Le numérique au service des pandémies : Comment instaurer un suivi des cas contact tout en préservant la confidentialité.

Pietrzak Aurélien

Lien du GitHub : https://github.com/MrADOY/covivre

Contexte **2**

Objectif **2**

Aperçu **2**

Pour résumer **3**

Fonctionnalités **3**

Comment échanger des données entre utilisateurs afin de détecter une rencontre ? 3

Verrou(s) levé(s) 3

Solution proposée 3

Comment contacter les cas contacts ? 6

Verrou(s) levé(s) 6

Solution proposée 6

Développement du POC **10**

Conclusion **12**

Contexte

Dans le cadre de l'urgence sanitaire et des recommandations d'expert dans le domaine médical, une des solutions pour éviter une vague épidémique et de procéder au tracking des personnes en contact avec des personnes présentant une infection au covid-19. Ces personnes appelées **"cas contacts"** sont ensuite amenés à rester en "quatorzaine" et d'effectuer un test afin d'éviter la propagation du virus.

Objectif

L'objectif ici est de pouvoir retracer l'ensemble des personnes qui ont été en contact avec des personnes malades en dehors du cercle familial, le problème comment contacter les personnes que nous pourrions avoir croisé dans un endroit confiné et dans un temps suffisant pour infecter une autre personne ? Les exemples sont assez vastes : Magasins, transports en commun, médecin, pharmacie.
En plus des éventuelles barrières technologiques, une barrière éthique se pose : comment traquer des personnes tout en respectant leur vie privée et en respectant les différentes lois établies concernant la vie privée des personnes comme par exemple le règlement général sur la protection des données (RGPD) et les recommandations de la commission nationale de l'informatique et des libertés (CNIL).
Notre objectif est ici de lever les barrières technologiques en trouvant des solutions qui existent déjà afin de mettre en place l'application rapidement mais aussi de débattre de différentes solutions pour d'éventuelles futures épidémies, mais aussi de trouver un moyen d'anonymiser les données afin de respecter la vie privée des utilisateurs.

Aperçu

Lorsque deux appareils se rencontrent, ils échangent des messages non personnellement identifiables qui contiennent des identifiants temporaires. Les identifiants tournent fréquemment pour empêcher des tiers de suivi des utilisateurs. L'historique des rencontres de l'utilisateur est stocké localement sur l'appareil ; aucune de ces données ne peut être accès direct, ces données sont accessibles uniquement par le serveur car chiffrées par un algorithme asymétrique, le serveur est donc le seul à pouvoir connaître les informations identifiables afin de contacter les cas contacts.

Pour résumer

- Limitation des données identifiables.
- Stockage local de l'historique des rencontres.
- L'identifiant temporaire change fréquemment, empêchant les acteurs malveillants de suivre les individus utilisateurs au fil du temps.

Fonctionnalités

Comment échanger des données entre utilisateurs afin de détecter une rencontre ?

Verrou(s) levé(s)

Comment est-il possible techniquement de localiser des utilisateurs, quelles sont les technologie à utiliser ? Quid de la consommation énergétique et des limitations des constructeurs ?

Solution proposée

L'application doit être développée en code natif pour chacun des systèmes d'exploitation **Java pour Android et Swift ou Objectif C pour iOS**, afin d'accéder facilement aux fonctionnalités natives du téléphone à savoir dans notre cas le bluetooth.

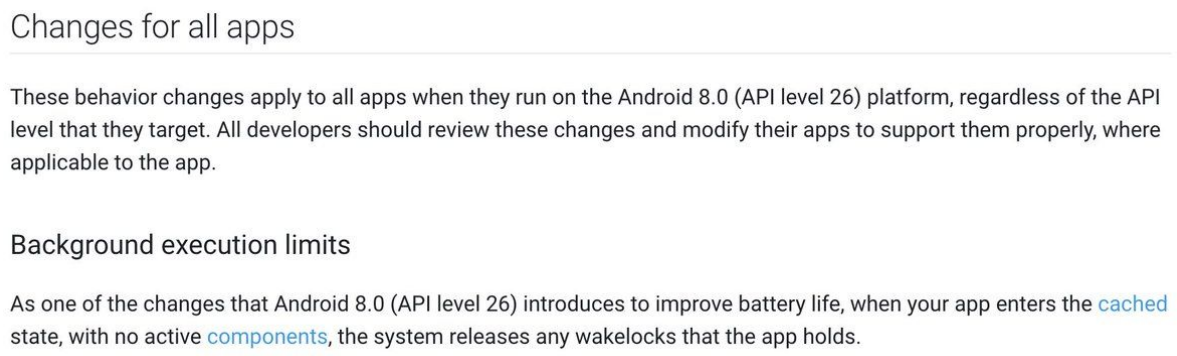
Pourquoi utiliser le bluetooth à la place de la géolocalisation ?

1. L'utilisation de la batterie, le bluetooth est beaucoup moins consommateur en ressources et donc en batterie que la géolocalisation, choix primordial quant on sait que notre application tournera tout le temps en arrière plan.
2. Un argument qui donne du poids à notre débat, la géolocalisation ne fonctionne pas avec précision dans les souterrains notamment dans les métros, et dans le sens inverse la géolocalisation ne prend pas en compte l'altitude et donc on ne peut pas distinguer la hauteur dans des grands bâtiments : grands magasins, immeubles etc..
3. Tous les deux ont une portée de 10 mètres environ, mais le bluetooth est plus facilement interprétable dans le sens où l'on peut déterminer la force du signal et donc avoir un rayon plus restreint.

4. Le bluetooth peut envoyer des données entre mobile ce qui nous évite des échanges avec le serveur.

Limitations technologiques ?

En raison de l'abus de certains développeurs l'utilisation des fonctionnalités en arrière plan est fortement restreinte, en effet il est impossible de les laisser tourner à l'arrière plan.



Changes for all apps

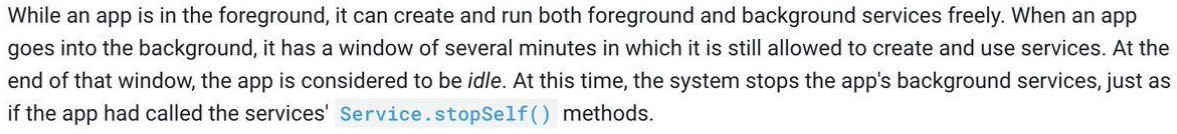
These behavior changes apply to all apps when they run on the Android 8.0 (API level 26) platform, regardless of the API level that they target. All developers should review these changes and modify their apps to support them properly, where applicable to the app.

Background execution limits

As one of the changes that Android 8.0 (API level 26) introduces to improve battery life, when your app enters the cached state, with no active components, the system releases any wakelocks that the app holds.

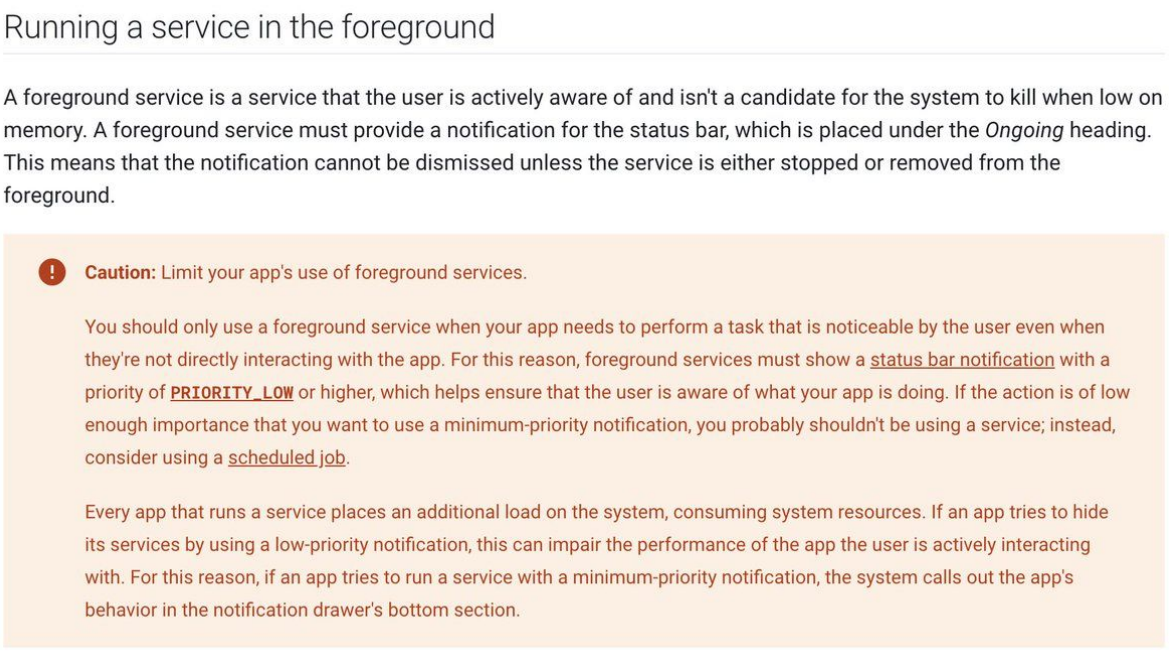
Limitation fixée par Google concernant les applications en arrière plan

Au moment où l'application passera en arrière plan, l'application aura une fenêtre de quelques minutes pour créer et utiliser ses services. Ensuite, le système tuera tous ses services.



While an app is in the foreground, it can create and run both foreground and background services freely. When an app goes into the background, it has a window of several minutes in which it is still allowed to create and use services. At the end of that window, the app is considered to be *idle*. At this time, the system stops the app's background services, just as if the app had called the services' `Service.stopSelf()` methods.

Il faut donc d'après la documentation Google utiliser un service au premier plan, côté Android on parle de foreground service mais si une application utilise abusivement un foreground service, Android affichera une notification à l'utilisateur en alertant qu'une application abuse des ressources de son téléphone.



Running a service in the foreground

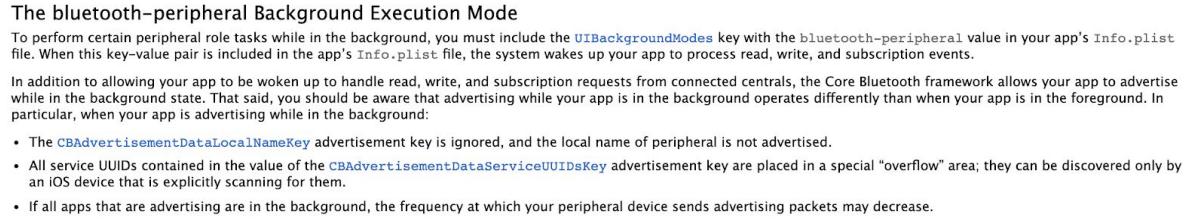
A foreground service is a service that the user is actively aware of and isn't a candidate for the system to kill when low on memory. A foreground service must provide a notification for the status bar, which is placed under the *Ongoing* heading. This means that the notification cannot be dismissed unless the service is either stopped or removed from the foreground.

Caution: Limit your app's use of foreground services.

You should only use a foreground service when your app needs to perform a task that is noticeable by the user even when they're not directly interacting with the app. For this reason, foreground services must show a status bar notification with a priority of `PRIORITY_LOW` or higher, which helps ensure that the user is aware of what your app is doing. If the action is of low enough importance that you want to use a minimum-priority notification, you probably shouldn't be using a service; instead, consider using a scheduled job.

Every app that runs a service places an additional load on the system, consuming system resources. If an app tries to hide its services by using a low-priority notification, this can impair the performance of the app the user is actively interacting with. For this reason, if an app tries to run a service with a minimum-priority notification, the system calls out the app's behavior in the notification drawer's bottom section.

Ces mêmes limitations existent chez Apple en effet il est impossible d'échanger des données aux autres téléphones lorsque les applications sont en arrière plan.



The bluetooth-peripheral Background Execution Mode

To perform certain peripheral role tasks while in the background, you must include the `UIBackgroundModes` key with the `bluetooth-peripheral` value in your app's `Info.plist` file. When this key-value pair is included in the app's `Info.plist` file, the system wakes up your app to process read, write, and subscription events.

In addition to allowing your app to be woken up to handle read, write, and subscription requests from connected centrals, the Core Bluetooth framework allows your app to advertise while in the background state. That said, you should be aware that advertising while your app is in the background operates differently than when your app is in the foreground. In particular, when your app is advertising while in the background:

- The `CBAdvertisementDataLocalNameKey` advertisement key is ignored, and the local name of peripheral is not advertised.
- All service UUIDs contained in the value of the `CBAdvertisementDataServiceUUIDsKey` advertisement key are placed in a special "overflow" area; they can be discovered only by an iOS device that is explicitly scanning for them.
- If all apps that are advertising are in the background, the frequency at which your peripheral device sends advertising packets may decrease.

L'application doit être en premier plan et en action pour être parfaitement fonctionnelle.

Comment lever ces limitations technologiques ?

Google et Apple ont travaillé main dans la main pour fournir une API permettant le contact tracing, celle-ci permet de lever le verrou sur cette limitation du bluetooth tout en respectant la vie privée des utilisateurs et l'abus des développeurs car cette API est disponible pour des comptes développeurs gouvernementaux ou institutionnels.

Privacy-Preserving Contact Tracing

Notre solution pour localiser les personnes est d'échanger des informations lors de la rencontre de celles-ci mais il existe des limitations technologiques imposées par les constructeurs, le bluetooth reste viable mais avec ses limitations.

Comment prévenir les cas contacts ?

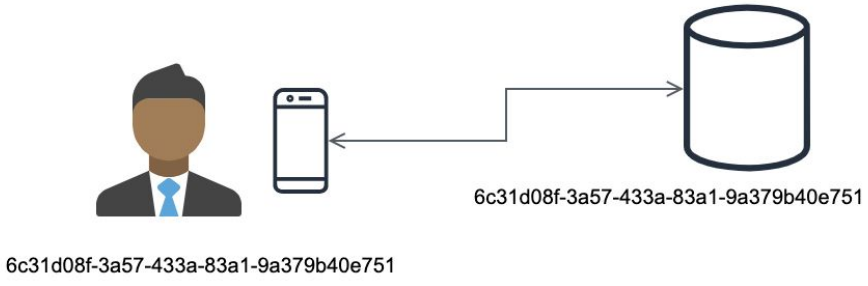
Verrou(s) levé(s)

Maintenant nous supposons que nous pouvons échanger les données via bluetooth, nous allons vous exposer quelles sont les données qui circulent, leur cycle de vie ? où sont-elles transmises ? comment sont-elles anonymisées ?

Solution proposée

A l'installation de l'application un token d'identification est demandé à l'API, ce token unique permettant d'identifier un téléphone, le token est de type a **universally unique identifier (UUID)** sous ce format : 6c31d08f-3a57-433a-83a1-9a379b40e751

Notre token est le seul moyen d'identifier un téléphone, nous ne stockons aucune information personnellement identifiable.



Rappelons maintenant le principe, dès que nous allons rencontrer une personne nous allons échanger des tokens que chacun va enregistrer, bien évidemment sans action de l'utilisateur. Mais ceci pose certains problèmes, nous générons actuellement des tokens statiques une

personne mal intentionnée pourrait très bien lire les tokens sur l'application et suivre les mouvements de chacun ce qui pose un problème de sécurité.

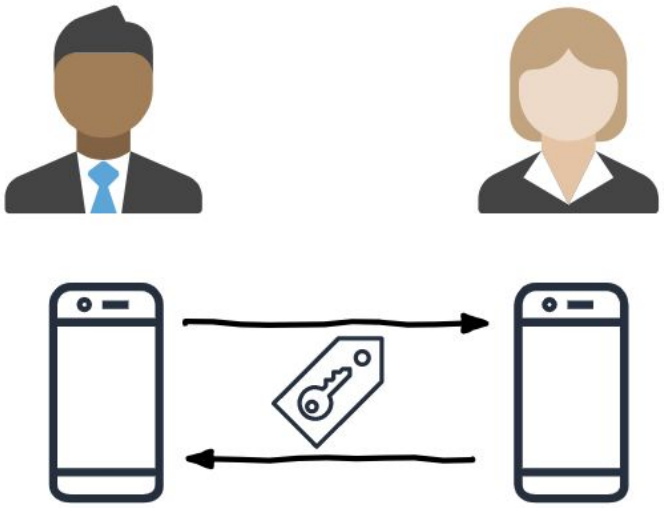
Dans ce cas nous avons trouvé une solution, nous allons générer **des tokens de rencontre.**

Les tokens de rencontre sont générés via un chiffrement à l'aide du protocole RSA, le client effectue un appel à l'API afin de demander un token de rencontre, nous pourrions pu très bien également générer les tokens de rencontre sur le smartphone directement les deux solutions sont viables, mais pour des raisons d'homogénéité nous générons le token de rencontre côté serveur comme l'est le token d'identification.

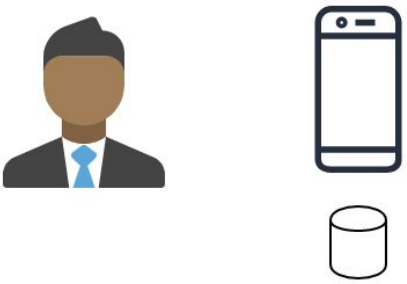
Token d'identification (36 caractères)	Date au moment de la génération format yyyy.MM.dd HH:mm:ss	Date d'expiration du jeton yyyy.MM.dd HH:mm:ss	Vecteur d'initialisation (16 caractères)

Toutes ces informations sont chiffrées à l'aide de la clé publique et pourront donc être uniquement déchiffrés par le serveur via la clé privée.

Comme vous pouvez le voir le token a une date d'expiration nous l'avons fixé à 20 minutes, ce qui permet de ne pas tracer les personnes que l'on rencontre, aucune information personnelle, aucune donnée statique.



L’ensemble des tokens est stocké sur le smartphone en local pour éviter de les envoyer au serveur quand on croise une nouvelle personne.



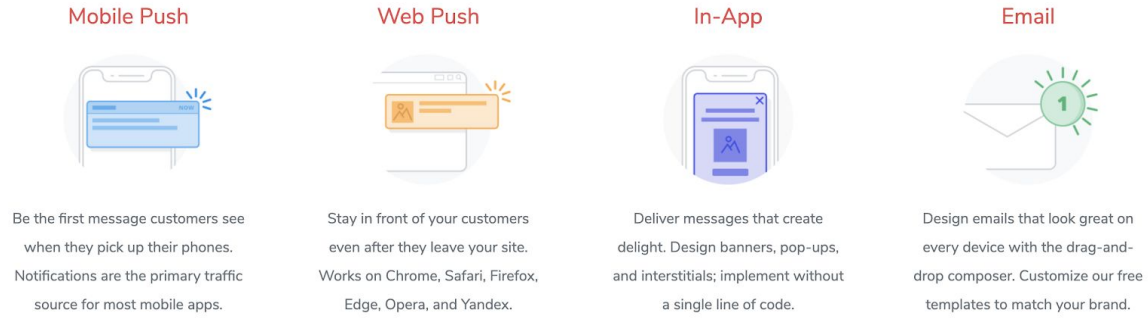

```
{
 "tokensId" :  [
    {
    "tokenTemporary" : "zefonaonoezn",
    "startDate" : "2020.05.10 10:49:00",
    "expriraryDate" : "2020.05.10 11:09:00"
    },
    {
    "tokenTemporary" : "mlgkormkt",
    "startDate" : "2020.05.10 10:55:00",
    "expriraryDate" : "2020.05.10 11:15:00"
    }
 ]
}
```

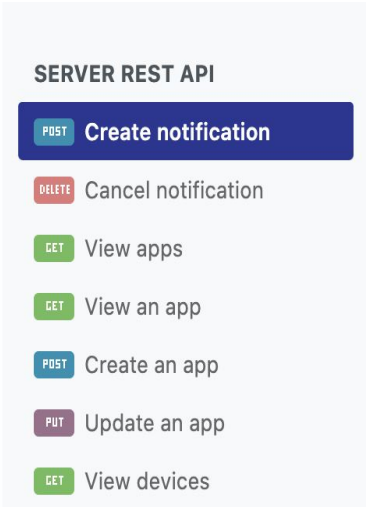

Une fois qu’une personne est testée positive au coronavirus elle entre le numéro de son test puis l’application se charge d’envoyer les différents token de rencontre au serveur, le serveur déchiffre l’ensemble des tokens de rencontre pour récupérer les tokens d’identification, à partir de cela nous pouvons contacter les personnes en question.

Avec cette architecture aucune information n’est stockée côté serveur tout est en local.

Comment prévenir l’ensemble des personnes maintenant que nous savons qui prévenir ?
Nous n’avons aucun numéro de téléphone, comment envoyer des informations à l’application à partir du token unique ?

C’est là que le framework one signal entre en jeu à partir de celui ci les devices s’abonnent à une file et le serveur pourront leur envoyer tout type de notification dépendant de leur device.





L'api de one signal se repose sur le protocole Rest, pratique pour l'intégrer dans notre API Spring Boot.

La documentation de one signal est très bien fournie et propose énormément de code exemple pour énormément de framework mobile, on a aucune restriction au niveau de la technologie.

Pour rentrer dans les détails du fonctionnement, on crée une application sur un dashboard, la création de l'application nous fournira un "app id" c'est cet identifiant qui va être la clé pour identifier notre file, les consommateurs devront écouter sur celle file, la librairie one signal se charge du reste.

```swift
func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[UIApplicationLaunchOptionsKey: Any]?) -> Bool {

  //Remove this method to stop OneSignal Debugging
  OneSignal.setLogLevel(.LL_VERBOSE, visualLevel: .LL_NONE)

  //START OneSignal initialization code
  let onesignalInitSettings = [kOSSettingsKeyAutoPrompt: false, kOSSettingsKeyInAppLaunchURL: false

  // Replace 'YOUR_ONESIGNAL_APP_ID' with your OneSignal App ID.
  OneSignal.initWithLaunchOptions(launchOptions,
    appId: "YOUR_ONESIGNAL_APP_ID",
    handleNotificationAction: nil,
    settings: onesignalInitSettings)
```

Exemple de d'abonnement pour une application native iOS.

If you're looking to send notifications to a specific user device:

1. Get the user's userId/playerId with the `getUserIds` method on the Web Push SDK or `getPermissionSubscriptionState` method on the Mobile SDK you are using.
You can also send OneSignal your own unique user ids as "external_user_ids" using our `setExternalUserId` method on all our SDKs.

- For testing you can use the 'Player ID' shown in `All Users` (*you can force kill your app and open it again to bring your device to the top of the list.*)

Pour envoyer une notification à un seul et unique device il doit s'enregistrer avec son id unique généré lors de l'installation de son application côté one signal il s'appelle **"external user id"**

Développement du POC

Maintenant que nous avons vu la théorie voyons comment nous avons mis en place cette architecture, vu le temps imparti nous allons simuler l'échange des tokens de rencontre sur une interface graphique web, nous pouvons nous le permettre puisque one signal peut envoyer des notifications web.
La partie front est développé en Angular 9 et la partie back est développé en spring boot

POC application covid

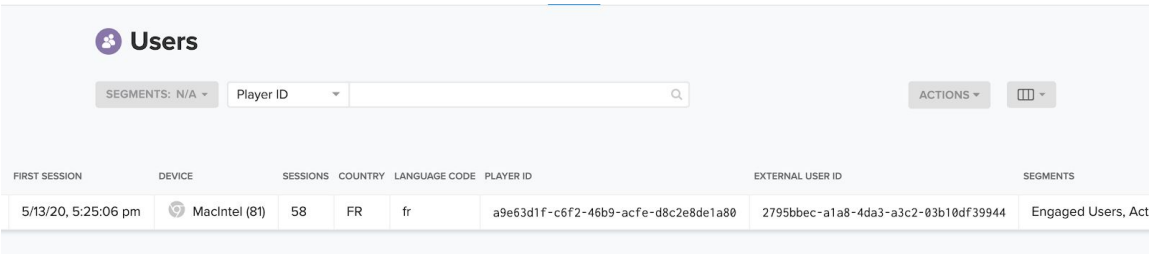
Ne fonctionne pas avec **Safari**

Veillez à bien autoriser l'envoie de notifications, les notifications proviennent du framework one signal

Toutes les données que vous voyez proviennent du back

Votre id de téléphone unique : **2795bbec-a1a8-4da3-a3c2-03b10df39944**

Simuler une rencontre



En cliquant sur "simuler une rencontre" vous générez un token de rencontre à partir de votre token d'identification, bien sûr normalement il devrait être envoyé par le téléphone que l'on rencontre mais pour des raisons évidentes il est plus simple de faire ainsi.

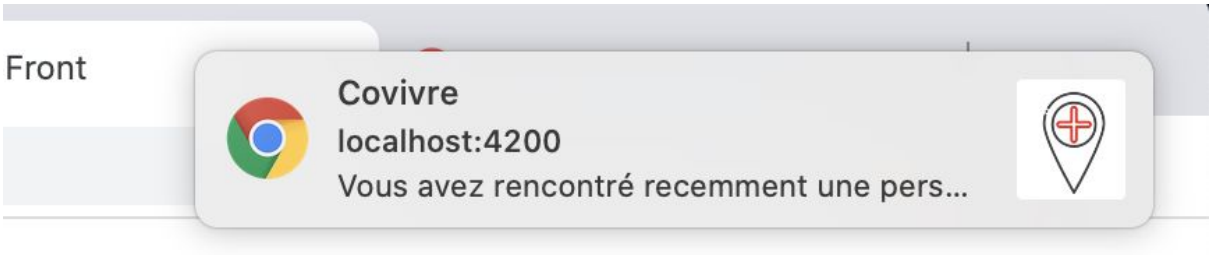
Vos tokens de rencontre

OFkZfwiWWObk6H4uiYIGt3+I/iRT3YZAd/pWAZDIc1vKUgLBMQIxsvyxfurPoZScVrzj1UGTIO

Pour le moment 1 personne est à contacter

Alerter les utilsateurs

En cliquant sur simuler une rencontre l'ensemble des tokens est envoyé au serveur pour être déchiffré et contacter l'ensemble des personnes, puisque les tokens de rencontre sont générés via votre token d'identification vous allez recevoir la notification sur la même page.



Aucune action du POC n'est "mockée" et il pourrait être utilisé comme tel pour une application de production.

Lien du GitHub : https://github.com/MrADOY/covivre

Conclusion

Pour conclure, nous avons démontré qu'il existait des solutions pour contourner les verrous pour l'échange de données utiliser l'API développé par Google et Apple, même si la solution du bluetooth reste fiable mais avec quelques défauts. Concernant l'échange des données et l'anonymisation de celles-ci nous avons démontré qu'il était possible de contacter une personne sans avoir aucune information personnelle sur celle-ci, ce qui lève entièrement le verrou concernant l'anonymisation et du respect de la vie privée de l'utilisateur.

Sources

Limitation technologique Google
https://developer.android.com/about/versions/oreo/android-8.0-changes#abll
Nouvelle API Google / Apple
https://developer.apple.com/documentation/exposurenotification/building_an_app_to_notify_users_of_covid-19_exposure
Protocole RSA
https://fr.wikipedia.org/wiki/Chiffrement_RSA
One Signal
https://documentation.onesignal.com/docs
Angular
https://angular.io/docs
Spring Boot
https://spring.io/projects/spring-boot