lui, est un projet développé par Google depuis 12 ans, largement utilisé et documenté. Cela nous permet d'utiliser une architecture basée sur des composants réutilisables et testables.

2. Plugins

La deuxième problématique que nous avons rencontrée est la **conception d'un système de gestion de plugins**. En effet, il fallait trouver un moyen de concevoir une architecture Core / Plugins qui permette l'utilisation du framework Spring Boot. Pour cela, nous avons utilisé le framework PF4J dont une version est compatible avec Spring Boot. Cependant, la mise en place d'une telle architecture est complexe et demande beaucoup de temps. En effet, la première version fonctionnelle de notre architecture a vu le jour en fin de premier sprint.

Cependant, cette première version présentait de nombreux défauts. Par exemple, le partage d'informations, entre plugins ou entre les plugins et le core, n'était pas possible. De ce fait, nous avons décidé d'utiliser un nouveau framework, SBP (pour Spring Boot Plugins), qui offre un meilleur niveau d'abstraction et utilise toujours Spring Boot et PF4J. Ce framework nous a permis d'obtenir enfin une architecture stable et solide pour notre backend. Nous avons ainsi pu connecter le frontend et le backend en début de second sprint. Cependant, bien que facilitant l'utilisation combinée de Spring et PF4J, ce nouveau framework n'a pas fait que nous simplifier la tâche. En effet, les frameworks SBP et PF4J étaient très peu documentés. De ce fait, il était parfois difficile d'obtenir des réponses lorsque nous étions confrontés à un problème.

Pour toutes ces raisons, nous avons accumulé, tout au long du développement de cette architecture, un retard très conséquent par rapport à nos estimations. Ce retard nous a obligé à abandonner un certain nombre de fonctionnalités. Le choix des fonctionnalités à conserver s'est fait avec un regard très attentif sur nos axes. La priorité était de répondre correctement à nos deux axes. Nous constatons cela particulièrement avec nos plugins actuels :

- Les **plugins shops et events** ont permis de développer et démontrer l'extensibilité et le polymorphisme de notre produit. En effet, dans le plugin shops, nous produisons des promotions, tandis que dans le plugin events, nous produisons des événements. En réalité, ces éléments sont tous deux des types de publications. De cette manière, il est possible de récupérer les promotions et les événements de manière indépendante. Il est également possible de récupérer les publications de manière générale en faisant une seule requête. Cela permet à des systèmes de traiter tous les types de publications même si cela n'avait pas été prévu initialement. La seule différence réside dans les informations supplémentaires fournies par chaque élément. Par exemple, un événement propose des informations supplémentaires telles que la durée de l'événement, le lieu où il se déroulera, etc.

- Le **plugin parking** a permis de démontrer la consommation de plusieurs API externes par notre produit. Chaque API externe à son plugin d'adaptation dédié qui permet de renseigner les informations que fournissent ces API à notre plugin parking.

- D'autres fonctionnalités ont été volontairement mises à l'écart car jugées moins importantes par rapport à nos axes. Par exemple, l'implémentation de la détection et l'enregistrement de la position des visiteurs de la ville aurait apporté de la valeur pour Sylvain Dupont mais cela se serait fait au détriment de Jérémie Belpois et Hugo Allen qui mettent l'accent sur nos axes.

3. Base de données

La troisième problématique que nous avons rencontré est la **persistance des données des plugins**. En effet, puisque nos plugins devaient être capables d'étendre et de définir de nouveaux types de données, la question de la persistance des données s'est vite posée. En effet, il fallait pouvoir stocker toutes les données en évitant les conflits entre les plugins. Pour cette raison, nous avons décidé de connecter chaque plugin à une base de données indépendante. De cette manière, nous pouvons garantir la protection des données. Ensuite, chaque plugin peut partager, ou non, les services qui permettent d'accéder et de modifier ces données aux autres plugins. Ainsi, les développeurs ont le choix de partager ou non l'accès à leurs données aux autres plugins.

Nous avons choisi d'utiliser des bases de données SQL car la plupart des membres avaient de l'expérience sur ce type de base de données. De plus, les bases de données relationnelles SQL offrent plusieurs niveaux de sécurité supplémentaires. Notamment, elles garantissent l'atomicité des opérations et la consistance des données entre chaque transaction à l'aide d'un schéma relationnel prédéfini.

4. Docker

La quatrième problématique que nous avons rencontré est le **déploiement de notre produit**. En effet, avec un projet composé d'un serveur, d'un client et d'autant de bases de données que de plugins, nous nous sommes vite posé la question du déploiement du projet. D'une part, parce que la mise en place de base de données est un processus long lors de la phase de développement. D'autre part, parce que le déploiement du projet entier serait beaucoup trop long lors de la phase de déploiement. Pour toutes ces raisons, nous avons décidé de dockeriser notre projet. De cette manière, il est possible de lancer tout le projet en une seule commande lors de la phase de déploiement. Il est également possible de lancer seulement les bases de données nécessaires lors de la phase de développement. Cela permet donc de déployer notre solution rapidement mais également à l'avenir d'utiliser des outils d'orchestration de conteneurs, tels que Kubernetes, pour déployer et surveiller les conteneurs de notre solution pour les différentes villes lors de la phase de production.

5. Tests

Enfin, la dernière problématique que nous avons rencontré est le **développement des tests pour notre produit**. En effet, notre architecture et nos fonctionnalités reposant essentiellement sur notre système de plugin, il était compliqué de tester entièrement notre architecture. Plusieurs options se sont offertes à nous : effectuer des tests unitaires pour chaque module du projet et effectuer des tests d'intégration de l'ensemble des modules du projet. Les tests unitaires ont pour but de tester chaque module du produit de manière indépendante. Quant aux tests d'intégration, ils permettent de vérifier que l'ensemble des modules fonctionnent bien ensemble.

Dans notre cas, ces tests d'intégration se présentent sous la forme d'un Workflow sur GitHub Actions. Notamment, sous la forme de notre Workflow pour le serveur qui se charge de lancer tous les tests unitaires, d'assembler le serveur avec tous les plugins, de lancer le serveur, d'ajouter un plugin externe au repository et de lancer l'ensemble de tests de notre collection Postman. Ce test d'intégration pour le serveur nous permet de nous assurer que tous les modules du serveur passent leurs tests unitaires mais également que tous les plugins sont chargés, activés et qu'ils fonctionnent correctement. De plus, il nous permet de vérifier que toutes les requêtes du serveur sont accessibles et fonctionnent comme prévu.

Enfin, les Workflows étant liés à notre repository GitHub, ils nous permettaient de surveiller l'intégration continue du projet et d'être prévenus en cas de problème suite à une modification. Le but de tous ces tests est avant tout de garantir la qualité de notre produit pour les clients.

III. Innovation

1. Description

Nous avons voulu trouver une innovation qui réponde à un **besoin de modernisation des infrastructures de la ville**. Nous avons alors pensé à moderniser les parkings de la ville. En effet, dans un parking classique, il faut prendre un ticket à l'entrée et le conserver jusqu'à la sortie pour payer. Cependant, il n'est pas rare de perdre ce ticket ou même d'oublier dans quel parking ou sur quelle place nous nous sommes garés. De plus, lorsque l'on perd le ticket du parking, il n'est pas rare de devoir payer une journée entière de stationnement. Pour toutes ces raisons, nous avons pensé à une **solution de dématérialisation du ticket de parking** pour faire en sorte que les visiteurs n'aient plus à s'inquiéter de la perte de leur ticket de stationnement.

a. Parking fermé

Lorsqu'un visiteur entre dans un parking fermé de la ville, un code QR est généré et affiché sur la borne d'entrée. Le visiteur peut alors utiliser son smartphone pour scanner le code QR affiché. Il est alors redirigé vers le site de SmartParking qui se charge de valider et de sauvegarder le ticket dans ses cookies. La borne d'entrée peut alors détecter la validation du ticket et ouvrir la barrière. En cas de problème, le visiteur a toujours la possibilité de demander un ticket classique sur la borne d'entrée.

Une fois dans le parking fermé, le visiteur peut trouver une place et se garer. Une fois stationné, il a la possibilité de scanner le code QR qui se trouve devant sa voiture. De nouveau, le visiteur est redirigé vers le site de SmartParking qui se charge de sauvegarder la place de stationnement dans les cookies. De cette manière, depuis son smartphone, le visiteur a accès à son ticket numérique et à la place dans laquelle il s'est garé.

Pour sortir du parking, le visiteur peut payer son stationnement depuis le site de SmartParking. Il peut également payer son stationnement depuis la borne de paiement du parking. En effet, il lui suffit de scanner son ticket numérique ou de saisir l'identifiant de son ticket numérique sur la borne de paiement.

b. Parking ouvert

Une fois stationné, le visiteur peut repérer la borne de paiement la plus proche. Une fois à proximité de la borne de paiement, le visiteur peut scanner le code QR qui s'y trouve. De cette manière, il est redirigé vers le site de SmartParking qui lui permet d'indiquer sa plaque d'immatriculation et d'indiquer la durée initiale de son stationnement. Enfin, pour valider et recevoir son ticket numérique, il lui suffit de procéder au paiement.

Cependant, le visiteur peut également choisir de ne pas scanner le code QR et utiliser la borne de paiement la plus proche. Il doit également saisir sa plaque d'immatriculation ainsi que la durée initiale de son stationnement. De cette manière, le visiteur peut procéder au paiement et recevoir un ticket papier muni d'un code QR. Ce code QR peut être scanné par le visiteur s'il souhaite recevoir un ticket numérique et profiter de ses avantages.

Depuis le site de SmartParking, le visiteur a accès à son ticket numérique. De plus, le visiteur peut visualiser le temps de stationnement restant ou décider de payer une durée supplémentaire de stationnement. Les informations fournies aux agents de surveillance de la voie publique sont alors mises à jour et le visiteur évite une contravention.

2. Originalité

Notre innovation tire son originalité de sa **simplicité et de sa liberté d'utilisation**. En effet, d'autres solutions existent déjà pour dématérialiser le ticket de parking des visiteurs. Cependant, ces solutions requièrent le plus souvent le téléchargement d'une application particulière ou la saisie des informations de l'utilisateur (telles que son nom, son adresse, sa plaque d'immatriculation ou encore sa carte de crédit). Dans notre cas, le fonctionnement **requiert simplement un smartphone et une connexion internet**. Le ticket est stocké dans les cookies et peut facilement être récupéré par l'utilisateur. Il est possible ensuite de payer à la borne ou directement depuis le site internet. De plus, l'utilisateur peut toujours décider d'utiliser le ticket papier s'il ne souhaite pas utiliser cette nouvelle technologie. Enfin, notre solution permet de **sauvegarder la place de stationnement** du visiteur. C'est une fonctionnalité que nous n'avons trouvé dans aucune des solutions existantes.

IV. Analyse critique

1. Limites

a. Dépendances

Nous avons majoritairement construit notre projet à l'aide de dépendances populaires, matures, testées et bien documentées. Cependant, une de nos dépendances échappe à cette règle. En effet, le framework SBP (pour Spring Boot Plugins) nous a grandement aidé lors de ce projet en proposant une meilleure abstraction pour l'utilisation de Spring Boot et PF4J. Cependant, **le framework SBP n'est clairement pas recommandable en termes de qualité**. En effet, ce projet est principalement maintenu par une seule personne, peu testé, peu maintenu, et surtout sa documentation est sommaire.

Malheureusement, dans les conditions de ce projet, il n'aurait pas été possible pour nous d'atteindre un produit aussi abouti sans utiliser cette dépendance. En effet, du retard s'était déjà accumulé en développant notre architecture. De plus, même si nous avions remarqué les défauts de ce framework plus tôt, il n'aurait pas été possible pour nous de faire autrement. La recherche d'un autre moyen ou le développement de notre propre architecture utilisant Spring Boot et PF4J aurait demandé un temps considérable. Nous avions besoin de concevoir notre architecture Core / Plugins au plus vite pour pouvoir développer le reste du projet. Pour conclure, dans le cadre de notre projet, le framework SBP était nécessaire mais dans le cas d'un projet avec plus de temps, d'expérience et de ressource, il faudrait éviter ce framework ou contribuer à ce projet en développant des tests et de la documentation.

b. Tests

En ce qui concerne les tests, les workflows et les tests unitaires couvrent majoritairement notre projet. Cependant, **nous aurions aimé faire plus de tests unitaires et plus de tests pour la partie client** de notre solution mais nous avons pris la décision de ne pas les réaliser pour plusieurs raisons. La première raison est que leur importance au sein du projet et leur utilité auraient été minimes. Certes tout test est le bienvenu puisqu'il permet d'assurer le bon fonctionnement du projet, mais le peu de valeur que ces tests auraient apporté et le manque de temps, qui est notre seconde motivation, nous ont poussé à ne pas réaliser de tests en Angular pour le client.

c. Langages

En ce qui concerne les langages de programmation, nous pensons avoir fait les bons choix. En effet, **Java est un standard dans la conception logicielle**. Cela nous permet d'utiliser des frameworks et des outils standards et matures. Notamment, cela nous permet d'utiliser le framework JUnit ou encore l'écosystème Spring. De plus, nous avons trouvé le framework PF4J qui permet d'implémenter une architecture plugins assez rapidement. De même, **JavaScript est un standard dans le développement web**.

Cela nous permet d'utiliser des frameworks nombreux et matures tels qu'Angular. Notamment, Angular nous permet d'utiliser des composants préconçus avec des bibliothèques telles qu'Angular Material.

d. Documentation

Notre backlog manque un peu de documentation. En effet, s'il est plutôt bien commenté, ce n'est pas ça qui le rendra utilisable par celles et ceux qui voudront utiliser l'API. Nous nous sommes penchés un jour sur la production d'une telle documentation. Nous pensions utiliser Swagger, cependant, ce framework, qui normalement est appliqué sur un code Spring, ne se combinait pas correctement avec PF4J. Nous avons ensuite essayé de nous pencher sur Spring Rest Docs mais ce framework s'est avéré trop long à prendre en main et à écrire. C'est après plus d'une demi-journée de recherche que nous avons décidé de mettre de côté ce besoins afin de nous concentrer sur d'autres ayant un meilleur rapport valeur/temps-nécessaire. A ce jour, nous avons documenté les **routes du serveur** à l'aide de notre collection Postman et le **déploiement du projet** à l'aide de Docker.

2. Pratiques

Malgré des pratiques de gestion de notre backlog plutôt bonnes, celles-ci présentaient tout de même certaines lacunes.

Premièrement, bien que nous renseignions toutes les issues sur lesquelles nous travaillons et que nous y attribuons nos commits non-mineurs, nous n'utilisons pas toutes les fonctionnalités offertes par GitHub comme l'attribution d'un responsable à une issue.

Deuxièmement, un grand flou résidait concernant la marche à suivre des issues abandonnées ou remises à plus tard. Nous les fermions alors qu'une issue fermée devrait être une issue terminée, pas remise à plus tard. Nous aurions plutôt dû utiliser un label "remis à plus tard" ou quelque chose comme ça.

Finalement, un autre défaut est le manque de releases présentes sur le GitHub. De plus, celles-ci auraient pu être automatisées à chaque vérification valide de la Definition of Done définie ci-dessous.

3. Definition of Ready

- La fonctionnalité est bien décrite.
- La fonctionnalité a été estimée en termes de points.
- La fonctionnalité a été estimée à l'aide de la méthode MoSCoW.
- Des critères d'acceptation sont définis.
- Au moins un test permet de vérifier les critères d'acceptation.

4. Definition of Done

- Les tests d'acceptation passent.
- Les nouveaux et anciens tests unitaires passent.
- Les fonctionnalités sont intégrées sur la branche commune.
- Les tests d'intégration continue passent.

5. Répartition dans l'équipe

Pour la répartition du travail au sein de l'équipe, nous cherchions à **échanger le plus régulièrement possible les responsabilités de chacun** afin de permettre à tous les membres de toucher à toutes les parties du projet. En effet, cela permet d'éviter à un membre de se cantonner à une seule technologie et d'améliorer notre productivité.

João	Charly	Quentin	Loïc
130	100	85	85