Ministère de l'enseignement supérieur et de la recherche

Institut National des Sciences Appliquées
Toulouse

Projet de fin d'études

Développement Android au sein d'une startup

LIBERTY RIDER

Hugo Djemaa
Informatique et Réseaux

Liberty Rider
32 Rue des marchands
31000 Toulouse

Août - Décembre 2017

Remerciements

Merci Martin, mon maitre de stage qui réflechit très vite, constamment et à bon escient, pour m'avoir encadré et avoir su me partager ton amour du beau code et parfois même des tests. Merci Manu pour m'avoir accueilli chaleureusement au sein de ta famille, qui est ensuite aussi devenu ma famille, pour tes conseils d'entrepreneurs et la source de motivation. Merci Julien pour ton humanité, ton amitié et pour la cohésion que tu apportes au sein de cette famille. Merci Jérémie de partager ta science, ton goût du python et pour le monitoring que tu prodigues aux développeurs, qui nous empêche de tout casser. Merci Ruben pour tes inombrables conseils, reviews, et pour les assistances entre 13h et 14h. C'était et c'est un plaisir de travailler avec toi. Merci Alex, Axelle, Benji, Coralie, Jonathan, Julien, Marion, Mathieu, Paul, Phileas, Pichoun, Sebastien. Merci à vous d'être qui vous êtes et d'apporter autant, humainement et professionnellement. Merci Najda pour ton regard et ton support au quotidien. Merci Alain et Jeanne pour votre soutien moral et financier pendant ce stage mais aussi depuis le début de mon enseignement supérieur, ou même depuis le début de ma vie, merci. Merci Mr Le Botlan et Mme Huguet, d'avoir su libérer de votre temps pour m'écouter raconter cette histoire.

Table des matières

Introduction

Je suis entré à l'INSA de Toulouse en 3ème année après une licence de Mathématiques. J'ai décidé de changer d'orientation après un premier semestre commun Mathématiques-Informatique, pour continuer en préorientation Informatique puis en 4ème année Informatique et Réseaux. A la fin de la 4ème année, je suis parti au Brésil en août 2015 pour effectuer la mobilité requise pour le titre d'ingénieur de l'INSA pour un accord d'échange à l'UNICAMP, université d'Etat de la ville de Campinas, de 6 mois initialement, j'ai transformé cet accord en double diplôme fin 2015 et j'ai continué à étudier à l'UNICAMP jusqu'à l'obtention de mon titre d'ingénieur brésilien en juillet 2016. C'est à ce moment là que je suis rentré en France et ai postulé à une offre de stage de développeur Android pour l'entreprise Liberty Rider à Toulouse, dans le cadre de mon projet de fin d'études. Après un entretien visiophonique et une première rencontre le sujet du stage fut décidé général : « Développement Android ». Puisque Liberty Rider était à mon arrivé une entreprise dôté d'un effectif de 10 personnes et portant des projets de développement de 2 à 3 semaines, il n'était pas pensable de positionner une personne pendant 5 mois sur un unique projet. Je pris effectivement part à de nombreux projets ainsi qu'à la maintenance de l'application. Ce rapport présente dans un premier temps la structure d'accueil, son organisation et son historique ; aborde dans un second temps son organisation du travail et son architecture logicielle et finit par la présentation d'un projet que j'ai porté durant une partie de ce stage.

Chapitre 1

La startup

Liberty Rider est une startup créée autour de l'application mobile du même nom, qui a pour objectif prévenir les secours lorsqu'un motard chute. Le motard lance la détection d'accident sur l'application avant son départ, et commence à rouler. En cas de chute - que l'application détecte via l'accéléromètre du téléphone et qui est confirmée après suivi de position et tentative d'appels infructueux au motard - les secours sont prévenus avec en prime la localisation. Le temps entre la chute du motard et l'appel aux secours est au maximum de cinq minutes. Autour de la détection d'accident, qui est le coeur de l'application, des fonctionnalités secondaires ont été développées.

- **Le suivi de trajet** donne la possibilité de partager son trajet en temps réel à ses proches, et de les prévenir, lorsque le motard termine la détection, de son bon déroulement.
- **L'historique de trajets** permet d'archiver et de revoir ses précédentes sorties.
- **La fonction balade** connecte plusieurs utilisateurs, les aidant à communiquer simplement lors d'un trajet de groupe et à suivre la position des participants.
- **Le carnet d'entretien** rappelle les différentes révisions à effectuer sur la moto et archive les révisions passées.

En plus de ces fonctionnalités, l'application a un aspect ludique : plus un utilisateur roule, plus il engrange une monnaie virtuelle qu'il pourra par la suite échanger contre des cadeaux dans la boutique en ligne.

1.1 Lean startup

« A startup is a human institution designed to deliver a new product or service under conditions of extreme uncertainty. »

Cette citation est d'Eric Ries un entrepreneur américain responsable de la création du concept de *Lean Startup*. C'est le modèle sur lequel a été construit Liberty Rider et continue de l'être.

L'idée est simple : privilégier le besoin plutôt que la technologie, mettre en production

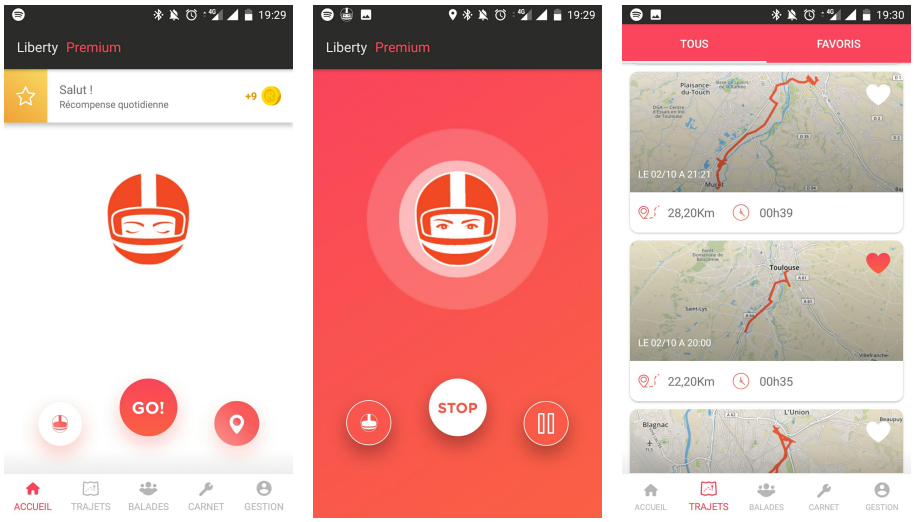

FIGURE 1.1 – Quelques écrans de l'application

le plus vite possible un produit qualifié de *Minimum Viable Product*, c'est à dire un produit basique répondant au besoin de la manière la plus simple possible, l'améliorer un peu après l'avoir confronté avec le consommateur et recommencer. Dans une entreprise telle que Liberty Rider, cela correspond à sortir une application très basique, analyser le ressenti utilisateur, ajouter des petites améliorations répondant aux besoins, mettre à jour l'application, et recommencer.

1.2 Histoire : pré-seed et seed

Liberty Rider est créée en juin 2016. C'est le fruit de la rencontre entre EMMANUEL PETIT, MARTIN D'ALLENS, JÉRÉMIE FOURMANN et JULIEN LE, qui occupent respectivement les postes de CHIEF EXECUTIVE OFFICER, CHIEF TECHNICAL OFFICER, CHIEF SCIENTIFIC OFFICER et CHIEF MARKETING OFFICER. Elle commence sans autre ressources que ses quatre fondateurs qui ne la supportent qu'avec leurs économies ou en s'engageant personnellement en faisant un prêt d'honneur. Il faut valider son marché, et même si la croissance de la startup est minime, elle est tout de même existante, et s'il est possible de réduire les frais, il y en a toujours et de plus en plus : c'est la période de pré-seed. La réduction de frais se fait par l'embauche de stagiaires car ils sont moins coûteux, et par l'utilisation d'outils gratuits ou payant uniquement lors de leur utilisation, donc peu chers. Pour prendre en charge les frais restants, il faut courir après les financements : les différents tremplins et concours permettent parfois de se financer en récupérant quelques milliers d'euros quand on en est lauréat, et toujours de se faire un réseau et de bénéficier d'une exposition médiatique. Liberty Rider a été entre juin 2016 et juin 2017 lauréat du *trophée des services innovants*, du *trophée de l'économie numérique*, du *prix innovation sécurité routière*, du *prix Moovjee* et du *councours Jeune Pousses*. Par ailleurs,

l'équipe a aussi organisé une campagne de financement participatif (crowdfunding) sur la plateforme *KissKissBankBank* permettant de collecter plus de 18k€ en décembre 2016, et compte sur les subventions des différentes entités territoriales. Grâce à ces aides financières, l'application prend forme et la startup peut se permettre d'embaucher ses premiers employés, un *lead* développeur par plateforme (Android, iOS et Web) en novembre 2016. Après cette période de pré-seed vient la période de seed. C'est la première levée de fond à proprement parler, dans laquelle l'entreprise dilue son capital une première fois et s'engage à atteindre certains objectifs moyennant un budget déterminé, auprès d'acteurs tels que des compagnies d'assurance, des banques ou des entreprises. Liberty Rider lève 1.6 millions d'euros en octobre 2017 auprès de la Matmut, d'Inter Mutuelles Assistance, de la Macif, de la Mutuelle des Motards ainsi que de Racer.

1.3 Organisation

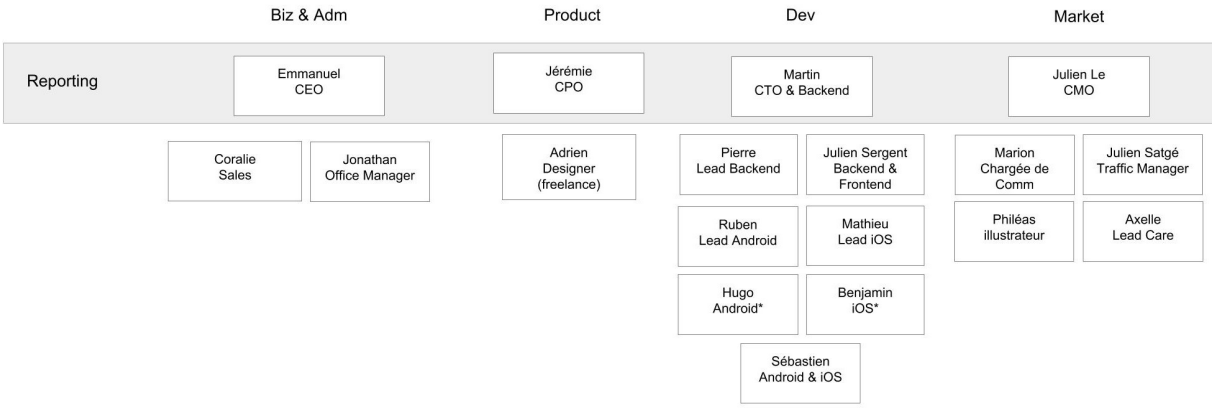

FIGURE 1.2 – Organigramme de Liberty Rider

Les quatre fondateurs occupent quatre postes importants de l'entreprise. Emmanuel est CEO, il est celui qui créé la vision de Liberty Rider, il décide des directions macroscopiques, il imagine l'avenir à moyen et long terme. Il en parle au CPO - Jérémie à mon arrivée - qui se charge de créer une suite de projets à court et moyen termes, permettant de s'approcher au mieux de cette vision. Le CPO est aussi très proche des utilisateurs et est le premier représentant de leurs besoins au sein de l'entreprise. Julien, le CMO est aussi très proche des utilisateurs, mais dans l'autre sens. Il s'occupe de la communication et du marketing qui vont être mis en place pour atteindre le marché. Martin enfin, le CTO, est le responsable de la technique. Il est celui qui aura le dernier mot sur le choix d'une technologie ou d'une autre, et aide le CPO à comprendre les besoins techniques de chaque *user-stories* - histoires permettant de résumer un besoin fonctionnel de l'utilisateur - et donc de les prioriser.

1.4 Business Model & vision

Le modèle économique de Liberty Rider est une application freemium B2C (Business-to-Client) : l'application est gratuite en globalité, mais quelques fonctionnalités sont débloquées par le paiement d'un abonnement mensuel appelé «Premium». Par exemple : tout utilisateur peut partager son trajet avec ses proches, mais sans souscrire à l'abonnement, il ne pourra voir que ses cinq derniers trajets. Un utilisateur premium pourra avoir accès à tout son historique. Si Liberty Rider a levé 1.6 millions d'euros, son futur n'est pas assuré pour autant. Elle a uniquement de quoi subsister jusqu'à début 2019. Cette levée va permettre dans un premier temps de recruter afin d'améliorer l'application, et de donner plus d'outils au marketing et à la communication. Ceci pour au mieux devenir rentable, et au moins créer un modèle intéressant pour les actuels et futurs partenaires

afin de réaliser une seconde levée : la série A. Lors de cette série, nous devrons justifier d'avoir atteint les objectifs fixés lors de la seed, et proposer un modèle de business scalable et potentiellement rentable. Cette levée pourra permettre par exemple de s'orienter vers des marchés étrangers. Entre temps, la course à la rentabilité s'effectuera encore par le biais des ventes B2C, mais aussi par un nouveau moteur : la vente B2B (Business-to-Business). L'idée est de proposer des packs d'abonnements à des professionnels de la moto (assureurs, constructeurs, vendeurs d'accessoires) à un prix inférieur à celui proposé sur les différents stores. Ils pourront alors revendre ces abonnements à leurs clients, par exemple sous forme de bonus lors de la souscription à une assurance, d'une carte cadeau lors de l'achat d'un casque... En terme de vision fonctionnelle, nous souhaitons devenir une référence d'application motarde en étoffant le panel de fonctionnalités actuellement proposées, et en travaillant l'aspect communautaire de l'application.

Chapitre 2

Architecture

Lors de mon stage, la compréhension de l'organisation de Liberty Rider, que ce soit de l'application ou de la startup, fut une étape épineuse mais nécessaire à mon intégration dans les différents projets et à mon efficacité. J'aborde ici dans un premier temps l'organisation des interactions entre les différentes équipes, puis l'architecture de l'application elle-même.

2.1 Organisation humaine

J'ai pu voir une certaine différence entre les process utilisés lorsque je suis arrivé et ceux mis en place à la fin de mon stage. Pas seulement grâce aux recrutements de personnes ayant une bonne expérience des différentes méthodologies de gestion de projet, mais aussi par de continuelles remises en question permettant d'améliorer notre façon d'utiliser nos outils. La première partie de cette section décrira les principaux outils que nous utilisons pour gérer les différents projets, puis nous aborderons l'organisation du workflow comme il était à mon arrivée ; pour enfin voir l'apport des méthodologies utilisées lors du dernier mois de mon stage.

2.1.1 Outils

Les principaux outils que nous utilisons sont brièvement décrits ici :

— **Asana** est une application (web principalement) permettant d'aider la communication entre les différentes équipes au sein d'une entreprise, et entre les différentes personnes au sein d'une équipe. Elle a été créée par Dustin Moskovitz et Justin Rosenstein respectivement co-fondateur et lead ingénieur chez Facebook. Elle permet par exemple le découpage d'un projet en tâches, l'assignation de ces tâches à des personnes et le suivi d'avancement de ces tâches via des labels (Todo, Doing, In review...).

— **Slack** est une application web et mobile de communication interne, similaire à un chat IRC (les échanges sont divisés en channels qui correspondent à un sujet de conversation). Il est alors facile de retrouver les traces de ce qui a été dit. Slack est l'acronyme de « Searchable Log of All Conversation and Knowledge », qui résume bien son intérêt.
— **Zendesk** est une interface de support, permettant de communiquer avec les utilisateurs et de garder une trace de ces communications.
— **Zeplin** est une plateforme permettant de stocker et de décrire les maquettes d'une application. Chaque mock-up y est entreposé ainsi que les détails de ses différents éléments (couleur d'un fond, taille et fichier source d'une image, police d'un texte...).
— **L'interface administrateur** est une fabrication maison. Elle regroupe les différentes données des utilisateurs sur les différentes bases de données existantes, et permet par exemple d'afficher les trajets de n'importe quel utilisateur.
— **Fabric** est une application permettant de diagnostiquer le bon comportement de l'application. Elle répertorie les différents crashs des utilisateurs et peut permettre aux développeurs d'être alertés lors d'une mise en production amenant des bugs, ainsi que de comprendre l'origine de ces bugs.
— **Amplitude** est une plateforme listant les analytics - marqueurs permettant de suivre le parcours d'un utilisateur au sein de l'application - envoyés par l'application et permettant de suivre des funnels - série d'évenements ayant une relation de cause à effet et permettant de comprendre les raisons et les étapes d'un choix fait pas l'utilisateur. Elle est aussi utilisée en parallèle avec fabric pour diagnostiquer les bugs.

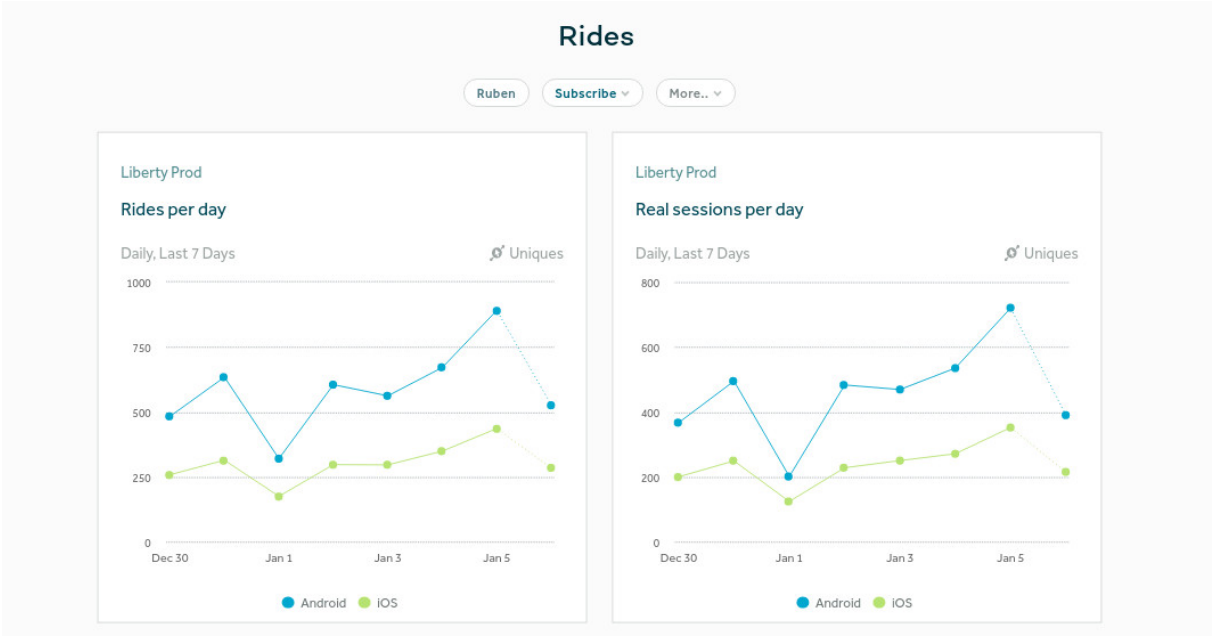

FIGURE 2.1 – Comportement utilisateur sur Amplitude

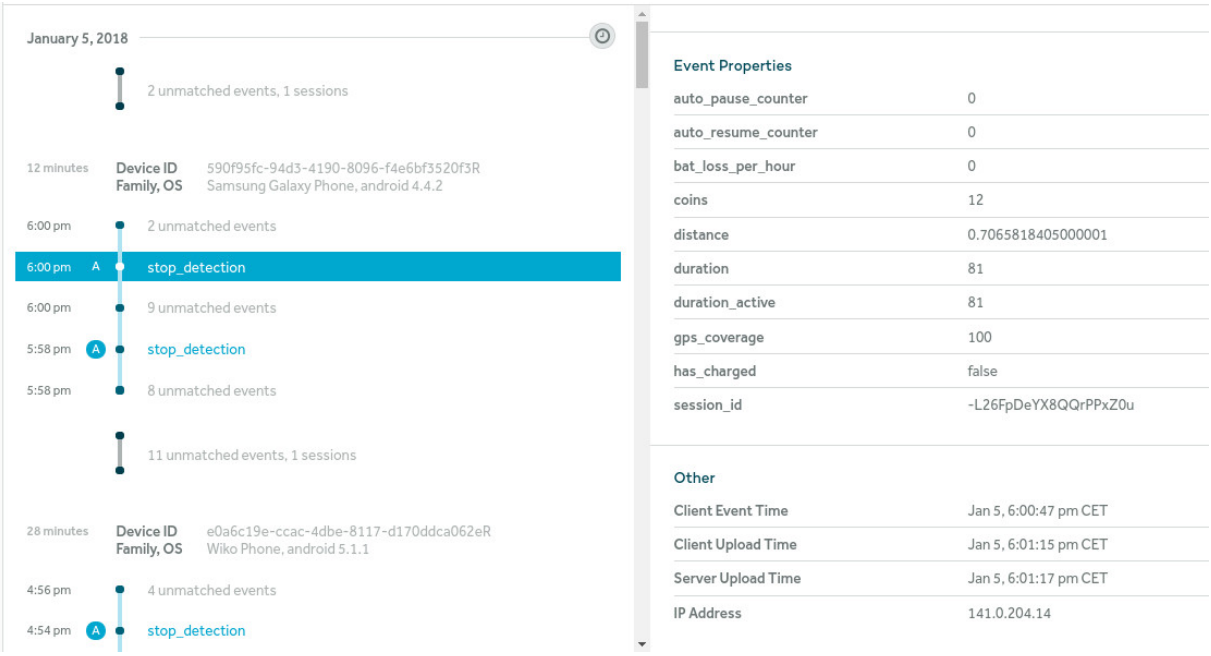

FIGURE 2.2 – Analytics d'un utilisateur sur Amplitude

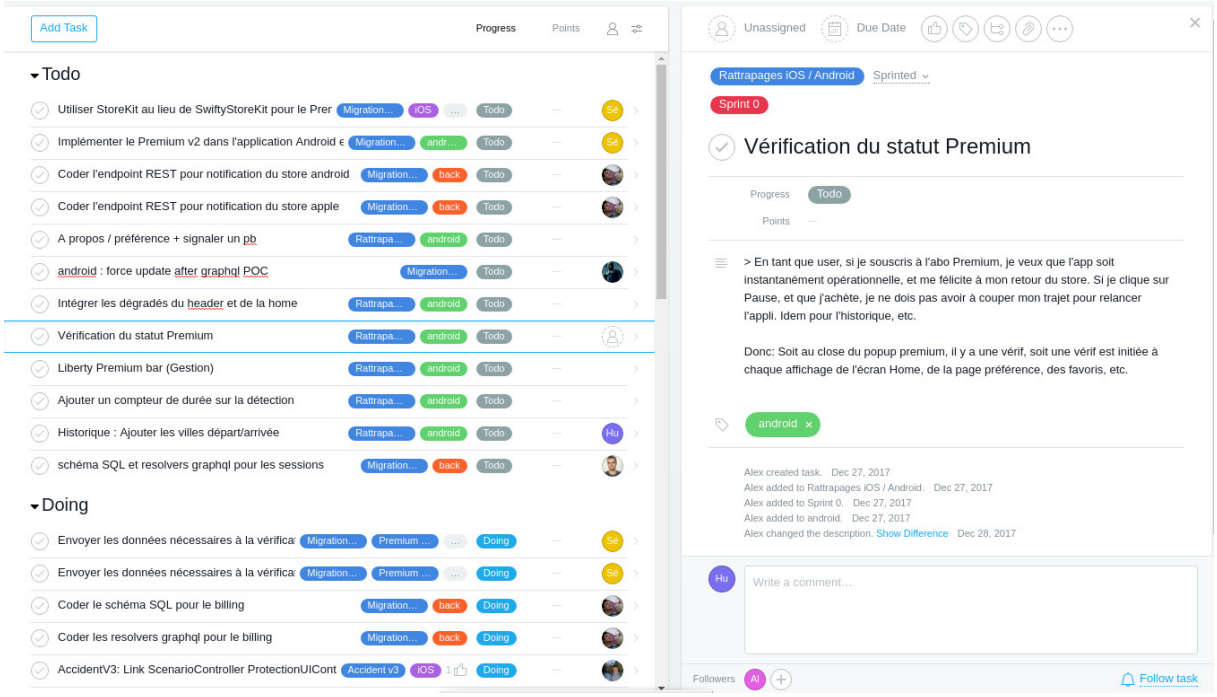

FIGURE 2.3 – Liste de tâches sur Asana

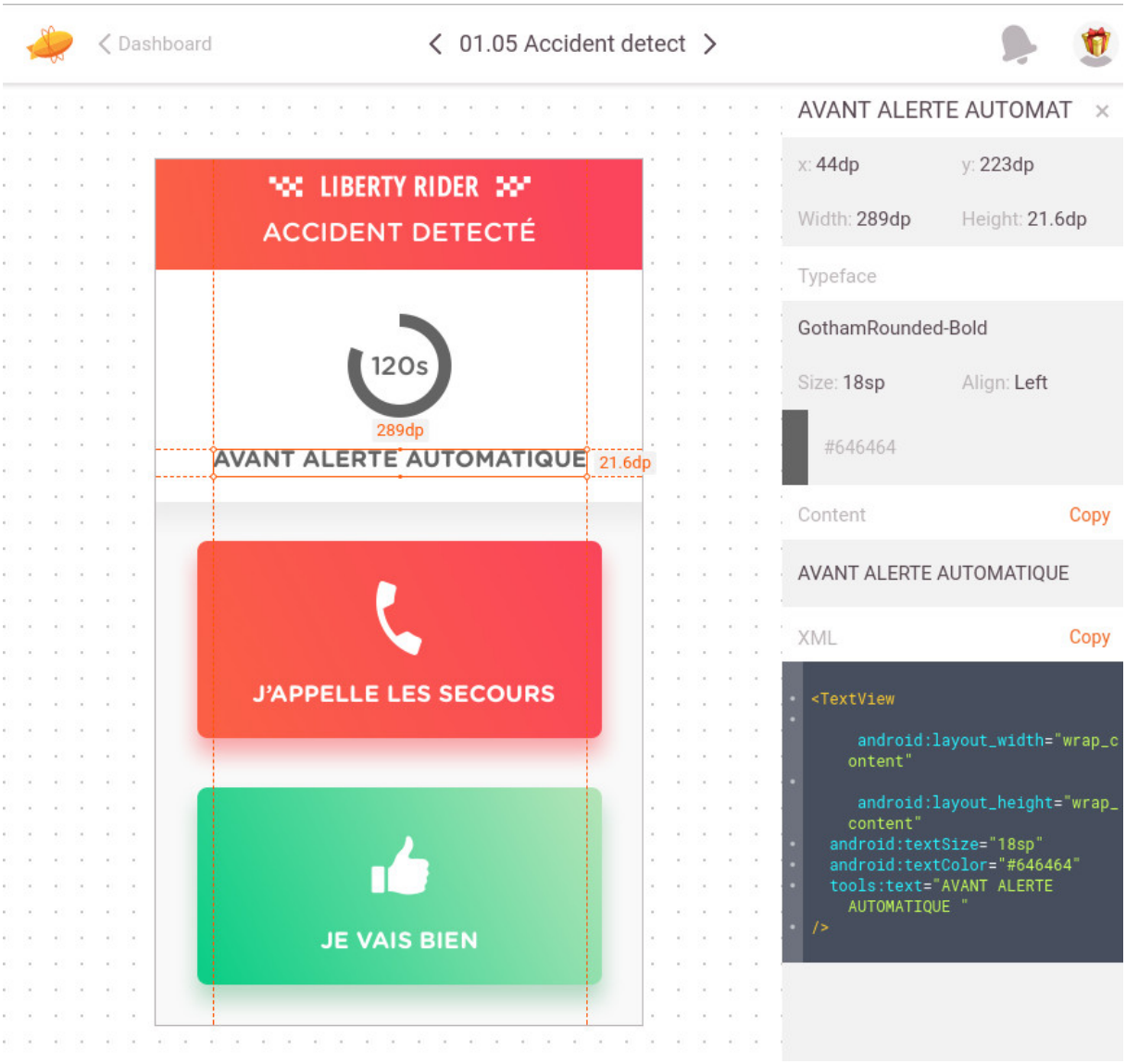

FIGURE 2.4 – Maquette de l'écran d'accident sur Zeplin

2.1.2 Workflow général

La **Technique** interagit avec de nombreuses entités :

— prévient la **Communication** et le **Marketing** des features à venir pour que la startup parle efficacement à ses utilisateurs et à son marché.

— parle continuellement avec le **Care** pour connaitre les bugs utilisateurs et expliquer leur origine.

— reçoit du **Design** les vues et le comportement de chaque nouvelle feature, fait des retours et défend les patrons de conception recommandés par sa plateforme comme par exemple le Material Design de Google, un guide aidant à créer une application jolie et fonctionnelle.

— répond au besoin fonctionnel du **Produit**, qui représente l'utilisateur au sein de la startup, en le décomposant en besoins techniques puis en l'implémentant.

Voici les details du workflow technique. Le CPO décide, après discussion avec le CEO, des nouvelles fonctionnalités à implémenter. Il convoque une réunion avec les partis prenant : Chef de Projet ainsi que Technique, Design et Marketing et décrit les différentes taches permettant de réaliser les objectifs du projet. Le développeur fait son travail sur les projets qui lui sont attribués, puis rend compte en réunion de son avancée. Le CPO orchestre le tout en fixant les échéances de chaque tâche.
C'est de cette manière que nous fonctionnions la quasi totalité de la semaine, exception faite pour vendredi. Vendredi c'est les bugs : tous les développeurs se concentrent sur les problèmes remontés tout au long de la semaine par le Care et tentent de les résoudre, ou au moins de les diagnostiquer.
A une échelle microscopique, voici une description du workflow pour les différents dépôts de code.Chaque dépôt de code est basé sur deux branches principales : Integration et Staging. Integration est un dépôt qui regroupe les dernières modifications devant partir en production. Staging est une copie de la production, elle permet de faire des réparations rapide du code de production si Integration diffère du code de production.
Lorsqu'un développeur souhaite implémenter une feature, il crée une branche à partir d'intégration, il développe son code en séparant en plusieurs commit, puis crée une pull request de sa branche vers intégration qui sera revue par les autres développeurs de la plateforme puis mergée après d'éventuelles modifications. Une fois que suffisamment de branches ont été mergées sur integration, une pull request est créée d'integration vers staging, puis l'application est créée à partir du code de staging et mise en production sur le store, d'abord en bêta que seulement quelques utilisateurs, les "beta-testeurs", vont utiliser, puis pour tous les utilisateurs.

2.1.3 Différence post-seed

Dans les faits, le modèle décrit ci dessus ne fonctionne pas de la meilleure des manières. Il arrive souvent que les projets prennent du retard, parce que le développeur et le Produit ne s'étaient pas bien compris ou que la quantité de travail à effectuer n'a pas été estimée correctement. Quand une partie du projet prend du retard, tous les projets en prennent aussi, l'interdépendance des taches et des projets n'aidant pas.
Après la levée de fond Alexandre arriva pour soulager Jérémie de sa fonction de CPO, qu'il occupait en plus de son poste de CSO, apportant avec lui une nouvelle manière de fonctionner. La fin de mon stage vit le début de la mise en place de SCRUM, un schéma d'organisation proche des méthodes agiles. Il se base sur la priorisation des différentes user-stories dans un backlog, leur découpage en tâches techniques et leur distribution dans des sprints. Un sprint est donc un ensemble de tâches à effectuer en un temps limité, deux semaines chez Liberty Rider, et correspondant à quelques user-stories. Les développeurs ne s'occupent (idéalement) que des tâches du sprint pendant les deux semaines et font une

rétrospective à la fin du sprint de ce qui s'est bien et moins bien passé. Puis commence le sprint suivant. L'ensemble de plusieurs sprints forme une Release. C'est le développement d'un ensemble de features cohérentes, ou d'une version majeure en développement logiciel.

2.2 Architecture technique

2.2.1 Outils

Les outils d'un développeur Android ressemblent aux outils de tout développeur. Un IDE : Android Studio qui est un fork d'IntelliJ doté d'outils spécialement créés pour compiler sur le système Android. Il possède par exemple un analyseur de mémoire RAM et d'utilisation CPU, un prévisualisateur d'écran permettant de visualiser une vue tout en la construisant, ou encore un gestionnaire de SDK permettant d'installer des librairies propres à Android.
Git est utilisé comme gestionnaire de version via github.com, et nous utilisons la plateforme d'intégration continue CircleCI pour pouvoir tester les modifications continuellement.

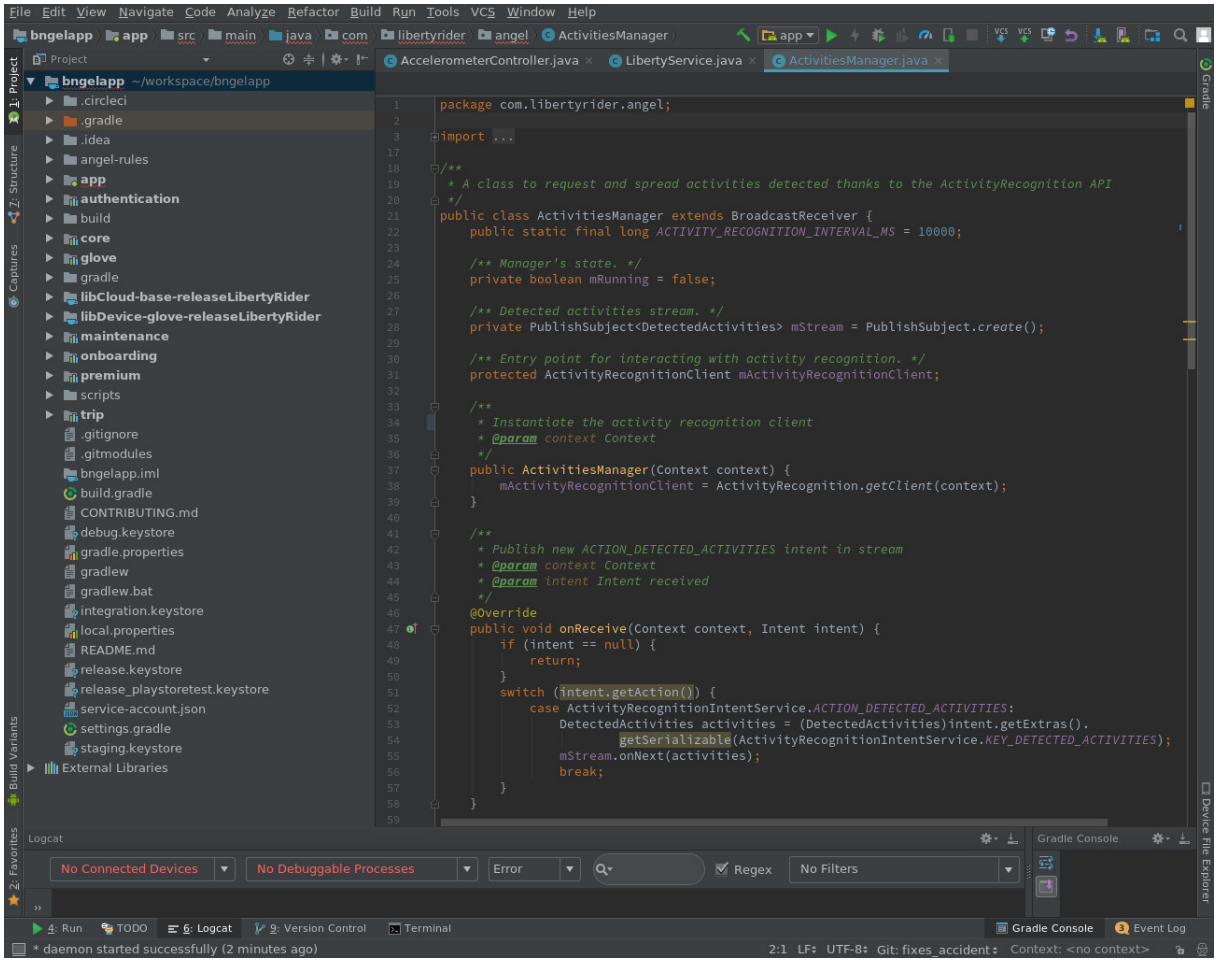

FIGURE 2.5 – Android Studio

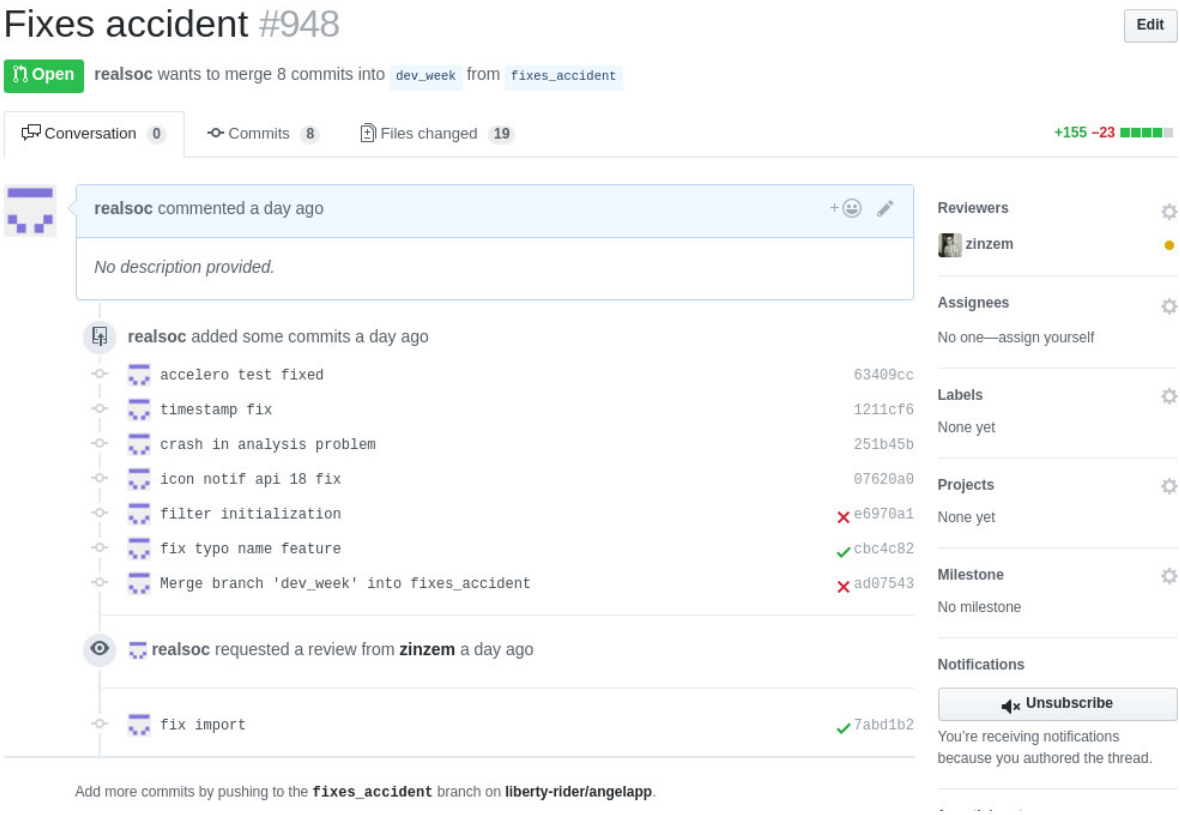

FIGURE 2.6 – Pull request sur github.com

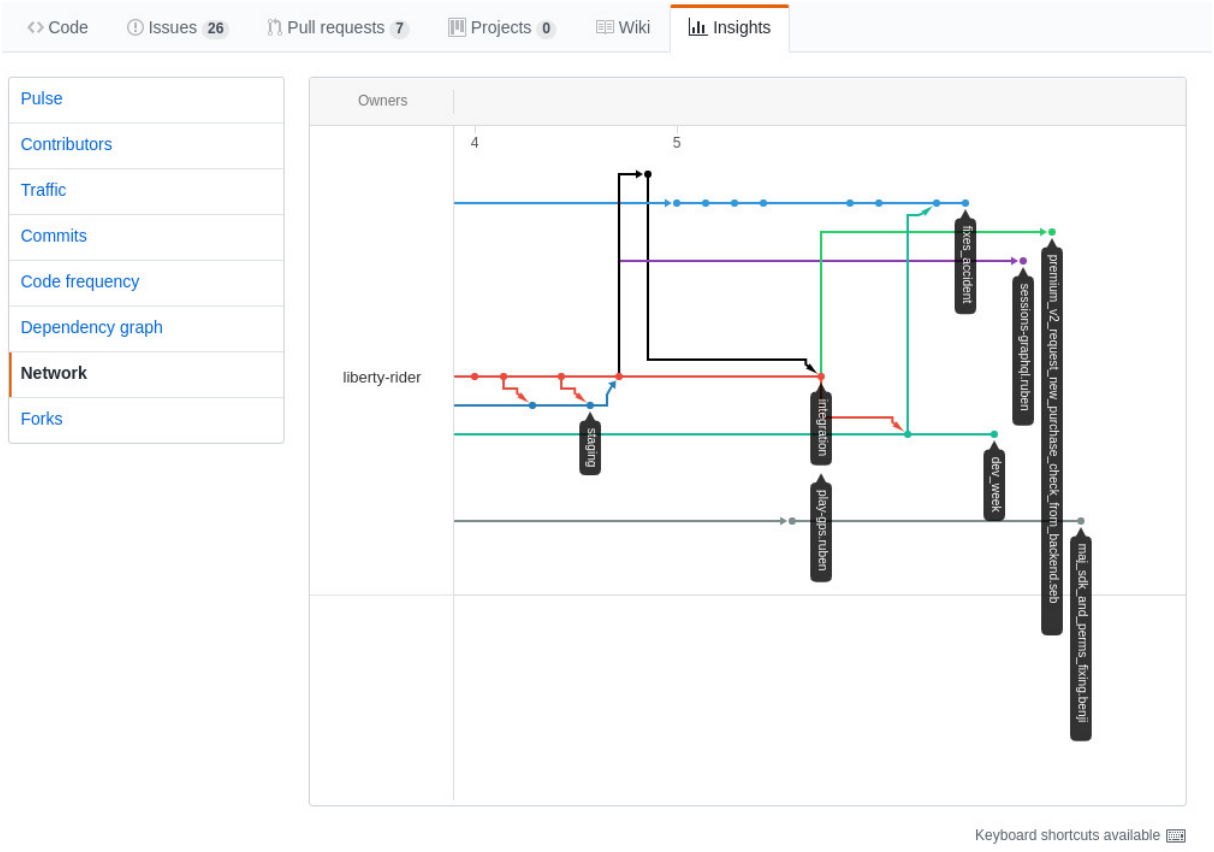

FIGURE 2.7 – Représentation des différentes branches git

2.2.2 Architecture générale

L'application Android peut être décomposée en deux ensembles. Une partie du code correspond à la vue, elle est constituée d'Activités et de Fragments. Une Activité représente une action que peut accomplir l'utilisateur. C'est une classe logique en Java ou en Kotlin associée à un fichier xml qui décrit la vue. L'autre partie correspond à la logique de la détection d'accident, invisible à l'utilisateur. Elle est construite autour d'un Service qui tourne constamment en tâche de fond sur le téléphone. Les vues communiquent globalement ensemble de manière synchrone, par des appels de méthodes, et les différents contrôleurs du service pareillement. La communication d'une vue au service ou du service à une vue s'effectue, elle, de manière asynchrone, par l'envoi d'un Intent, une sorte de message asynchrone contenant une action. Le design-pattern *Model View ViewModel* (MVVM) est fortement utilisé dans l'application. Il permet de séparer les responsabilités de logique et de présentation. La View ne s'occupe que d'afficher des éléments graphiques et de capter les entrées utilisateurs qu'elle transmet au ViewModel qui s'occupe de changer le Model. Lorsque le Model est changé, le ViewModel est prévenu de ce changement de manière asynchrone par le biais d'un autre design-pattern contenu dans MVVM : Observer. Enfin, le ViewModel et la View implémentent du data-binding : le xml lit la valeur de la donnée à afficher dans la mémoire directement, et affiche instantanément la nouvelle information lorsque la donnée change.
L'application n'utilise pas de stockage local tel que des fichiers ou une base de données, tout est stocké sur internet, et pour ce faire nous utilisons le Backend as a Service de Google : Firebase ; construit sur un business model freemium. Via son API Android, il est possible de stocker et lire des données simplement, sans faire d'administration système comme il aurait fallu le faire dans une solution moins clé en main. A l'usage, il se trouve que nous avons rapidement dépassé le stockage limite de la version gratuite et que la montée en charge du nombre d'utilisateur a rendu l'utilisation compliquée, causant des ralentissements et empêchant des opérations simples et nécessaires comme par exemple le tri des utilisateurs. Et pour cause : nous dépassions de 1000% la limite de stockage recommandée par Firebase.

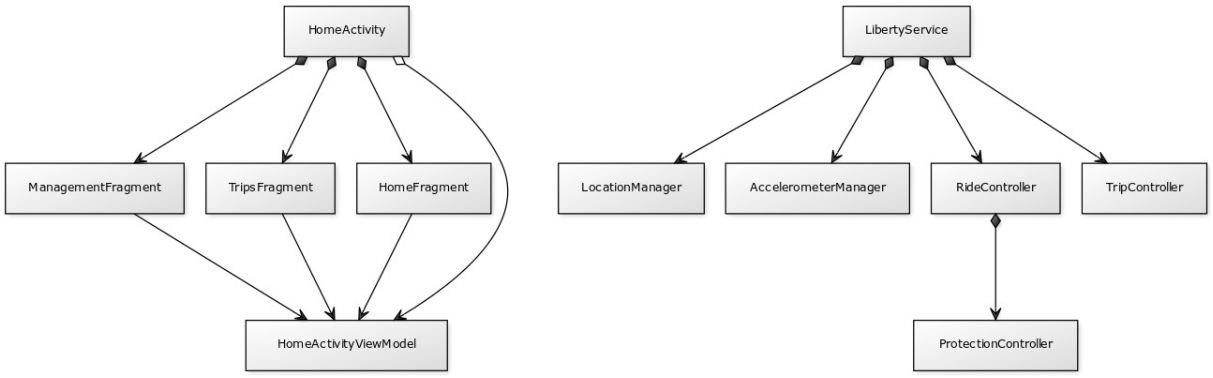

FIGURE 2.8 – Représentation non exhaustive des classes de l'application

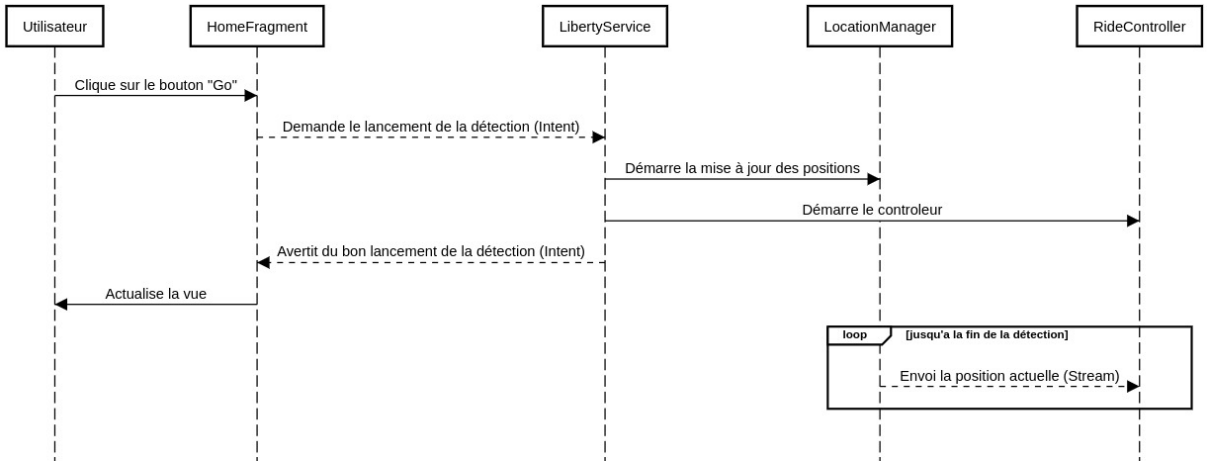

FIGURE 2.9 – Communication interne entre les différentes entités

FIGURE 2.10 – Simple représentation d'un MVVM en diagramme de classe

2.2.3 Architecture post-migration

Pour ne plus se heurter aux limites de Firebase nous avons décidé de migrer pour un serveur bien à nous doté de sa propre base de données. La base de données sera gérée par PostgreSQL, un système de gestion stable et open-source, et les requêtes entre l'application et les différentes APIs seront faites en GraphQL. GraphQL est un langage de requête ayant pour principales caractéristiques :

— Le client décrit la donnée qu'il veut recevoir, i.e. la réponse du serveur doit posséder le même format que la requête du client.

— Si plusieurs types d'objets avec des attributs peuvent être liés, les liens (edges) entre différents objets (nodes) possèdent eux aussi des attributs. Ce langage permet de servir les données comme un graphe.
— Il est possible de définir et d'utiliser des types complexes pour les opérations de lecture (query) comme d'écriture (mutation).

Et ces caractéristiques amènent trois principaux avantages par rapport à son concurrent principal qu'est le REST :

— Le client ne reçoit que ce dont il a besoin. S'il souhaite ne récupérer qu'un seul attribut d'un objet qui en possède des milliers, il ne recevra en réponse que cet attribut ; contre tout l'objet en rest.
— Le client reçoit tout ce dont il a besoin. GraphQL permet d'économiser les appels à l'API. C'est le serveur qui se charge de construire l'objet que demande le client, en allant chercher dans plusieurs tables de sa base de données si nécessaire.
— Il permet d'utiliser de la pagination, permettant de lire un très grand volume de donnée petit à petit, en fonction du besoin de l'utilisateur.

Exemple de requête du client :

```
{
  human(id: "1000") {
    name
    height(unit: METER)
  }
}
```

Et la réponse du serveur :

```
{
  "data": {
    "human": {
      "name": "Anakin Skywalker",
      "height": 5.6430448
    }
  }
}
```

NB : Il est en effet possible de passer des arguments à chaque champ.

Chapitre 3

Le développement d'une feature

Dans une entreprise de cette taille, un développeur se voit souvent confié l'entièreté de l'implémentation d'une feature sur sa plateforme et en tant que lean start-up, les features s'enchainent à bon rythme. J'ai pu au cours de mon stage contribuer à l'application par l'implémentation de plusieurs features : le changement et l'upload de photo de profil, la suppression de trajet, la refactorisation du profil moto, l'implémentation d'un menu développeur permettant le debug de l'application sur les téléphones de test... Mais aussi par les corrections de bugs et par des projets moins visibles par l'utilisateur comme la migration de base de données, ou la refonte de l'algorithme de détection de chute. J'ai par ailleurs eu l'opportunité d'être chef de projet à deux reprises : d'abord lors du projet d'optimisation de batterie, puis lors du projet de la feature pause. J'aborde ici le déroulement d'un projet d'un point de vue du chef de projet d'une part et de développeur d'autre part puisque j'étais aussi la principale ressource sur la plateforme Android lors de ce projet. *Pause* est un projet en deux parties. La pause manuelle est une feature permettant à un utilisateur de mettre en pause son trajet par le biais d'un bouton sur l'écran principal, l'objectif est de ne pas vider inutilement la batterie du téléphone d'un utilisateur qui s'arrête au milieu de sa balade pour manger ; pause automatique est une feature un peu plus intelligente qui est censée anticiper les pauses en analysant le comportement de l'utilisateur via les différents capteurs du téléphone et ainsi déclencher la mise en pause automatiquement. Les deux features sont destinées à une utilisation par les membres premiums uniquement.

3.1 Kick-off meeting : Le lancement

La première phase du projet est son lancement. Il peut être décomposé en deux étapes : *le pré kick-off*, et le *kick-off meeting*. Le pré kick-off est la première réunion entre le CPO et le chef de projet. Avant cette réunion, le chef de projet ne connait à priori rien du projet. Les sujet abordés sont principalement les ressources qui vont être allouées au projet ainsi que le temps affecté à chacune, et bien sur les objectifs du projet. Dans le pré kick-off de pause l'objectif fut présenté comme des réponses à trois questions :

— **Quoi ?** Pouvoir mettre en pause la détection pendant un trajet.

— **Pourquoi ?** Avertir les proches que je ne suis pas en train de rouler, réduire la consommation d'énergie, ne pas couper les sessions en deux dans l'historique, ne pas comptabiliser le temps de pause dans la durée affichée dans l'historique.

— **Comment ?** Manuellement par le biais d'un bouton sur la page principale, automatiquement avec un algorithme détectant les pauses et les reprises de trajet.

Les ressources mises à disposition sont :

— Moi sur Android avec 10 jours-hommes de développement.

— Adrien sur la partie design avec 5 jours-hommes.

— Mathieu sur iOS avec 5 jours-hommes.

— Julien sur la partie web avec 1 jour-homme.

— Jérémie en tant que CPO, 1 jour-homme.

Suite à cette réunion le chef de projet découpe le projet en tâches. Ici le découpage est :

— Réaliser les maquettes de la pause manuelle, 1 jour.

— Implémenter la pause manuelle sur Android, 3 jours et demi.

— Implémenter la pause manuelle sur iOS, 2 jours et demi.

— Designer l'algorithme de pause automatique, 3 jours.

— Implémenter l'algorithme de pause automatique sur Android, 3 jours et demi.

— Implémenter l'algorithme de pause automatique sur iOS, 2 jours et demi.

— Implémenter les modifications sur l'interface Web, 1 jour.

Il créé alors un diagramme de Gantt pour présenter visuellement les périodes de travail sur chaque tâche et la date de rendu des livrables. Il réunit ensuite les ressources dont il dispose pour valider avec elles le Gantt (validé avec le CPO au préalable), les différentes tâches et le temps alloué. Après discussion, la réunion se termine idéalement avec une répartition claire des tâches pour toutes les personnes impliquées.

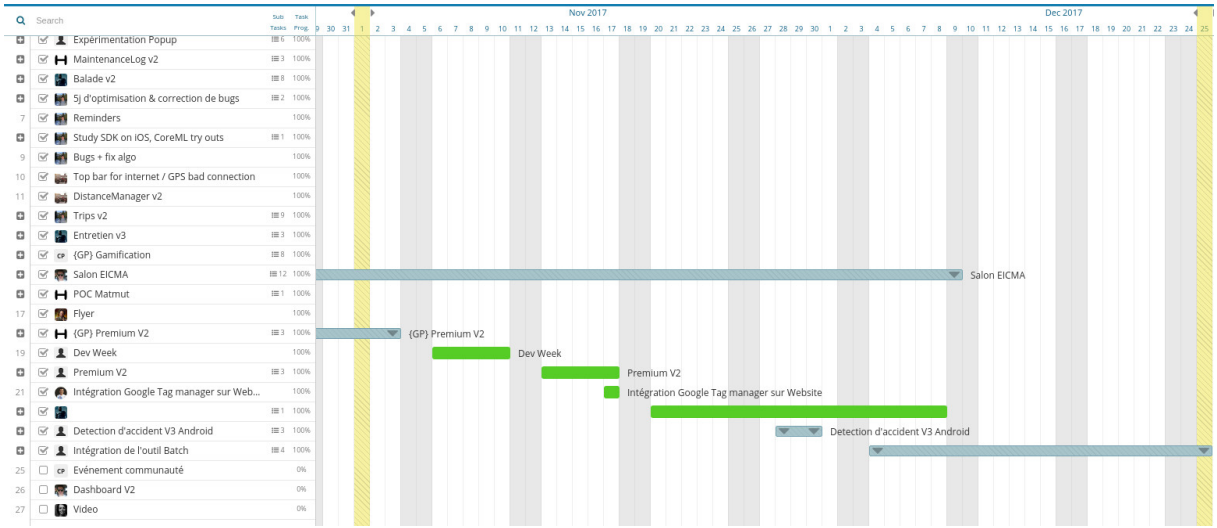

FIGURE 3.1 – Gantt de Liberty Rider

3.2 Design de la pause manuelle

La première tâche était la réalisation de maquettes pour la pause manuelle. Elle parait bloquante pour l'implémentation, mais ne l'est que pour la réalisation de l'interface graphique. Le coeur du code étant la mise en pause effective de la détection - ne nécessitant pas d'interface graphique - j'ai pu m'y atteler avant d'avoir le premier livrable d'Adrien. Ce livrable arriva le lendemain. Et c'en fut fini pour la partie UI/UX de la pause manuelle.

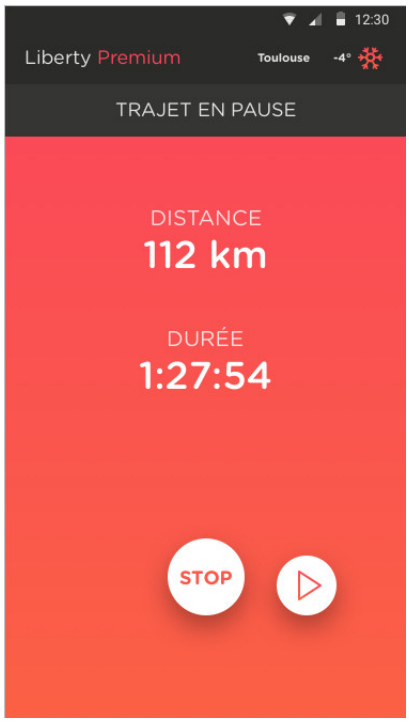

FIGURE 3.2 – Maquette de l'écran de pause

3.3 Développement

Le code de l'application est donc décomposable en deux grandes parties : les vues formées par l'ensemble des activités et des fragments, et le service assurant la survie en foreground de l'application et les calculs nécessaires pour la détection d'accident ou la gestion des balades, effectués via de nombreux contrôleurs. Pour la pause manuelle, l'idée générale est la suivante : l'utilisateur appuie sur un bouton dans une des vues ; l'information est envoyée au service via un intent, afin de mettre les différents contrôleurs en pause ; un dernier intent est envoyé à la vue par le service pour confirmer la mise en pause ; la vue pause est affichée. Certains contrôleurs contiennent l'état du trajet et doivent être maintenus en vie pour ne pas le perdre, sachant que cette information est gardée en RAM sous l'état de variables, et que le cycle vie des variables est directement lié à celui des classes dans lesquelles elles sont contenues. D'autres contrôleurs ne contiennent rien d'important si la détection n'est pas activée ou contiennent un état pouvant être enregistré

dans d'autres contrôleurs. Plus concrètement :

— La vue ne subit pas de grands changements : seulement une cinquantaine de lignes dans le fichier xml dédié à la vue de l'écran de détection et quelques lignes de Java pour capter l'appui du bouton pause par l'utilisateur et transmettre l'information au service, quelques lignes de plus pour recevoir la réponse du service et demander le changement de la vue.

— Au niveau du service c'est un peu plus compliqué, il faut mettre en place le mécanisme d'envoi et de réception d'intent pour communiquer avec la vue, mais surtout mettre en pause, stopper ou réactiver les différents contrôleurs, en sauvegardant et restaurant par la suite leur états pour ne pas perdre d'information.

3.4 Design de la pause automatique

Le principe de la pause automatique est donc de mettre en pause automatiquement la détection à partir d'informations fournies par le téléphone. Après discussion avec le CPO, les différents scénarios à couvrir par cette fonctionnalité sont :

Mettre en pause quand

— L'utilisateur oublie de désactiver la détection en rentrant chez lui, après quelques minutes (5 maximum).

— L'utilisateur s'arrête lors d'une balade, pour manger par exemple, après quelques minutes.

— L'utilisateur gare sa moto et commence à marcher, après quelques minutes.

Ne pas mettre en pause quand

— L'utilisateur roule normalement.

— L'utilisateur avance au ralenti ou s'arrête dans des embouteillages.

— L'utilisateur s'arrête à un feu rouge, de durée maximum de $1\frac{1}{2}$ minutes.

— L'utilisateur passe dans un tunnel.

Et dans tous les cas, relancer très rapidement la détection si l'utilisateur retourne sur sa moto.

3.4.1 Idée initiale

Pour concevoir l'algorithme, je me suis vu accorder une liberté totale à la seule condition que Ruben, le lead développeur sur Android, approuve l'architecture. Mon premier choix fut d'utiliser les positions fournies par le GPS du téléphone. Comme l'application les utilise déjà, cela ne consommerait pas plus d'énergie de les utiliser aussi à un autre endroit. Cette stratégie d'utilisation de ressources est appelé "opportuniste" et est beaucoup utilisée par Google dans ses différentes API : si mon application fonctionne mieux avec

des points GPS, mais peut aussi s'en passer, il suffit d'attendre qu'une autre application demande le GPS pour profiter de l'activation du capteur et récupérer les données qui seraient de toute manière générées pour cette autre application. Mais pourquoi utiliser le GPS alors ? Parce que les comportements devant déclencher une pause sont tous des comportements à basse vitesse, et qu'une relation d'équivalence directe existe entre ces points et la vitesse du motard. Mais ce n'était pas suffisant. En effet, il existe de nombreux téléphones dont le GPS ne fonctionne pas bien, ou même des zones dans lesquelles des téléphones avec des puces GPS totalement fonctionnelles ne réagissent pas bien, comme dans un tunnel par exemple. Cela signifie que lorsqu'un motard est dans un tunnel, nous ne pouvons pas connaitre sa vitesse, ni même si il est en mouvement. Une autre zone où un utilisateur peut ne pas être couvert par le GPS est son domicile, en intérieur. Selon la qualité de la puce du téléphone et son environnement (condition météorologique, épaisseur des murs...) un téléphone peut ne plus recevoir de GPS du tout. Donc d'une certaine manière, avec seulement cette donnée, un utilisateur roulant dans un tunnel serait considéré dans le même état qu'un utilisateur rentré chez lui. Il me fallait au moins une autre donnée pour améliorer mon champ des possibles.

3.4.2 Activity Recognition API

Après une discussion avec Ruben et sur ses conseils, je me suis tourné vers l'Activity Recognition API de Google qui est une API proposant de détecter périodiquement les activités de l'utilisateur à partir de capteurs utilisant peu d'énergie (car elle même fonctionne de manière opportuniste). Elle utilise différents capteurs du téléphone : le magnétomètre, l'accéléromètre, le GPS et les réseaux mobile et wifi pour déterminer quel type d'activité peut être en train de faire l'utilisateur. Elle peut détecter jusqu'à 8 activités différentes :

- **IN_VEHICLE** : l'utilisateur est dans un véhicule
- **ON_BICYCLE** : l'utilisateur est sur un vélo
- **ON_FOOT** : l'utilisateur est à pied
- **RUNNING** : l'utilisateur est en train de courir
- **STILL** : l'utilisateur est immobile
- **TILTING** : l'utilisateur a changé d'inclinaison
- **UNKNOWN** : Inconnu
- **WALKING** : l'utilisateur est en train de marcher

A chaque retour de l'API (qui rappelons le, a été appelé périodiquement), le système fournit un objet contenant entre une et huit de ces activités, associée un coefficient de confiance. Par exemple un objet :
* IN_VEHICLE = 60
* ON_BICYCLE = 20
* ON_FOOT = 80

GPS	ACT	DEC
UNK	UNK	NOP (1)
UNK	MOT	NOP (2)
UNK	NMO	PAU (3)
FST	UNK	NOP (4)
FST	MOT	NOP (5)
FST	NMO	NOP (6)
LOW	UNK	PAU (7)
LOW	MOT	NOP (8)
LOW	NMO	PAU (9)

TABLE 3.1 – Tableau représentant les possibles décisions basées sur l'état du GPS et de l'Activité utilisateur

* WALKING = 10
* RUNNING = 2

est envisageable et pourrait représenter par exemple un utilisateur en train de rouler en skateboard.

Cette donnée pourrait permettre de trancher lors d'une indécision entre *utilisateur dans un tunnel* et *utilisateur à la maison* par exemple. Et plus globalement, associer à un objet contenant ce type d'activités les valeurs : *probablement en moto*, *probablement non en moto* et *inconnu*, pourrait permettre de trancher dans le cas d'un GPS peu fiable.

3.4.3 Prise de décision

A partir de cette réflexion me vint l'idée de faire de même pour le GPS. Seulement trois cas de figure existent : le GPS indique un déplacement rapide, il indique un déplacement lent ou il indique un déplacement indéterminé (ou rien, similaire au dernier point). A partir de ces 3×3 informations, je possède un univers de possibilités de taille 9, et je peux associer à chaque couple (GPS, Activité utilisateur) une décision : mettre ou non le trajet en pause.

Dans le tableau ci contre, GPS est pour GPS, ACT pour Activité utilisateur et DEC pour Décision. UNK signifie Inconnue, NOP : Pas de décision, PAU : Pause, FST : Rapide, LOW : Lent, MOT : En moto et NMO : Pas en moto.

A présent, reprenons la liste des comportements devant déclencher la pause ou non et associons chaque comportement au (ou aux) numéro(s) correspondant dans le tableau de décision.

Mettre en pause quand

— L'utilisateur oublie de désactiver la détection en rentrant chez lui : $(3 \mid 9)$ = Pause.

— L'utilisateur s'arrête lors d'une balade, pour manger par exemple, (9) = Pause.

— L'utilisateur gare sa moto et commence à marcher, $(7 \mid 9)$ = Pause.

Ne pas mettre en pause quand

— L'utilisateur avance au ralenti ou s'arrête dans des embouteillages, (8) = Pas de pause.

— L'utilisateur s'arrête à un feu rouge, (8 | 7) = Incertain.

— L'utilisateur passe dans un tunnel, (2) = Pas de pause.

Cette correspondance est ce que j'imaginais mais ne correspond pas forcément à la réalité, et il restait un cas (et potentiellement plus, non abordés) non pris en charge par l'algorithme : le cas du feu rouge. Pour résoudre ce dernier problème, comme un motard arrive à un feu rouge en moto avec une vitesse relativement haute, et repart de la même manière, introduire de l'inertie ou une notion de mémoire me sembla être une bonne idée. Ainsi, son comportement passé "haute vitesse" et "motard" empêcherait l'algorithme de décider trop vite d'une pause. Plutôt que d'utiliser seulement la dernière vitesse connue et la dernière activité connue pour la prise de décision, j'introduis de la bufferisation pour avoir des données sur les dernière minutes sous la forme d'un couple de buffers (de positions et d'activités), à partir desquels seraient calculées les valeurs uni-dimensionnelles (UNK/FST/LOW) et (UNK/NMO/MOT) respectivement.

3.4.4 Décision inertielle

Nous créons une nouvelle classe pour représenter ces buffers. L'objet est initialisé avec une limite de temps passé dans le constructeur. Il contient une liste ordonnée de taille extensible, mais limitée dans le temps. Il n'accepte que des objets estampillés, comparés avec le plus vieil élément de la liste à l'insertion. Si le temps passé entre les deux objets est supérieur à la limite fixée, le vieil objet est supprimé et on continue la comparaison avec le nouveau plus vieil objet de la liste.

Les buffers se remplissent donc au fur et à mesure de l'arrivée des activités et des positions dans le système. Toutes les minutes, l'ActivityController procède aux calculs qui vont permettre de déterminer s'il faut ou non mettre le trajet en pause.

Pour chaque buffer les étapes sont similaires :

— Possède-t-on suffisamment d'informations ?

  — Non : le buffer n'est pas fiable, il correspond à «UNK»

  — Oui : On continue le traitement

— Le buffer correspond-il à un comportement motard («FST» ou «MOT») ou non motard («LOW» ou «NMO») ?

Pour cette raison, je décrirai le procédé dans le détail uniquement pour le buffer de positions.

3.4.5 Traitement du buffer de position

Un buffer de position contient suffisamment d'information s'il contient de nombreuses positions à intervalle régulier. Ou autrement dit : si une grande partie du buffer est couverte par des positions et si la partie couverte est dense. Un coefficient est calculé à partir de la durée entre la première et la dernière position du buffer et sa taille temporelle maximale.

$Coeff_{CouvBuffer} = \frac{\sqrt{PartieCouverte}}{\sqrt{TailleMaximale}}, Coeff_{CouvBuffer} \in [0, 1]$

Un autre coefficient est calculé à partir de l'écart moyen entre deux positions dans la partie couverte du buffer.

$$Coeff_{Densite} = \begin{cases} 0 & \text{si } ecartMoyen \geq 60 \text{ secondes} \\ \frac{0.75}{30-60} \times ecartMoyen + \frac{-0.75 \times 60}{30-60} & \text{si } 30 \leq ecartMoyen < 60 \\ \frac{0.25}{10-30} \times ecartMoyen + \frac{0.75 \times 10 - 30}{10-30} & \text{si } 10 \leq ecartMoyen < 30 \\ 1 & \text{si } ecartMoyen < 10 \end{cases}$$

Si il y a 10 secondes d'écart moyen ou moins le coefficient vaut 1, il vaut 0.75 pour 30 secondes d'écart moyen et 0 au dela de 60 secondes. Il évolue linéairement entre ces points.

La fiabilité du buffer de position est :

$Fiabilite_{Position} = Coeff_{CouvBuffer} \times Coeff_{Densite} \times 100$

Si ce pourcentage vaut 60 ou plus, je considère que le buffer est «fiable» et continue le traitement. Sinon, je passe au traitement du buffer d'activité, pour déterminer quel comportement utilisateur je peux associer à mon comportement GPS inconnu (UNK) et me positionner entre la ligne 1, 2 ou 3 du tableau 3.1.

Si mon buffer de position est «fiable» donc, la suite du traitement est assez aisée : je calcule la vitesse moyenne indiquée par le buffer de position, c'est à dire la somme des distances entre chaque paire de positions consécutive divisée par le temps entre la première et la dernière position, et je la compare à un seuil. Une vitesse supérieur à ce seuil indique une pause, inférieur indique le contraire. Pour décider de cette valeur seuil, des tests sur le terrain ont été effectués. Finalement, la valeur de 12km/h fut la valeur retenue.

Je n'ai abordé ici que la détection de la pause. La reprise de détection étant très simple et basique : reprendre à la moindre position GPS possédant une vitesse supérieure à 10 km/h. En effet si le procédé pour mettre en pause est fastidieux, celui pour reprendre la détection est extrêmement simple puisque l'idée derrière l'algorithme est qu'il n'est pas grave de ne pas détecter une pause alors qu'il est impensable de ne pas détecter une chute.

3.5 Développement

L'implémentation de la pause automatique se résume à la création d'un nouveau manager permettant au système de récupérer les objets représentant les activités (ou detected

activities) et de les mettre à disposition aux différents contrôleurs de l'application, ainsi que la création d'un nouveau contrôleur appelé ActivityController récupérant les positions et les activités, en charge des calculs menant à la décision de mettre en pause la détection automatiquement.

FIGURE 3.3 – Représentation UML de l'ActivityController

Le traitement du buffer de position décrit plus haut est effectué dans la fonction processLocationsForPause(), cette fonction appelle d'abord calculateLocationsReliability() pour définir si le buffer est fiable ou non, fonction qui appelle à son tour calculateReliabilityLocationBufferDuration() et calculateReliabilityLocationsDensity(), puis si le buffer contient suffisamment d'information, la vitesse moyenne est calculée via une classe statique Utils, et un entier est renvoyé à la fonction appelante processPause() : -1 pour un comportement haute vitesse n'indiquant pas une pause, 0 pour un comportement inconnu et 1 pour une pause.
Je fis aussi une partie du développement Web de ce projet en javascript, pour créer la couche affichant les activités et les décisions de pause sur la carte de l'interface administrateur.

3.6 Fin du projet

Et c'est ainsi qu'en quelques semaines, entre la réunion de lancement du projet et la mise effective en production - après une étape de test alpha en interne et bêta à nos utilisateurs les plus dévoués - le projet pause fut conçu, implémenté et propagé. Le projet se termine théoriquement une fois que le code est mis en production par une réunion de conclusion de projet permettant de faire une rétrospective et de soulever les points positifs et les points à améliorer. Mais dans les faits, il y a des bugs à corriger, des seuils à changer et des données à analyser pour pouvoir réellement parler d'une fin de projet. C'est pourquoi il est courant de faire une réunion de rétrospective 3 semaines après la réunion de conclusion de projet. Suivent ci-dessous des captures de l'interface administrateur représentant le trajet d'un utilisateur ayant fait un trajet avec l'algorithme, quelques jours après sa mise en production. Les marqueurs rouges avec un café représentent une pause, les marqueurs verts avec une moto les reprises de détection. Ici, les pauses et reprises sont à titre indicatifs puisque dans un premier temps l'algorithme tourna sans couper réellement la détection, afin de vérifier la justesse de l'algorithme, et le modifier si nécessaire. Dans l'ensemble, le comportement est celui recherché : quand l'utilisateur s'arrête quelques minutes, pour mettre de l'essence ou pour manger par exemple, l'algorithme fonctionne extrêmement bien. Ce que ce trajet montre en revanche est une situation inattendue : la première pause du trajet est réalisée en plein milieu d'une rue de Toulouse alors que rien ne semble indiquer qu'il y a réellement eu une pause. L'explication est simple : les utilisateurs déclenchent la protection jusqu'à quelques minutes avant de commencer à réellement conduire leur véhicule (en sortant de chez eux par exemple). Dans ces cas là, assez fréquents, l'algorithme déclenche une pause quelques secondes après qu'ils commencent à rouler, et ce à cause de l'inertie voulue durant le design permettant de ne pas détecter de pause lors d'un arrêt au feu rouge par exemple. Bien heureusement, la reprise de la protection s'effectue, comme prévu en cas de telle erreur, très peu de temps après la mise en pause.

FIGURE 3.4 – Un trajet Toulouse Bordeaux contenant 4 pauses et 3 reprises de protection

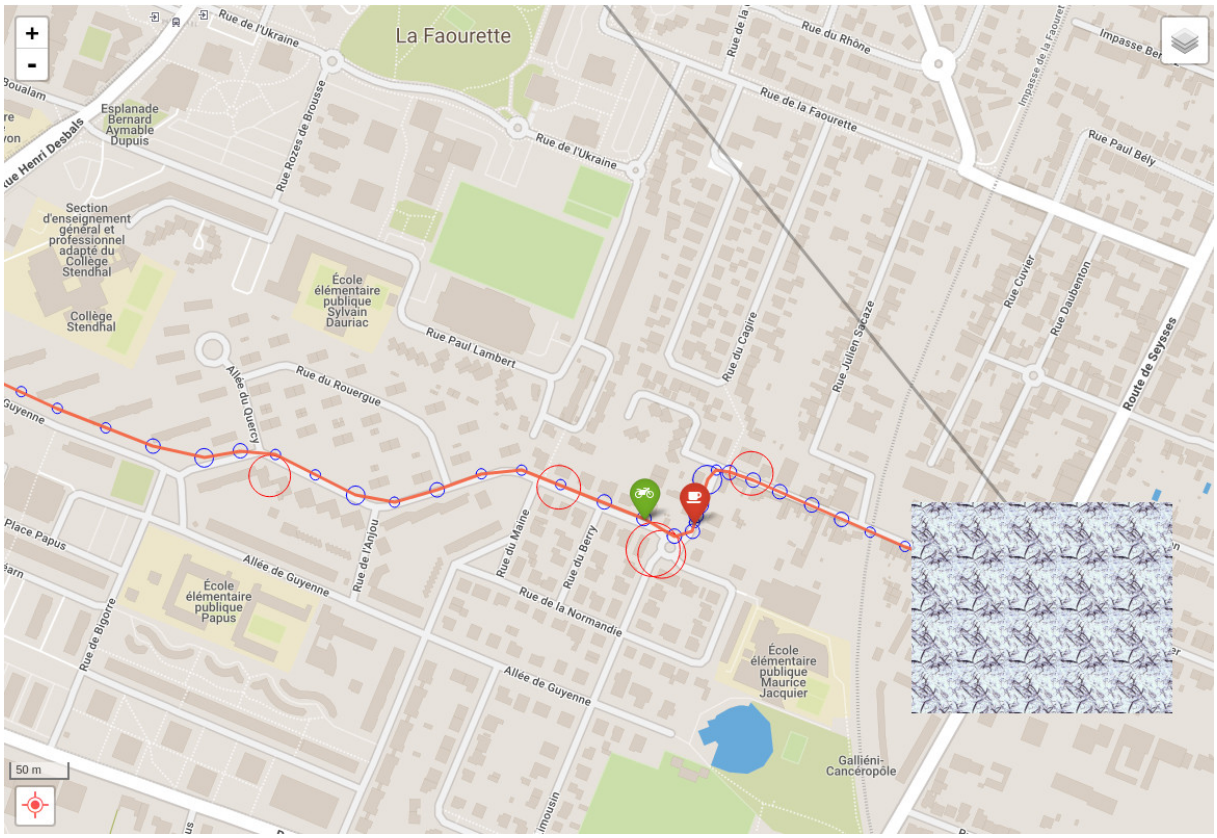

FIGURE 3.5 – Zoom sur la première pause, très peu de temps après le départ

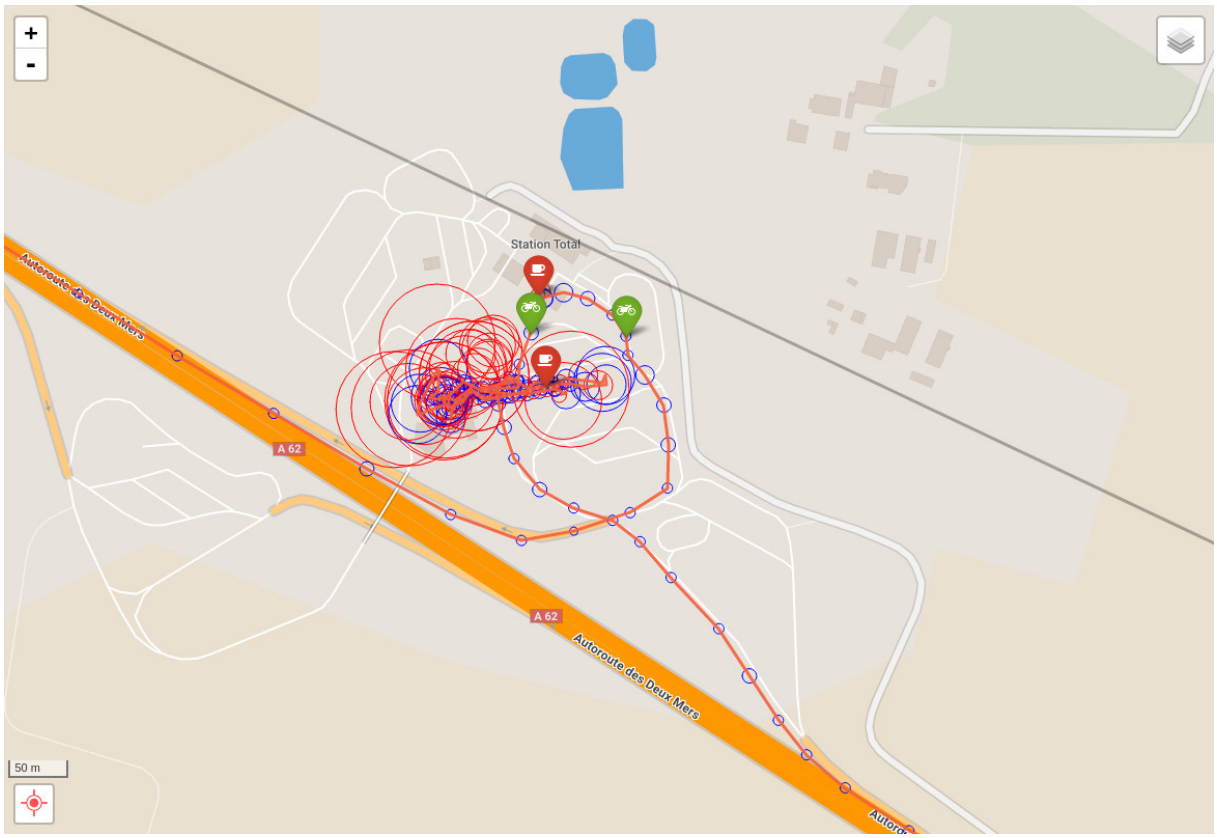

FIGURE 3.6 – Zoom sur une pause dans une station service

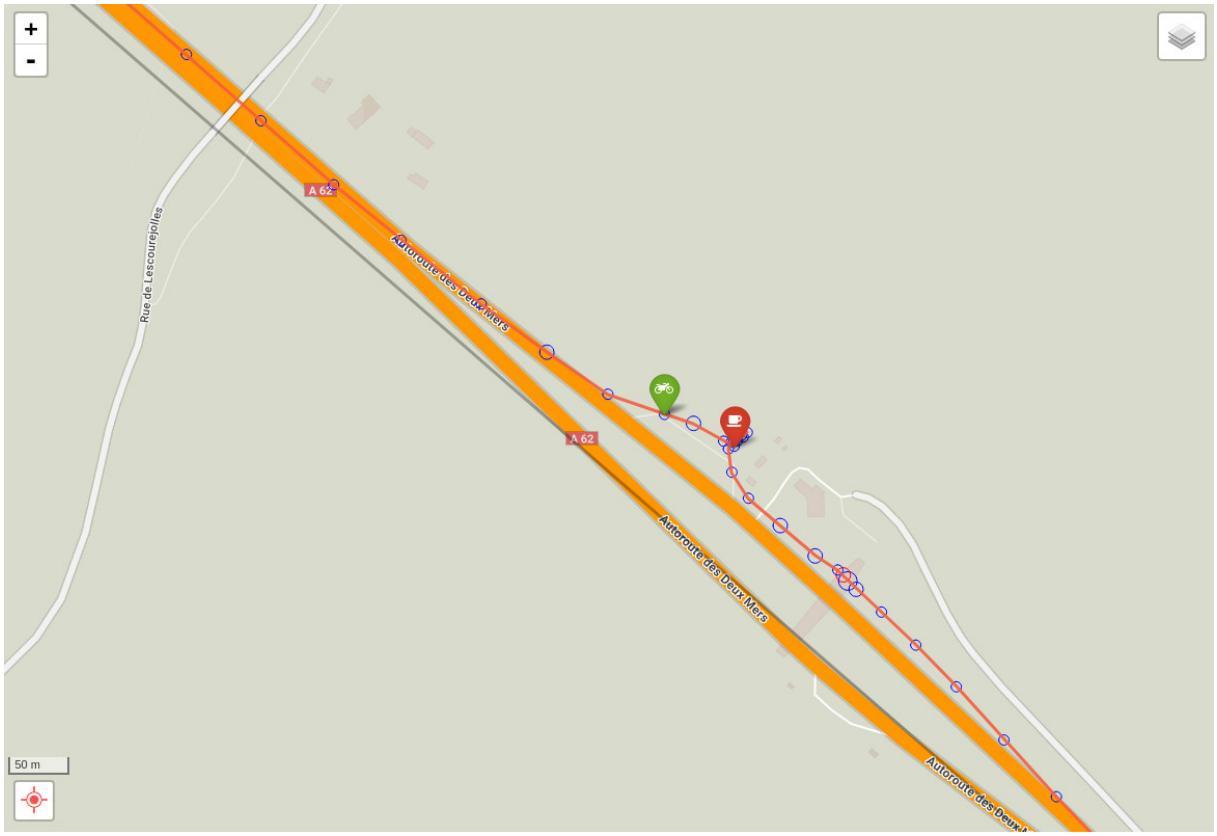

FIGURE 3.7 – Zoom sur une pause sur le bord de la route

Retrospective

Il s'est passé beaucoup de choses en 6 mois, et si au début j'ai regretté de ne pouvoir travailler durant toute la période du stage sur un même projet, du fait des besoins bien précis d'une petite entreprise en terme de développement, je me félicite d'avoir pu découvrir autant d'aspects différents d'une entreprise. Travailler dans une startup demande beaucoup d'implication, mais apporte une expérience très riche en retour : l'impression d'être un élément important, d'avoir une voix qui porte, la proximité aux autres secteurs et équipes de l'entreprise et la compréhension d'un problème dans son ensemble.

Mon stage se termine par une proposition d'embauche et je suis désormais un développeur à temps plein chez Liberty Rider, curieux et impatient de voir jusqu'où nous allons aller.

Table des figures