****

**Acclimate : rapport final**

**par**

**Jérémi Grenier-Berthiaume**

**Olivier Lepage-Applin**

**Sophie Savoie**

**Chapeauté par Mr Houari Sahraoui**

**Dans le cadre du cours**

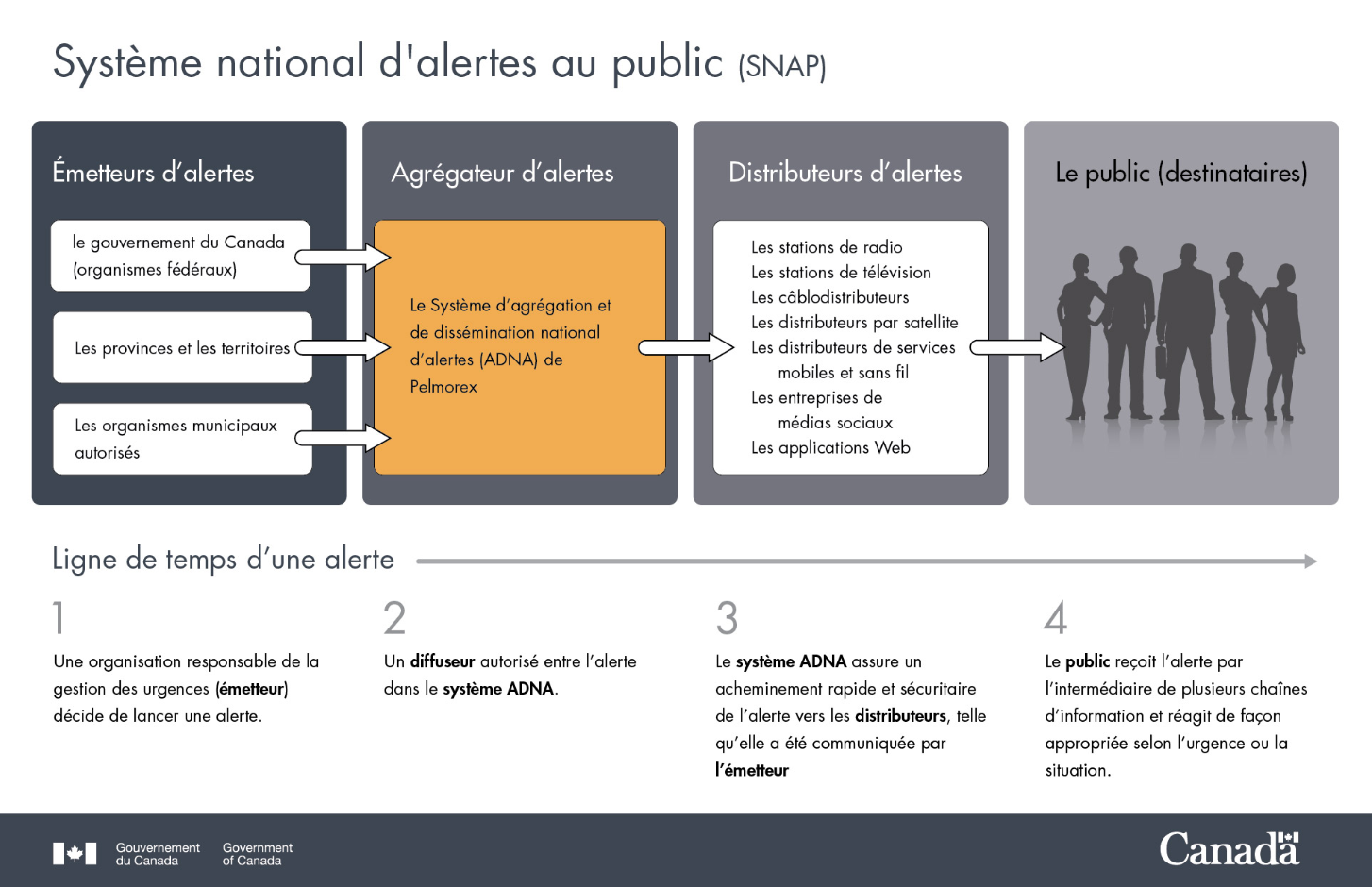
**Projet d’informatique – IFT 3150**

**Vendredi 14 septembre 2018**

**Détails sur le problème :**

L’idée du projet provient de l’observation de Sophie quant au manquement du gouvernement par rapport à une utilisation facilitée de [certaines données fournies au grand public](https://www.donneesquebec.ca/recherche/fr/dataset/carte-vigilance-multirisque-fil) (en fait, plus précisément, on remarque que le contenu de [l’onglet « Applications »](https://www.donneesquebec.ca/recherche/fr/dataset/carte-vigilance-multirisque-fil/related) indique qu’ « aucune application n’a encore été développée avec ce jeu de données »).

En soit, en analysant les alertes qui étaient publiées dans le flux RSS des données présentées ci-haut, on a pu remarquer que le gouvernement s’attendait à ce que les gens envoient un « gazouillis » (pour réutiliser leur mot) afin de les notifier de quelconque changement climatique en lien avec certaines des alertes publiées. Il était donc claire qu’**une application Android manquait cruellement à l’outillage** entourant l’initiative du SNAP ([Système National d’Alertes au Public](https://www.publicsafety.gc.ca/cnt/mrgnc-mngmnt/mrgnc-prprdnss/ntnl-pblc-lrtng-sstm-fr.aspx?wbdisable=true)). L’image présentée ci-dessous (provenant du précédent lien) est un autre témoin de l’oublie quant à l’importance de faciliter une communication dans les deux sens : il faut aussi penser à faciliter la signalisation d’alertes au gouvernement par les citoyens plutôt que d’avoir une approche qui n’est qu’unilatéral.



De ces observations, on peut retirer trois objectifs principaux pour l’application Acclimate :

1. Faciliter la visualisation des données d’alertes publiées par le gouvernement.
2. Permettre aux usagers de publier des alertes et mises-à-jour (retour d’informations).
3. Faciliter la notification par rapport aux alertes selon des intérêts divergents.

**Répartition des tâches :**

Dès la phase d’analyse, nous avons commencé le processus de répartition des tâches. Dans le cadre de ce cours universitaire, je ne vais préciser que la répartition qui concerne la période suivant le Hackathon (HackQc 2018 – [Gagnant du 3e prix](https://devpost.com/software/hackqc-remise)) puisque la majorité du travail y a été fait :

|  |  |
| --- | --- |
| **Noms** | **Assignations principales** |
| Jérémi Grenier-Berthiaume | Application Android, WebApp |
| Olivier Lepage-Applin | Application Android |
| Sophie Savoie | Base de données MySQL, REST API |

**Glossaire :**

Afin de clarifier certains concepts, j’ai cru pertinent d’offrir un glossaire à mon équipe pour ce qui touchait ma partie du développement.

* **Firebase**: Ensemble de services offerts par Google pour les développeurs.
* **Admin SDK**: Permet l'accès à certaines fonctionnalités d'un API de Firebase (modifier le mot-de-passe d'un usager de la base de données d'authentification, par exemple). Authentifie l'application faisant les requêtes. L'intégration de l'Admin SDK doit se faire « en privé » (elle nécessite d’insérer dans le code de l'information sensible que le public ne doit pas connaître).
* **FCM**: Firebase Cloud Messaging. Service facilitant l'envoie de Notifications à des appareils mobiles.
* **Token**: String devant être utilisée afin de mener à terme un certain objectif.
* **registrationToken**: Chaque appareil mobile unique ayant téléchargé l'application reçoit un 'registrationToken' unique. C'est ce String que l'on doit utiliser pour envoyer des notifications avec FCM. (Sous certaines conditions très particulières, celui-ci peut cependant changer.)
* **OAuth 2**: Protocole qui permet à un usager de s'authentifier sans nécessairement envoyer ses données sensibles (tel que son mot-de-passe) au serveur auquel il fait une requête.
* **idToken**: C'est un token utilisé par le protocole OAuth 2 de Firebase pour permettre d’authentifier la requête d'un usager (la requête est envoyée avec ce token).
* **JWT**: JSON Web Token. C'est un JSON encodé en base 64. Il possède la forme "XXX.YYY.ZZZ", où XXX est le '*header*', YYY le '*payload*' et ZZZ la '*signature*'. Dans le cas de l'authentification via Firebase (où l'idToken est un JWT), le 'header' contient une clé publique de Google ainsi que l'algo utilisé pour la signature (soit RSA 256 bits, ou "RS256"), le 'payload' contient les informations en lien avec l'usager qui fait la requête (email, UID, etc.), et la 'signature' permet d'authentifier la requête.
* **UID**: Chaîne de caractères unique assignée de manière permanente à tout utilisateur s'authentifiant via Firebase à une application.

**Description des accomplissements :**

[**Site web du projet**](https://www.webdepot.umontreal.ca/Usagers/p1071435/MonDepotPublic/Acclimate/index.xhtml) **(pas la WebApp) :**

J’ai fourni le patron pour les pages personnelles qui concernent le « rapport de progrès bihebdomadaire. »

Le code HTML et CSS du site provient de notre (Olivier et moi) TP1 remis dans le cadre du cours d’*Introduction au Design Web*. Le site possède un design dynamique, c’est-à-dire qu’il s’adapte à l’affichage pour quelconque support d’accès (ordinateur, tablette, téléphone mobile, etc.).

**Serveur (REST API):**

Quatre contributions directes :

* Intégration du système d’authentification du serveur auprès des serveurs de Google afin de permettre l’utilisation des fonctionnalités de l’Admin SDK de Firebase.
* Implémentation des patrons des méthodes de base en lien avec les notifications et la vérification des idTokens (ceux-ci permettent de vérifier l’authenticité et l’intégrité des requêtes envoyées par nos usagers) en s’assurant que tout soit prêt pour une intégration fluide et rapide de la part de Sophie.
* *Parsing* des alertes de la base de données historiques du gouvernement afin de les transférer dans notre propre base de données (ce qui nous permet alors de bonifier celle-ci avec les informations que nous récoltons via l’application).
* Déblocage d’un problème de *Thread* qui rendait la requête de « POST Zone Surveillée » lente et dépendante de la grosseur de l’aire couverte (on est alors passé d’un temps d’attente parfois supérieur à 30 secondes à un temps relativement fixe de 1 seconde).

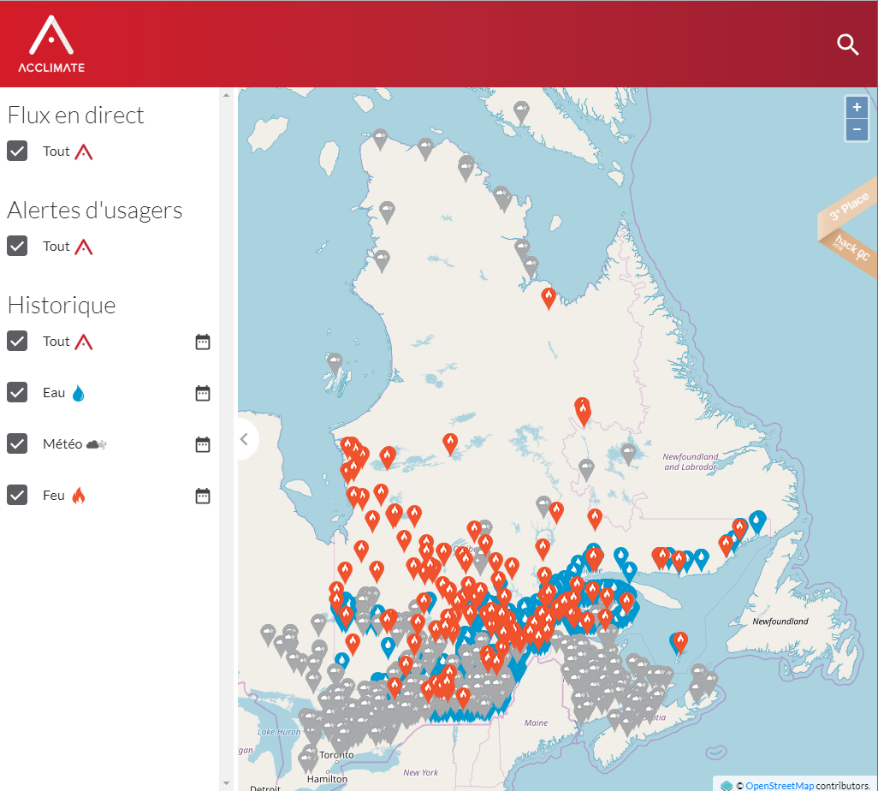
Deux propositions de **structures de données** pour améliorer l’efficacité du traitement des requêtes :

* *HashMap* des alertes pour effectuer une mise-à-jour en temps réel plus efficace des alertes et faciliter le traitement des notifications à envoyer. L’idée est de conserver en mémoire la version la plus à jour de la base de données (BDD) afin d’éviter d’avoir à faire une requête pour obtenir l’état actuel de la BDD. Ainsi, il est possible de n’envoyer que quelques requêtes de PATCH à la BDD plutôt que de tout recréer. Il faut savoir qu’une attention doit être portée aux alertes qui ne figurent plus dans le flux RSS du gouvernement car nous ajoutons alors celles-ci dans notre BDD que l’on dit « Historique » (ces données récoltées pourraient possiblement servir pour des projets d’apprentissage machine, par exemple).
* Système d’indexage des zones surveillées qui permet de trouver plus rapidement à quel usager on doit envoyer une notification pour une nouvelle alerte donnée. On utilise une fonction mathématique pour savoir dans quelle région géographique (ou « tuile ») tombe l’alerte, et chaque tuile contient un lien vers toutes les zones surveillées qu’elle contient (il n’y a alors pas autant de calculs et comparaisons à faire avec les coordonnées de l’alerte puisqu’on n’a plus de besoin de passer à travers l’entièreté des zones surveillées).

**WebApp :**

Comme expliqué dans le site web du projet, nous étions 5 à travailler sur le projet pendant la période précédant la fin du Hackathon. Par la suite, j’ai été assigné au développement de la *WebApp* car c’était *Charles-Philippe Lepage* qui s’en occupait avant.

Il y a donc eu un effort pour comprendre et modifier le code qui était déjà là afin de mener à terme les points mentionnés ci-dessous (entre autres, il a fallu lire de la documentation sur **AugularJS** et **OpenLayers**) :

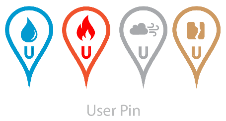
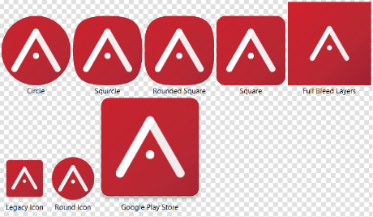
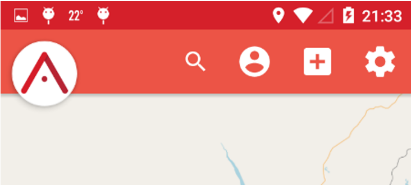
* Ajustement du positionnement des alertes lors de la décision de changer d’une référence de coordonnées (Lat, Lng) à (Lng, Lat).
* Réglage des bugs sur certaines classifications d’alertes et d’icônes.
* Implémentation des images différentes pour les pins des alertes d’usagers.
* Ajouts de catégories de classification d’alertes supplémentaires.
* Léger *refactoring* et ajout de commentaires pour augmenter la maintenabilité.
* Réglage des filtres d’affichage des alertes sur la carte.
* Changement des URLs de requêtes d’API lorsqu’on a migré le serveur.
* Ajout des descriptions des alertes d’usagers dans les infobulles.
* Implémentation de l’ajustement automatique du *zoom level* lors de la recherche d’une localisation.
* Déploiement sur une nouvelle URL et activation de la mise-à-jour automatique du site web lors d’un *commit*.

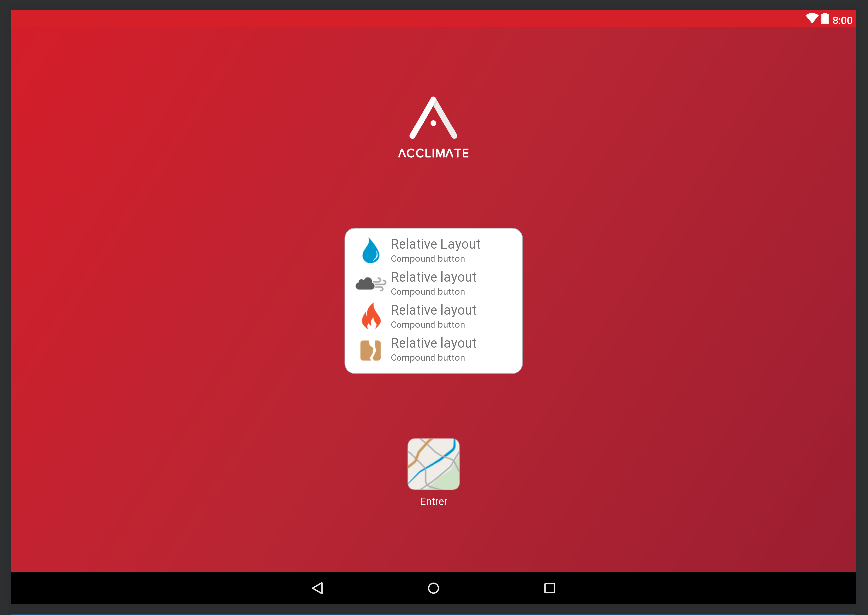
La WebApp remplie sa fonction principale : la visualisation des alertes « historiques » à travers tout le Québec. De plus, tout comme l’application Android, elle permet de voir les alertes de tous les types.

Pour l’instant, considérant que nous ne voulons pas permettre de publier des alertes usagers via cette WebApp, nous avons mis de côté l’implémentation des comptes usagers.

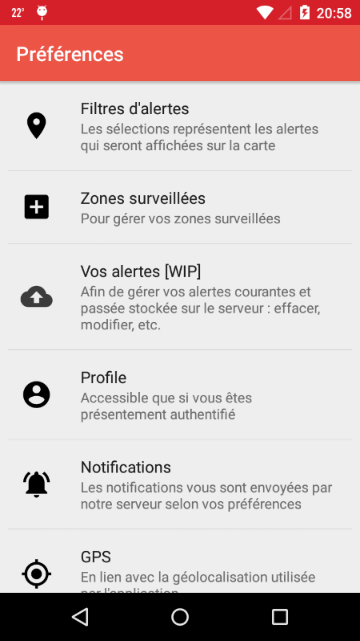
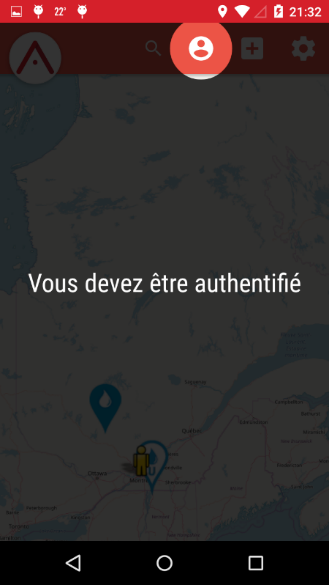
**Design / UI :**

L’attrait visuel d’un logiciel est généralement important pour son succès. Charles-Philippe a été le principal designer et a fourni la majorité du matériel de base durant le Hackathon, mais j’ai aussi contribué quelque peu (en plus d’être presqu’exclusivement celui qui a intégré les dessins de manière fonctionnelle dans l’application Android). Parmi mes différentes réalisations, on dénote :

* La création des « Pins usager » qui permettent de différencier plus rapidement si une alerte provient du gouvernement ou d’un usager.
* La génération des différentes versions du Logo qui sont utilisées en fonction de différents contextes (la version d’Android du téléphone qui possède l’application, la recherche de l’application dans Google Play Store, etc.).
* La ***Toolbar*** (la barre horizontale située tout en haut de l’écran et qui contient généralement des boutons afin que l’usager puisse interagir avec l’application Android).
* Les différentes « pages » de l’application Android (*Activities* dans le jargon d’Android) qui s’adaptent automatiquement à différentes grosseurs d’écran. Par exemple, voici la même Activité avec une résolution de **768 x 1280** et puis de **2560 x 1800** pixels.

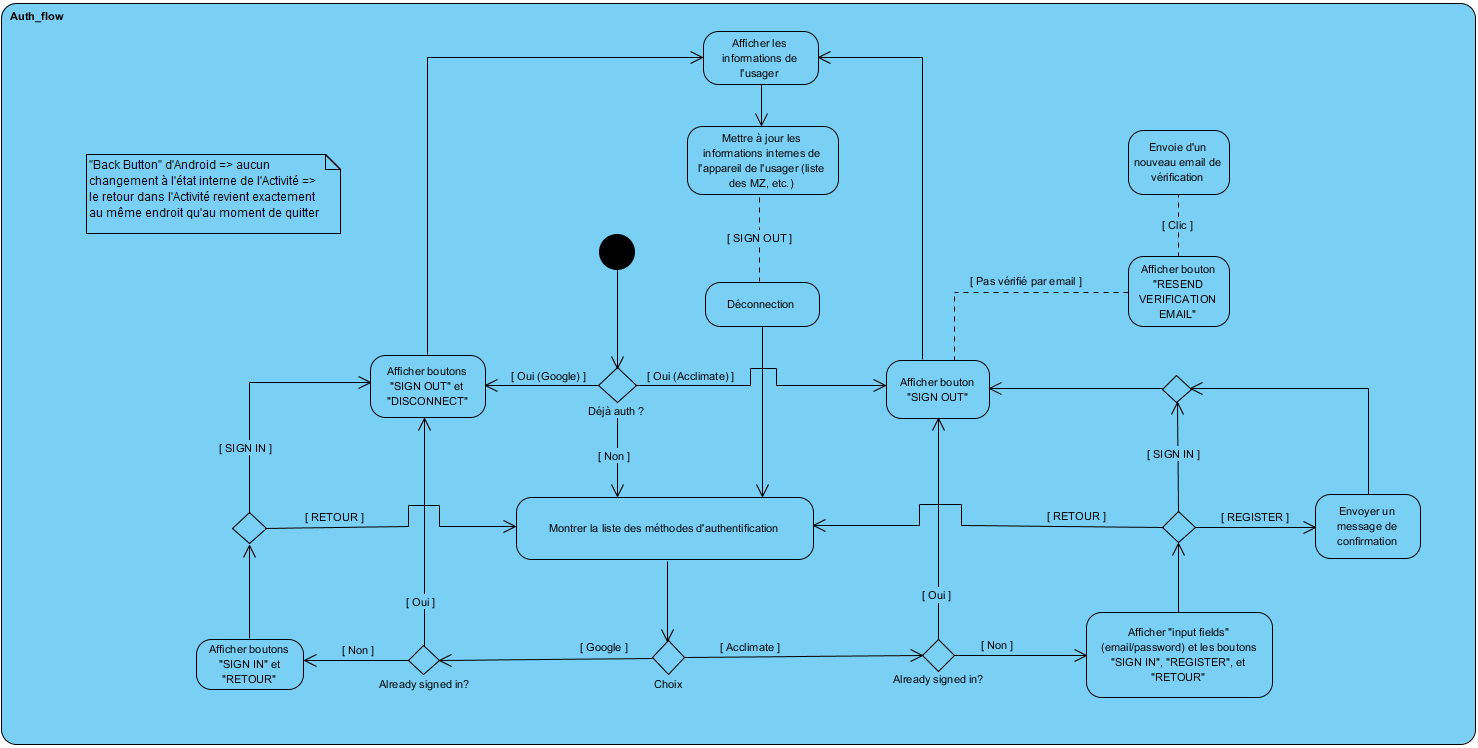
* **UI** (ici les pages d’Authentification et des Préférences, ainsi qu’un aperçu d’une animation servant à guider l’utilisateur):

**Application Android :**

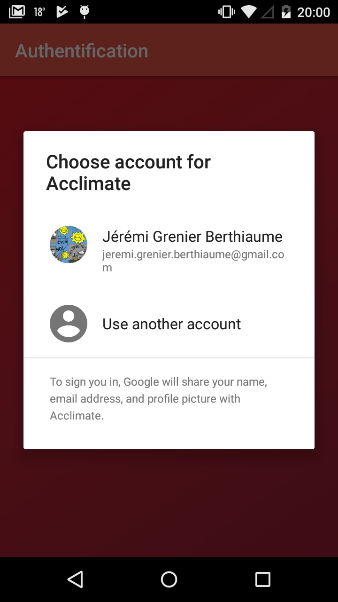
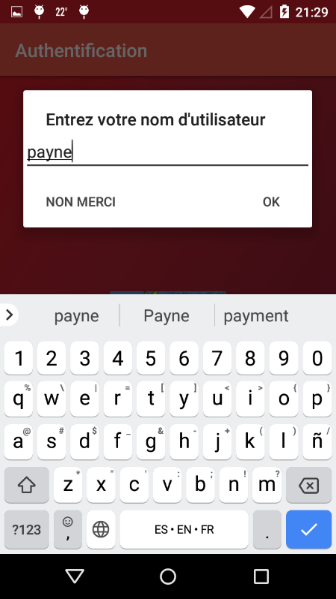
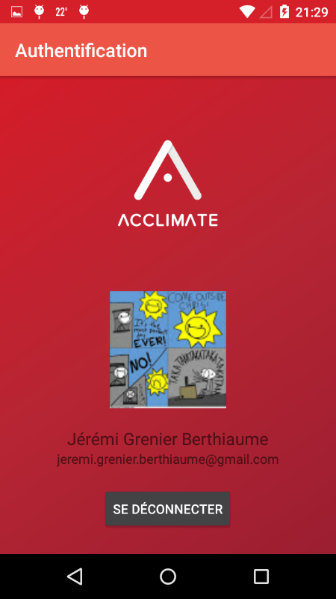
Facilement 95% de mon temps a été dédié au développement de l’application Android. Mes contributions sont multiples et [mon rapport hebdomadaire](https://www.webdepot.umontreal.ca/Usagers/p1071435/MonDepotPublic/ProjInfoPerso/) pour le cours dresse une liste détaillée de celles-ci. Ce rapport final peut cependant présenter une synthèse à l’aide d’une séparation par composantes. Ainsi, j’ai été le développeur principal de l’intégration de :

* **Authentification**

Via le protocole OAuth 2, il n’est plus nécessaire pour un usager d’envoyer ses données sensibles afin d’authentifier ses requêtes au serveur: un système de signatures électroniques et d’infrastructure à clé publique permettent de plutôt utiliser une chaîne de caractères. Ainsi, cela nous a permis de ne pas avoir à conserver ces informations (qui peuvent avoir des répercussions légales) puisque notre code ne nécessite pas de les injecter dans les requêtes.*Auth-flow (UPDATE : regToken and merge flows and no DISCONNECT/RETURN)*

Il a aussi fallu éduquer le reste de l’équipe sur le processus pour les requêtes (voir **Annexe**).

Voici quelques aperçus de différents flots d’authentification :

|  |  |
| --- | --- |
| * **Synchronisation** | * **Notifications** |
| Qui dit compte usager dit synchronisation : lorsqu’il lance l’application, il faut vérifier s’il est authentifié et agir en fonction. De plus, il a fallu planifier le comportement de l’application lorsque l’usager n’a pas de connexion internet.  Pour laisser savoir à l’usager qu’il est en attente d’une réponse, j’ai opté pour un système de couleur : le ***bleu*** signifie qu’il est en attente d’une réponse du serveur et le ***rouge*** signifie que les informations sont synchronisées. | Avec la librairie utilisée, chaque appareil (et non pas chaque usager) possède un identifiant unique (le « registrationToken ») qui permet au serveur de le rejoindre. Ainsi, il a fallu adapter le « auth-flow » pour qu’à chaque connexion et déconnexion d’un usager la liste d’appareils correspondant à cet usager soit mise-à-jour du côté serveur. |
|  |  |

|  |  |  |
| --- | --- | --- |
| * **Préférences** | * **Recherche d’adresses/endroits/etc.** | * **Localisation GPS** |
| Afin de permettre à l’usager de personnaliser son expérience, les préférences ont été intégrées : filtres sur les notifications, l’affichage de la carte, la connexion réseau, etc. | OSMDroid ne possède pas une bonne base de données d’adresses alors j’ai dû implémenter une requête passant par le serveur de Google. | Fonctionnalité de base pour une application affichant une carte. Sur Android, la gestion des permissions rends néanmoins son intégration pas aussi triviale qu’on pourrait le penser. |
|  |  |  |

* **Zones surveillées**

Sont des régions qui couvrent une partie de la carte. Seuls les utilisateurs authentifiés peuvent accéder à cette fonctionnalité. Une telle zone correspond à un abonnement pour recevoir une notification lorsqu’une alerte apparaît à l’intérieur de celle-ci. Voir l’**Annexe** pour une démonstration du cycle de vie d’une zone surveillée.

J’ai aussi intégré l’outil **Crashlytics** de Firebase qui permet de visualiser tous les crashs qui sont arrivés durant l’utilisation de l’application. Malheureusement, je n’ai connu l’existence d’un tel outil un peu trop tard (ce n’est que pour les quelques dernières semaines de développement que nous avions accès à l’outil), mais ce sera désormais un des premiers outils que je vais ajouter à un projet car j’en vois très bien la pertinence. Firebase Performance?

L’outil fait partie de la « **Firebase Console** » qui regroupe plusieurs outils et services et offre entre autres la visualisation de plusieurs graphiques. Heureusement, j’ai inscrit le projet à la Console assez tôt dans son développement et donc il y a bien plus de données générales à visualiser que de données en lien avec les crashs (voir **Annexe**).

Finalement, j’ai intégré une fonctionnalité de « *clusters* » pour les alertes provenant de notre BDD « historique » afin de permettre une expérience moins intense sur le processeur du mobile (car on parle ici de plusieurs milliers d’alertes) : dépendamment du niveau de zoom il y a donc des regroupements d’alertes d’affichés (voir **Annexe**).

**Résultats obtenus :**

Rappelons les trois objectifs principaux fixés pour l’application :

1. Faciliter la visualisation des données d’alertes publiées par le gouvernement.
2. Permettre aux usagers de publier des alertes et mises-à-jour (retour d’informations).
3. Faciliter la notification par rapport aux alertes selon des intérêts divergents.

Le premier point est résolu par la projection des alertes sur une carte à l’aide des coordonnées extraites des données. L’application Android autant que la WebApp répondent à ce besoin.

Le second point a été atteint de manière satisfaisante : les usagers peuvent bel et bien publier des alertes et un système de Karma a été développé pour permettre de filtrer celles-ci (des notifications ne sont donc pas nécessairement envoyées tant que le seuil minimum de confirmations n’a pas été atteint). Il manque encore une approche plus directe de retour d’informations (ce point est discuté dans la section « Améliorations futures possibles » du rapport). Néanmoins, temporairement, la publication d’une alerte est analogue à une mise-à-jour par rapport à une situation.

Le troisième point a lui aussi été atteint de manière satisfaisante : grâce aux zones surveillées, un usager peut s’abonner à une région ce qui laisse savoir à notre serveur que lorsqu’une nouvelle alerte apparaît dans cette zone il faut envoyer une notification à tous les appareils enregistrés par cet usager. Le système pourrait aussi inclure un abonnement à une alerte en soit (pour le jour où nous intégrerons le système de commentaires sur les alertes) et j’ai déjà partiellement implémenté dans le code les méthodes en lien avec une telle fonctionnalité.

J’estime qu’avec moins de 120 heures supplémentaires réparties entre les 3 équipiers du projet, les points 2 et 3 pourraient être menés à terme de manière assez complète.

**Langages de programmation, outils et librairies utilisés :**

J’ai travaillé avec deux langages de programmation : **Java 8** et **JavaScript** en plus de faire appel à la **documentation** de [Java 8](https://docs.oracle.com/javase/8/docs/api/), [Android](https://developer.android.com/guide/), [Firebase](https://firebase.google.com/docs/android/setup) et [OpenLayers](http://openlayers.org/en/latest/apidoc/). Le **JSON** et le **XML** ont aussi souvent été utilisés. Les outils et librairies suivants ont été employés :

|  |  |
| --- | --- |
| **Outils** | **Librairies** |
| **IntelliJ** 2018.2 | **AngularJS** v1.6.9 \* |
| **Android Studio** v3.1.4 | **OpenLayers** v4.6.5 (carte de la WebApp) |
| **Gradle** v4.4 | **JackSON** (JSON) |
| **Visual Studio Code** v1.26.1 | **OSMDroid** (carte dans l’application Android) |
| **GitHub Desktop** v1.3.4 | **Firebase Authentication** |
| **GitHub** | **Firebase Cloud Messaging** (notifications) |
| **BitBucket** | **Fused Location** (localisation GPS) |
| **Crashlytics (Firebase)** v2.9.5 | **Lombok** (annotations) |
| **Firebase Console** | **Picasso** (images dans Android) |
| **Herokuapp** (avec **UptimeRobot**) | **FancyShowCaseView** (animations dans Android) |
| **Adobe Illustrator CS6** et  **Adobe Photoshop CS3** |  |
| **Node.js** v8.11.4 |  |
| **UML Software** |  |

\* est en fait un *framework*

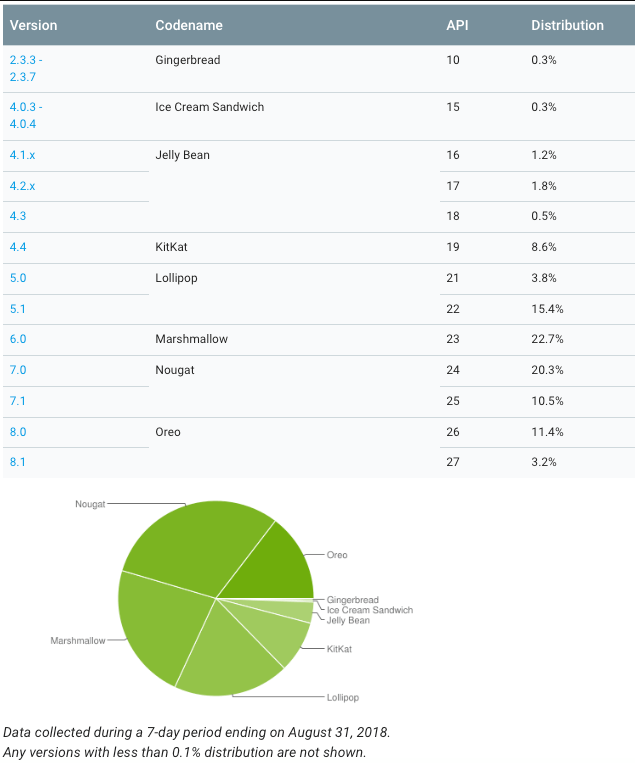
Je me dois aussi de faire une mention spéciale pour ma découverte de [**Stack Overflow**](https://stackoverflow.com/users/9768291/payne). 😊

**Défis :**

Pour nous trois, cela a été notre première expérience de développement faisant appel à une base de données, une application mobile et un serveur (ce qui adonne à être une architecture 3-tiers). En fait, nous entamions à peine le troisième trimestre de ce baccalauréat au moment de nous lancer dans ce projet : il va donc sans dire que ce fut une expérience des plus enrichissantes et formatrices.

Considérant que je n’ai pas touché au code concernant la base de données directement, la partie de ce que j’ai fait au niveau du serveur était plutôt triviale (de simples méthodes Java et lecture de la documentation traitant des librairies que je tentais d’intégrer).

L’utilisation de OAuth 2 a simplifié certains aspects de l’authentification, mais le temps requis pour s’assurer que l’implémentation est sans faille de sécurité est trop grand considérant le cadre de ce cours (voir [ce PDF](https://tools.ietf.org/pdf/rfc6819.pdf) de 71 pages sur le sujet).

Cependant, l’apprentissage du développement Android fut une longue tâche (qui n’est techniquement toujours pas terminée vu l’immensité de ce *framework*). N’ayant préalablement aucune expérience dans ce domaine, les décisions par rapport aux choix des librairies à utiliser pour l’intégration de certaines fonctionnalités ont nécessité beaucoup de recherches. De plus, la décision par rapport au « minSDK » (c’est-à-dire la version minimale d’Android nécessaire pour utiliser l’application) fut difficile à évaluer sans savoir d’avance quelles librairies nous allions utiliser. L’approche générale était de tenter de couvrir la plus grande part de marché sans néanmoins perdre l’accès à des fonctionnalités utiles offertes par des versions plus récentes. J’ai fait le choix de viser le niveau d’**API 16** car nous couvrons alors plus de **99.44%** des appareils ([source](http://www.gadgetdetail.com/android-version-market-share-distribution/) de l’image à droite). Pour des projets futurs, je pense qu’il serait plus raisonnable de viser l’**API 19** puisqu’il y avait un nombre décent d’options qui semblaient être offertes à partir du niveau 18 alors que je lisais la documentation d’Android pour résoudre certains problèmes (la perte sur la part de marché est minime et le sera de plus en plus au fil du temps).

Un autre grand défi fut de travailler pour la première fois dans un contexte réel d’intégration de parallélisme de tâches. En effet, ne pouvant prendre pour acquis que les données obtenues du serveur arriveront avant ou après la création de la carte (pour ne donner qu’un exemple), il a fallu intégrer des logiques d’affichage plus complexes que lors d’une approche (triviale) synchrone.

Aussi, le simple fait que ceci était un travail d’équipe et qu’il fallait tenter de travailler de manière parallèle le plus possible en réduisant le plus possible les temps d’attente entre nous a requis de bonnes capacités de communication. J’ai bien compris l’importance des canaux de communication et la pertinence du rapport « démocratique » de développement. De plus, j’ai pu remarquer que certains changements d’architecture peuvent parfois faire en sorte que certaines parties du code qui avaient été développées doivent alors être réécrites complètement.

Finalement, un de mes objectifs était de permettre l’utilisation de l’application par plusieurs usagers sur le même appareil, de même que l’utilisation par un même usager sur plusieurs appareils. Ceci a complexifié quelque peu le processus de synchronisation :

[ DIAGRAMME CONSÉQUENT ]

**Propositions d’améliorations futures :**

Parfois, il semblerait que la liste pourrait s’étendre à l’infini, mais voici quelques points :

* Permettre aux gens de s’abonner à une publication effectuée sans l’utilisation d’internet : textos.
* Sauvegarder certaines préférences en lien avec le Profile d’un usager au niveau de la BDD afin de permettre leur synchronisation lorsqu’un même usager utilise de multiples appareils.
* Utiliser un « [*Navigation Drawer*](https://developer.android.com/training/implementing-navigation/nav-drawer)» pour libérer l’espace dans la « *Toolbar* » dans le haut de la carte (ne laisser que « Rechercher »).
* Intégrer un « *Splash Screen* » pour mieux contrôler le flot d’initialisation de l’état de l’application (synchronisation des informations avec la BDD, principalement) ainsi que pour rendre le code plus aéré (maintenance facilitée par des logiques moins complexes).
* Stocker les dates en tant que « *long* » plutôt que « *String* » dans la BDD (un ‘long’ prends moins d’espace qu’une String et est une approche facilitant les comparaisons). Ce changement rendrait plus facile l’ajout de la fonctionnalité de filtre-par-date de la WebApp.
* Intégrer une vérification de la version de l’application afin de pouvoir gérer correctement les synchronisations (si une nouvelle version change quelque chose qui pourrait causer un crash dans une ancienne version, par exemple).
* Intégrer une Queue pour conserver une « référence forte » des requêtes à l’API pour s’assurer que le Garbage Collector ne détruise jamais celles-ci (nous n’avons toujours pas observé ce comportement, mais c’est néanmoins une idée que j’ai proposée comme amélioration future).
* Permettre une sélection par clavier numérique du nombre de mètres pour le rayon d’une nouvelle zone surveillée (présentement il n’y a que le « glisseur »).
* Intégrer les « Notifications GPS » qui utilise la localisation de l’usager pour faire des requêtes au serveur pour savoir s’il y a des alertes à l’intérieur d’un certain rayon (choisi par l’utilisateur). La fréquence des requêtes peut dépendre du temps ou encore de la distance parcourue (ou les deux).
* Ajouter des actions spécifiques à effectuer lorsqu’un usager clique sur une notification d’un certain type (centrer sur la zone surveillée, par exemple).
* Une option de déconnexion automatique (pour l’authentification). Présentement, par défaut, l’utilisateur reste connecté jusqu’à ce qu’il indique le contraire.
* Une suppression automatique des usagers qui n’ont pas effectuer la vérification par e-mail de leur nouveau compte à l’intérieur d’un certain nombre de jours. Il pourrait aussi y avoir des restrictions sur les actions permises aux usagers qui n’ont pas encore été vérifiés.
* Un peu comme certaines autres applications (pensez à *Messenger* et *Gmail* et au petit chiffre rouge), indiquer le nombre d’alertes (en général ou encore dans les zones surveillées) sur l’icône de l’application dans le « bureau » de l’Android d’un usager serait un bel ajout.
* Permettre aux usagers de commenter les alertes et permettre au publicateur de l’alerte de s’abonner à son alerte afin de recevoir des notifications lorsque des commentaires sont émis (cela permettrait entre autres de répondre au besoin du gouvernement de recevoir des mises-à-jour directes par rapport à leurs publications). Il pourrait y avoir un filtre qui ne permet d’alerter le publicateur que si la personne qui commente est à une certaine distance minimale de l’alerte (ce qui augmente substantiellement la crédibilité du commentaire).

Il y a plusieurs outils Firebase que j’aurais voulu intégrer, mais je n’ai pas nécessairement eu le temps. « TestLab », « Performance », etc.

**La suite du projet :**

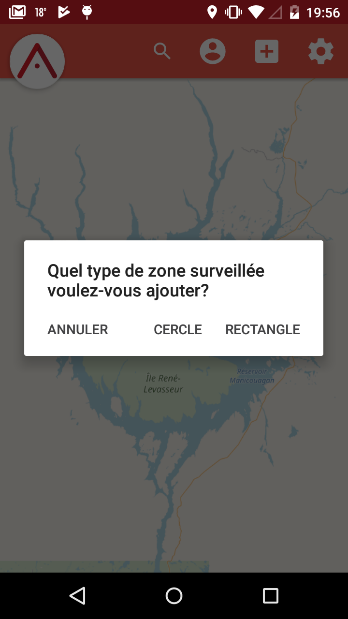
Je pense que l’application a atteint un stade qui dépasse le « *proof of concept* » et qui pourrait donc intéresser certains partis. Son utilité peut s’étendre à plusieurs domaines et nous gardions en tête ce fait durant le développement afin de nous assurer qu’il ne serait pas trop difficile d’adapter l’application à des besoins changeants ou encore des cas d’utilisation complètement différents.

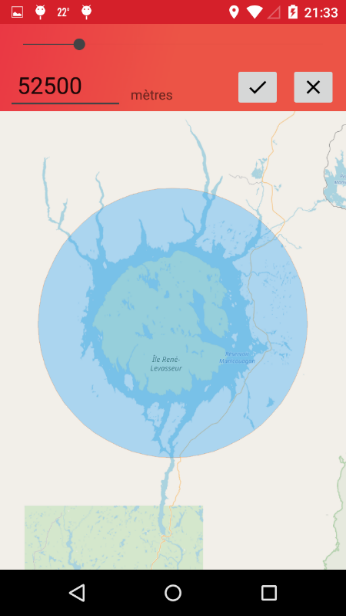
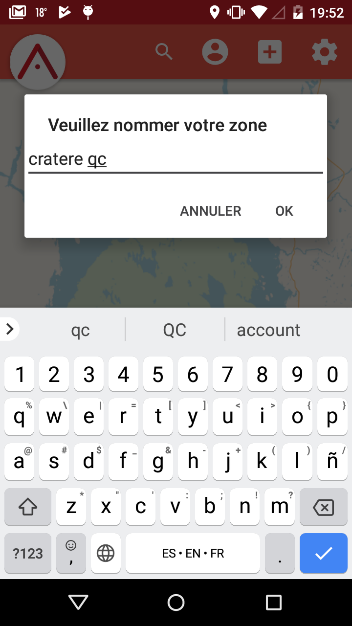
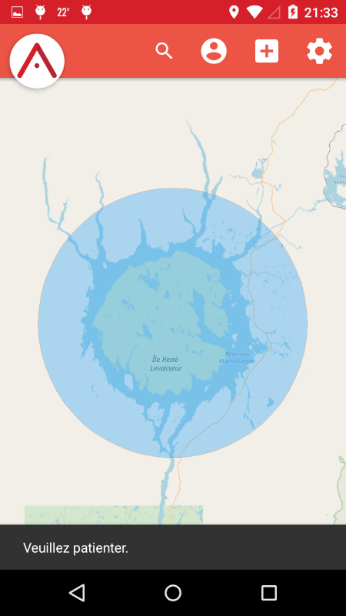
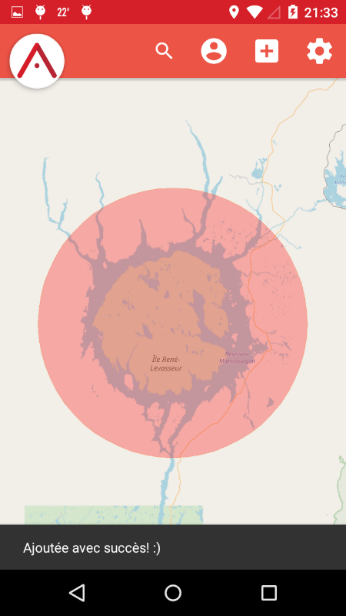
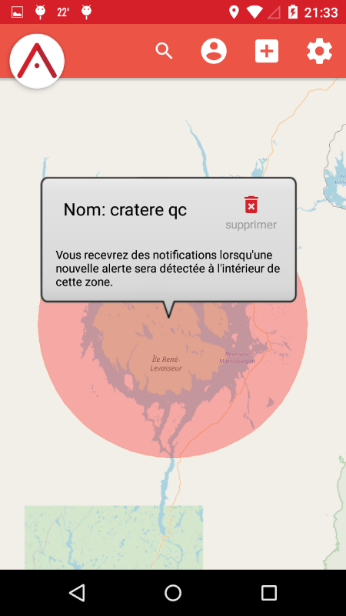
De plus, comme mentionné dans l’introduction, la motivation derrière le projet était de répondre à une lacune du gouvernement. Nous allons donc discuter de notre stratégie d’approche afin de présenter le projet pour qu’il soit possiblement intégré par le gouvernement.

Finalement, j’ai l’intention de proposer à l’équipe de soumettre notre candidature pour le [Prix pour service exemplaire en sécurité civile](https://www.publicsafety.gc.ca/cnt/mrgnc-mngmnt/xmplry-wrds/ctgrs-fr.aspx#a01) considérant qu’une des catégories concerne les « Communautés résilientes. » Indépendamment de l’obtention de ce prix, il me semble que les différents exemples de volets que pourrait couvrir un candidat potentiel sont de très bons points de départ pour tenter d’obtenir une meilleure vision à long terme du genre d’améliorations que l’on pourrait apporter à l’application.

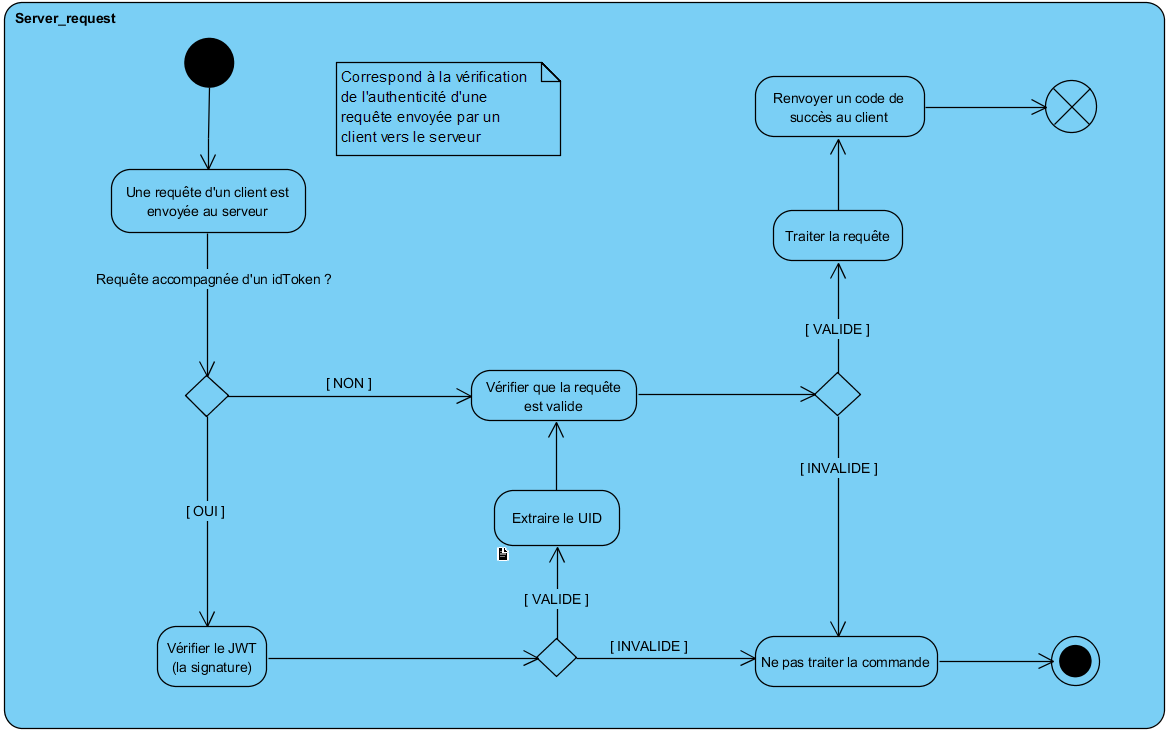
**Annexe**

**Cycle de vie d’une zone surveillée (circulaire)**

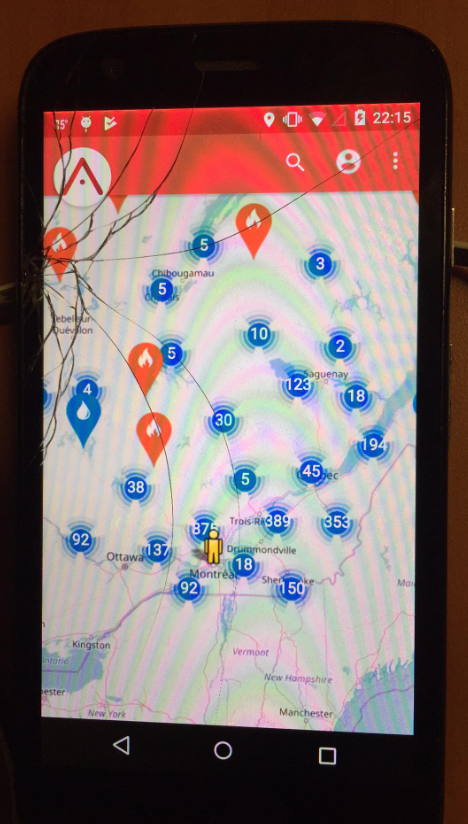
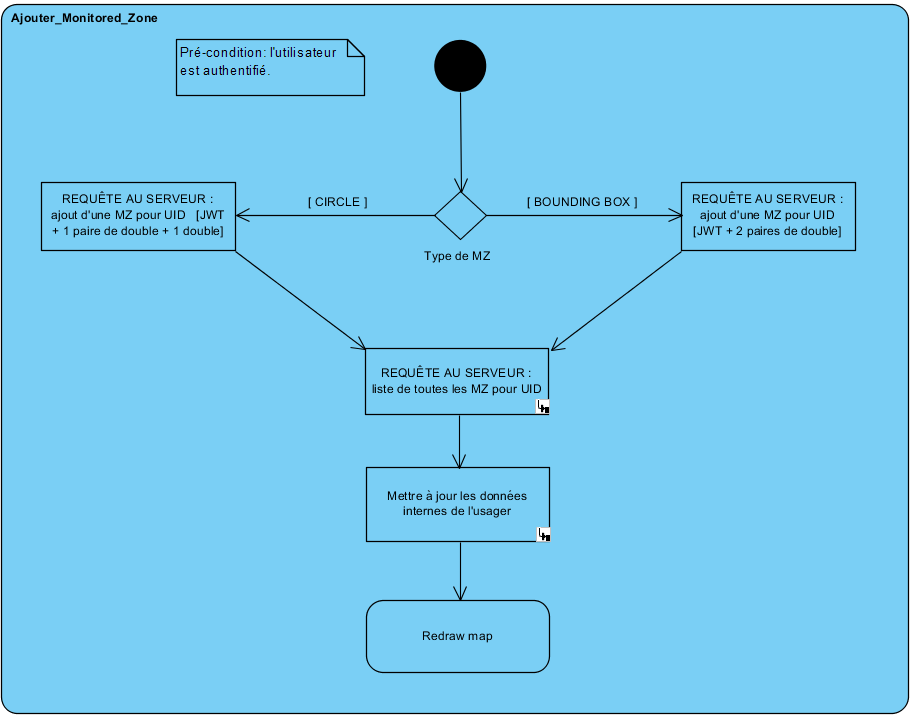


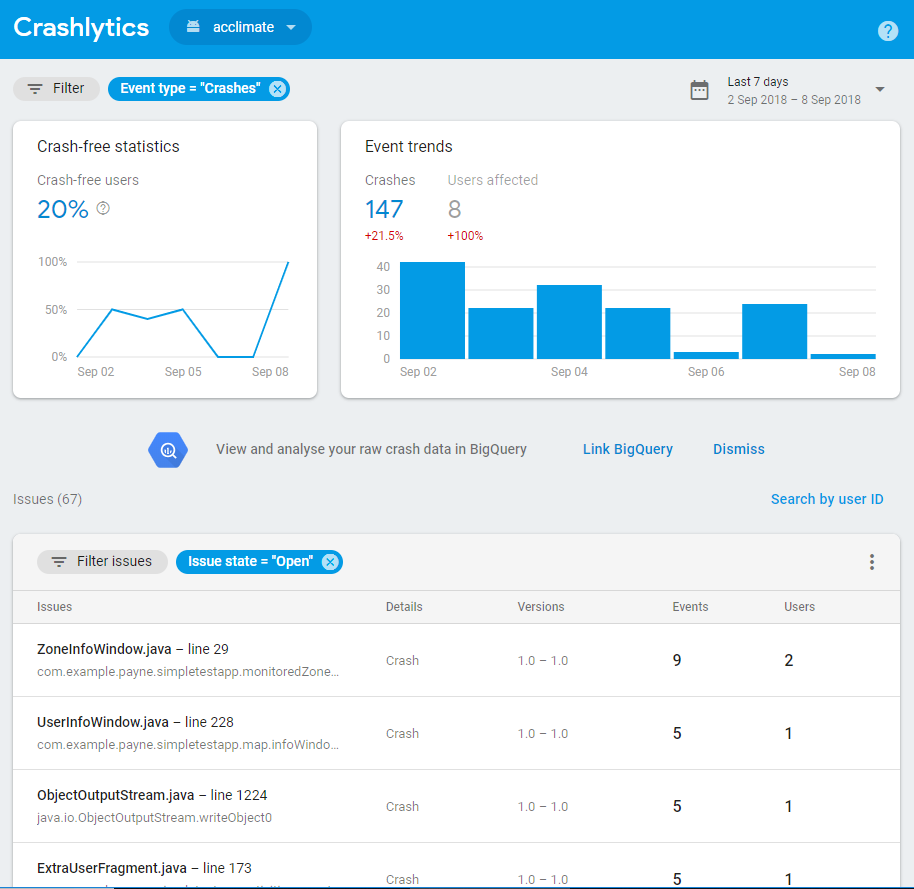
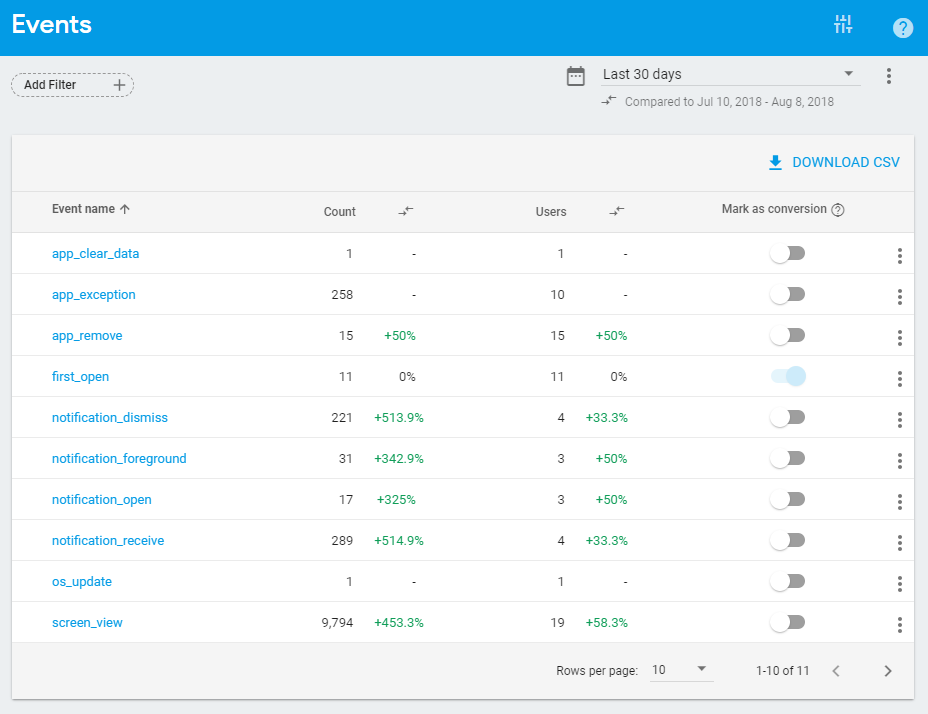
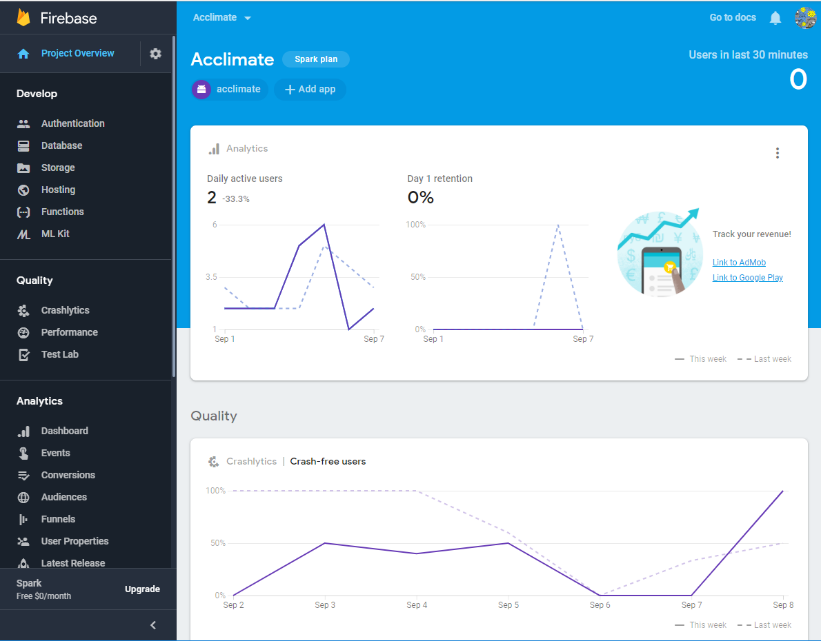
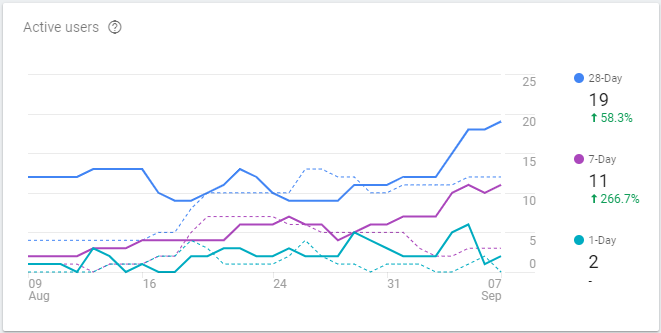
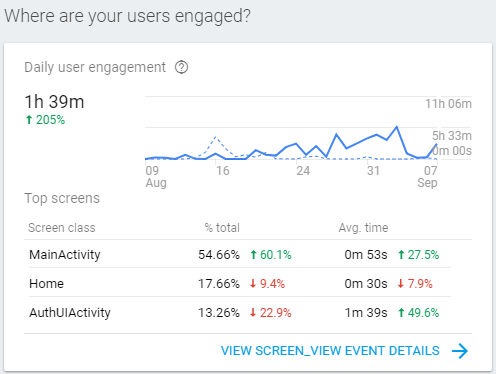
       

**Traitement d’une requête par le serveur**

****  
*Traitement d’une requête par le serveur*

**Ajouter une zone surveillée**

****  
*????????????????????????????????????????????????????*

**Firebase Console**

