Génie Logiciel

SIMULATEUR STOPCOVID
RAPPORT

20 septembre 2020

Alexis Plaquet
Otman Azziz

Université Claude Bernard Lyon 1

Table des matières

Chapitre 1

Éthique

1.1 Enjeux pour la vie privée

Une solution envisagée pour lutter contre le virus serait la création d'une application chargée de suivre les contacts entre chaque individu, et d'avertir ces derniers lorsqu'ils sont susceptible d'avoir été en contact avec le virus.

Posséder de telles données sur l'ensemble de la population serait une atteinte grave à la vie privée, et pourraient facilement être détournées à des fins malveillantes. Dans le pire des cas il serait possible de retracer l'itinéraire de chaque personne et de tous les contacts qu'elle a eu avec des usagers.

Il parait donc impensable de mettre en place officiellement une telle application sans se préoccuper de la sécurité de la vie privée de chacun, d'autant plus que ce genre d'enjeux devient de plus en plus important au sein de l'Union Européenne.

1.2 Solutions techniques existantes et leur approche sur la vie privée

1.2.1 Approche commune à toutes les solutions

Le problème le plus évident d'une application de traçage est que ses utilisateurs puissent se traquer, et traquer les malades.

Ce problème est résolu de façon plus ou moins similaire dans tous les protocoles : les utilisateurs émettent à l'aide du bluetooth des identifiants temporaires. La façon dont ces derniers sont générés varie avec le protocole. On appelle généralement ces identifiants temporaires des "pseudonymes" ou "clés". Cette approche empêche de remonter à un utilisateur à partir des données qu'il transmet, ce qui serait possible avec un identifiant unique.

Les utilisateurs prouvent leur infection à l'aide d'un certificat numérique délivré par les autorités chargées des dépistages (afin d'éviter les abus).

1.2.2 Protocole centralisé

Une approche possible est de centraliser les données sur un serveur : c'est ce dernier qui s'occupe alors de prévenir les utilisateurs qu'ils ont eu un contact avec un individu infecté. Pour

cela, les utilisateurs infectés transmettent l'ensemble des clés qu'ils ont rencontré les 14 derniers jours.

Le protocole ROBERT utilisé pour StopCovid rentre dans cette catégorie.

1.2.3 Protocole décentralisé

L'autre approche possible est que les utilisateurs transmettent les clés qu'ils ont généré ces 14 derniers jours lorsqu'ils sont infectés. Et c'est alors aux autres utilisateurs de vérifier si ils ont croisé ces clés.

Les protocoles PACT et DP-3T rentrent dans cette catégorie.

1.3 Avantages et inconvénients des approches possibles

Le protocole décentralisé offre une plus grande liberté aux utilisateurs, qui peuvent savoir quand ils ont rencontré la personne infectée, il est alors plus simple d'identifier la personne.

Le protocole centralisé permet mieux de garder l'anonymat au prix d'un diagnostic moins précis (impossible de savoir si la personne infectée a été rencontrée à une heure où l'utilisateur portait un masque ou non par exemple), et donne plus de responsabilité à l'authorité.

1.4 Solution technique utilisée dans notre projet

Notre projet utilise un protocole qui se rapproche de PACT ou DP-3T, le serveur ne fait que stocker des clés appartenant à des utilisateurs infectés.

1.5 Alternatives au suivi de contacts par proximité

Il existe des approches plus radicales au suivi de proximité actuellement en place. Ces dernières portent largement atteinte à la vie privée et sont mises en place dans des pays où ce genre de surveillance de masse ne pose pas de problème.

Une première approche est celle utilisée en Israel qui consiste à traquer directement les données des téléphones des utilisateurs [1].

La Corée du Sud a adopté une approche similaire, et suit l'activité de ses citoyens sur ses différents services tels que les historiques de dépenses, les données GPS et les caméras de surveillances. [2]

L'approche la plus extrême se trouve en Chine où tout le processus de vérification est intégré aux services proposés par Alibaba Group, et où l'installation de l'application est obligatoire. Les utilisateurs se voient attribué un code couleur (transmis automatiquement à la police en cas de changement). Les détails du protocoles n'ont pas été révélés [3].

Chapitre 2

Projet

2.1 Présentation

Le sujet du projet était de créer un simulateur d'application type "StopCovid", permettant de simuler plusieurs utilisateurs de l'application et leurs différentes actions.

L'objectif étant de réussir à structurer au mieux le projet en utilisant les concepts SOLID, GRASP, ainsi que les différents design pattern connus.

2.2 Structure et design patterns

2.2.1 MVC

On trouve dans le projet plusieurs MVC. Le premier est le MVC simulateur, qui est chargé d'agréger les MVC des différents acteurs (dans notre projet, il n'agrège que ceux des utilisateurs).

Le second est le MVC utilisateur, représentant les différents utilisateurs de l'application. Il est lié à deux modèle :

- Son modèle utilisateur local, qui lui est propre et qui représente les données locales de l'utilisateur (typiquement stockées en RAM puis sauvegardées sur l'espace de stockage du téléphone de l'utilisateur).
- Le modèle serveur "distant" qui représente le modèle tel qu'il serait sur les serveurs.

Tous les MVC sont basés sur la Figure 2.2, où les évènements sont gérés par un pattern Observer. Nous avons porté une attention particulière au découplage des modèles des contrôleurs et vues.

La Figure 2.3 montre les relations entre les deux MVC, avec à gauche le MVC simulateur, à droite les MVC utilisateurs, et au centre le modèle serveur. On observe ainsi le lien de composition qui relie les modèles utilisateurs au modèle simulateur, ainsi que celui qui relie les vues utilisateurs à la vue simulateur.

2.2.2 Modèle serveur

Le modèle serveur est composé d'une classe ServerModel qui contient tous les éléments nécessaires au fonctionnement du serveur. Afin de respecter au mieux le principe d'**inversion de dépendance**, dès qu'un de ces éléments peut être implémenté de plusieurs façon, nous avons

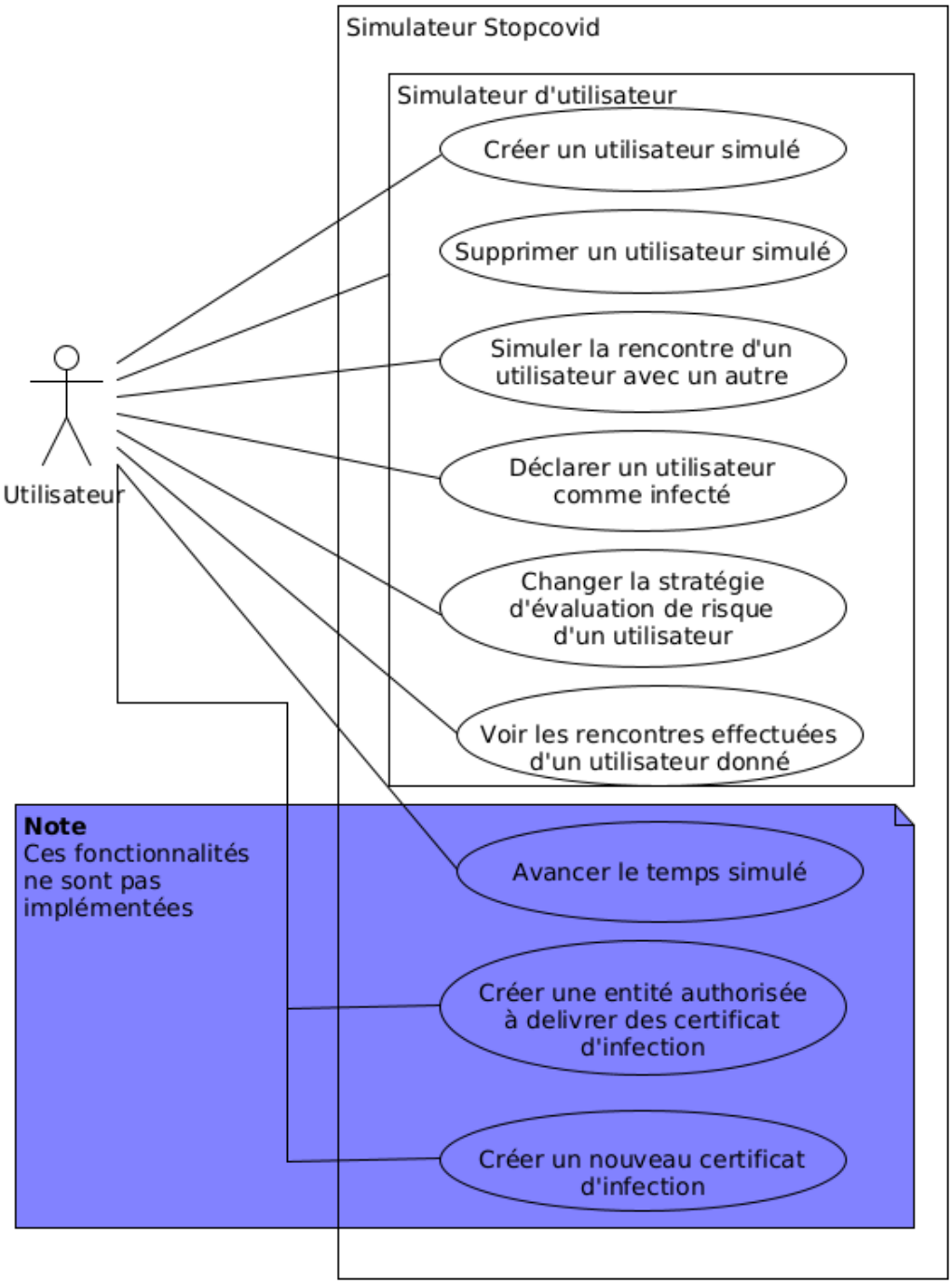

FIGURE 2.1 – Diagramme de cas d'utilisation du simulateur.

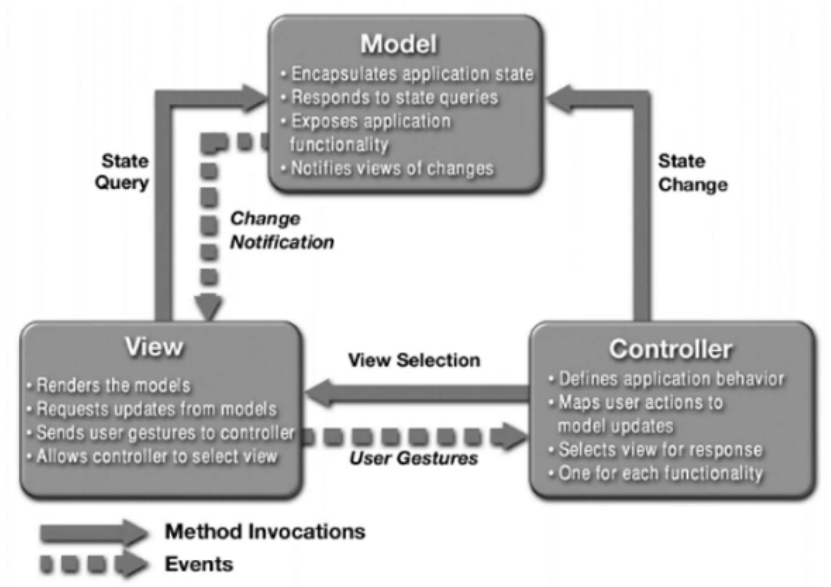

FIGURE 2.2 – Diagramme tiré du CM. Source Originale : BluePrint Java (Sun).

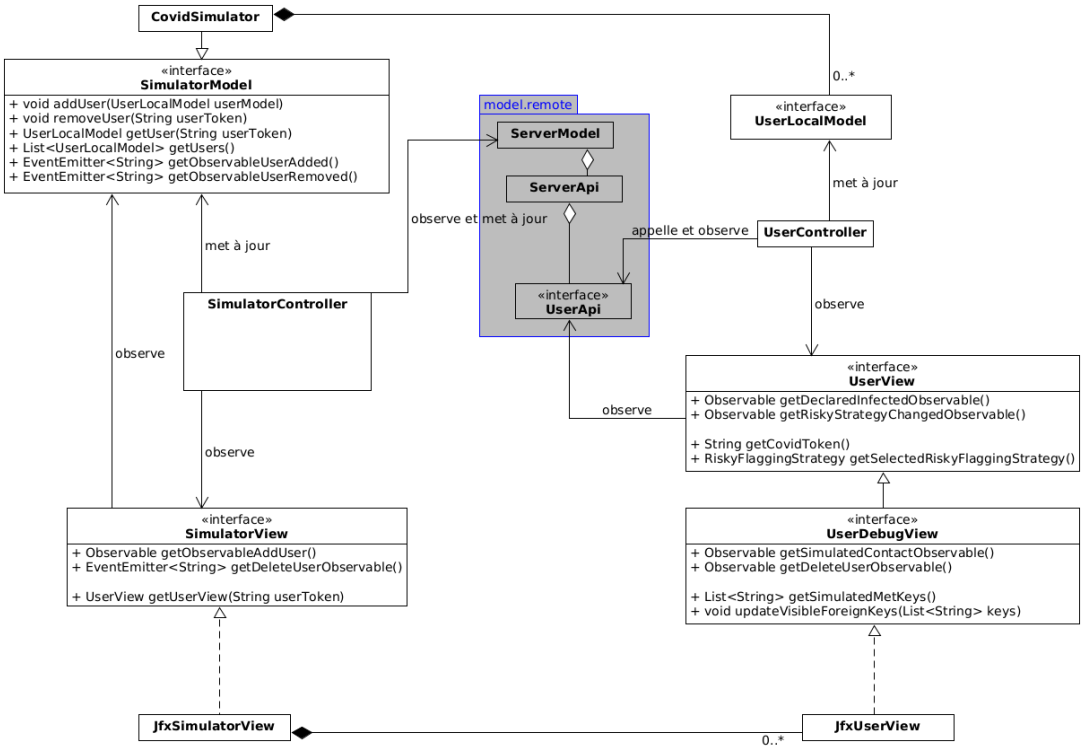

FIGURE 2.3 – Diagramme UML des MVC du projet.

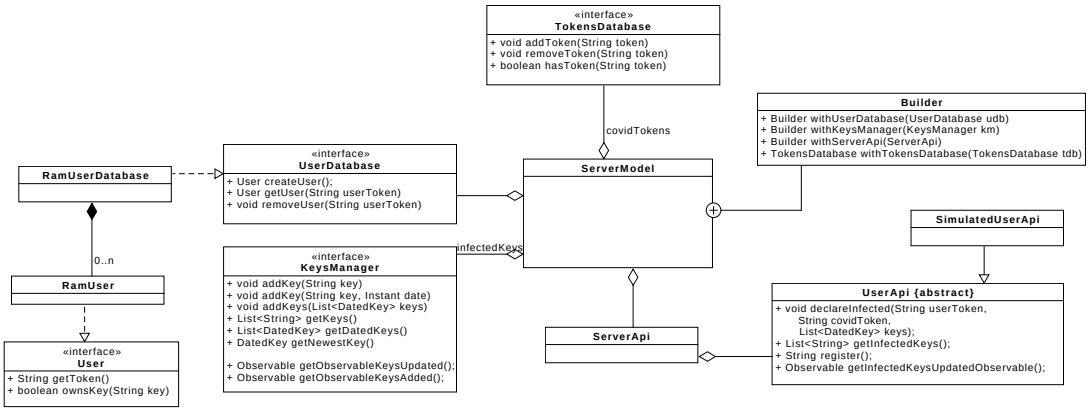

FIGURE 2.4 – Diagramme UML du modèle serveur.

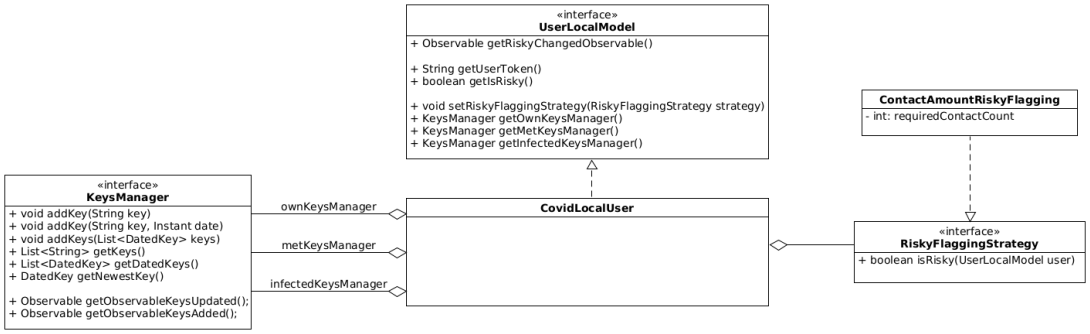

FIGURE 2.5 – Diagramme UML du modèle utilisateur local.

utilisé des interfaces ou classes abstraites. Ainsi si on voulait implémenter des bases de données utilisant MySQL il suffirait de créer une classe dédiée implémentant l'interface requise.

Seul le ServerModel lui-même ainsi que la classe ServerApi n'implémentent pas d'interface ou n'héritent pas de classe abstraite, en effet ils ne contiennent aucune logique propre et ne font que référencer leurs divers composants, il semble donc peu vraisemblable que l'on veuille l'implémenter différemment (et dans ce cas il suffirait sûrement d'hériter les classes).

Le modèle possède une API contenue dans ServerApi et qui n'est actuellement constituée que d'une UserApi. C'est cette classe qui se chargera d'appliquer les différentes actions appelées par les utilisateurs (au sens d'utilisateurs de l'application StopCovid), et est une utilisation du pattern **facade**. Les utilisateurs auront une référence vers UserApi et non une référence vers le ServerModel directement.

Enfin, la construction de ce modèle s'effectue à l'aide d'un **builder** interne à la classe ServerModel (`ServerModel.Builder`), ce qui permet une construction plus propre du modèle, étant donné le nombre d'éléments qui le compose.

2.2.3 Modèle utilisateur

Le modèle utilisateur a pour rôle de stocker ses propres clés, les clés qu'il rencontre, et les clés infectées récupérées sur le serveur, il doit aussi être capable de déterminer s'il y a un risque qu'il soit contaminé. Il possède donc trois KeysManager chargé du stockage des clés, et une référence à une RiskyFlaggingStrategy qui est chargée de déterminer si l'utilisateur est possiblement infecté.

L'interface RiskyFlaggingStrategy est évidemment un pattern **stratégie**, il est donc possible de changer celle d'un utilisateur à n'importe quel moment. Actuellement, seul une stratégie basé sur le nombre de rencontres avec un utilisateur infecté est implémentée. Mais on pourrait imaginer une stratégie qui prend aussi en compte le temps écoulé depuis la rencontre.

2.3 Fonctionnalités non implémentées

2.3.1 "Extensions obligatoires"

Impossibles à cause du protocole

- N'envoyer que les contacts rencontrés plusieurs fois
- Sélection des 10 contacts les plus fréquents

Impossible mais implémentée de façon altérée

- Choix de la stratégie de sélection des contacts à envoyer : l'utilisateur n'envoie plus les contacts, transformé en choix de la stratégie pour déterminer l'état de risque.

2.3.2 Implémentations alternatives d'interfaces

Il était prévu d'implémenter les différentes interfaces (bases de données principalement) de différente façons, cependant nous avons manqué de temps pour ça. Toutefois l'architecture reflète cet objectif et il devrait être possible de le faire sans modifier la base de code existante.

2.3.3 Aspects non implémentés du protocole

Par manque de temps, nous n'avons pas pu implémenter la génération cryptographique des clés temporaires, permettant au serveur de vérifier leur appartenance à l'utilisateur qui les envoie. Il était aussi prévu d'implémenter la simulation des organismes délivrant les Covid Token (certificats).

2.3.4 Simulation de temps

Le plus gros point faible du simulateur en l'état actuel est l'impossibilité de simuler le passage du temps.

L'ajout de cette fonctionnalité serait accompagné de **décorateurs** sur KeysManager (ou autre pattern permettant d'altérer son fonctionnement) qui permettraient d'en avoir une version qui efface automatiquement les clés vieilles de X jours.

Le système de clés/pseudonyme prendrait alors son sens, puisque actuellement, les utilisateurs n'ont qu'une seule clé personnelle, et ne les régénèrent jamais.

On retrouve un début d'implémentation de cette fonctionnalité dans la classe FakeTime qui utilise le pattern **singleton**, et qui aurait été appelée à la place de `Instant.now()`.

2.4 Tests

La partie test est sûrement la partie la moins aboutie du projet, nous avons toutefois créé quelques tests sur les fonctionnalités de base du logiciel.

Les tests suivent tous la structure *Given, When, Then.* Des explications brève sur le contenu des tests sont disponibles dans le code, notre place dans ce rapport étant limitée.

Tests sur DataKeyCollection

- Ajout de clés
- Récupération de la clé la plus récente
- Test des observables

2.4.1 Tests sur les modèles

- Création d'utilisateurs suite à leur "inscription" sur le serveur
- Ajout d'utilisateurs au modèle simulateur
- Stockage des rencontres entre utilisateurs dans leurs modèles respectif
- Utilisateur se déclarant infecté
- Suppression d'utilisateurs dans le modèle simulateur
- Changement de stratégie d'évaluation de risque

Bibliographie

[1] Holmes, Oliver
(2020-03-17). *Israel to track mobile phones of suspected coronavirus cases*. The Guardian. ISSN 0261-3077. R
`https://www.theguardian.com/world/2020/mar/17/israel-to-track-mobile-phones-of-suspected-coronavirus-cases`

[2] Ackerman, Spencer
(2020-03-26). *It's Probably Too Late to Use South Korea's Trick for Tracking Coronavirus*. The Daily Beast.
`https://www.thedailybeast.com/too-late-to-use-south-koreas-coronavirus-tracking-technique`

[3] Paul Mozur, Raymond Zhong and Aaron Krolik
(2020-03-01). *In Coronavirus Fight, China Gives Citizens a Color Code, With Red Flags*. The New York Times
`https://www.nytimes.com/2020/03/01/business/china-coronavirus-surveillance.html`