Université Claude Bernard Lyon 1

Rapport Mif01 :
Gestion de Projet et Génie Logiciel

Dorian SALMI
Raoufdine SAID

Groupe gr-46
Septembre 2020

Sommaire

Introduction

Ce rapport présente le résultat du travail réalisé dans le cadre de l'unité d'enseignement "Gestion de projet et génie Logiciel".

Le projet correspondant à ce travail consiste à partir d'un simulateur de l'application *StopCovid* et de le retravailler en appliquant les bonnes pratiques de la gestion de projet et de la conception logicielle.

Design Patterns

Dans cette partie nous allons considérer les designs patterns que nous avons mis en place au cours de la réalisation du projet. Le diagramme complet de l'application (généré via IntelliJ) est disponible en Annexe 2 au besoin.

- **MVC**

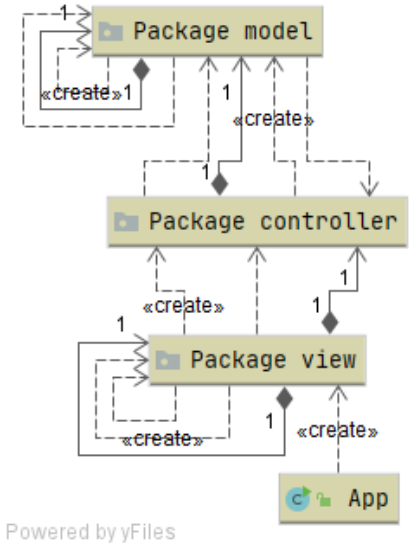

Figure 1 Diagramme UML des packages

L'objectif dans la mise en place de ce pattern est avant tout de dissocier les différentes fonctions de notre application. On a le cœur du simulateur dans le package « *model* », toute l'interface utilisateur dans le package « *view* » et le package « *controller* » fait le lien entre les deux précédents. Cela nous permet d'assurer la réutilisabilité de ceux-ci.

Dans le package « *view* » que nous avons réalisé, l'interface utilisateur présente des informations et fonctionnalités à la fois des utilisateurs simulés et du serveur associé. On pourrait aisément imaginer replacer celui-ci afin de n'afficher que l'interface d'un seul utilisateur simulé par exemple. Cela influe la conception jusqu'au découpage à l'intérieur des packages.

En effet notre package « *view* » contient trois classes : *ViewUser* (interface d'un utilisateur), *ViewServer* (interface du serveur), et *ViewApplication* (utilisant les deux précédentes).

- **Strategy**

Une fois que les utilisateurs se déclare comme infectés, on souhaite pouvoir sélectionner de manières différentes les cas contacts à notifier. Deux stratégies mises en place sont la notification de tous ces cas, peu importe la qualité de ce contact (que l'on mesure en nombre répété de contact avec un même utilisateur), ainsi que de ne notifier que les utilisateurs rencontrés à plusieurs reprises.

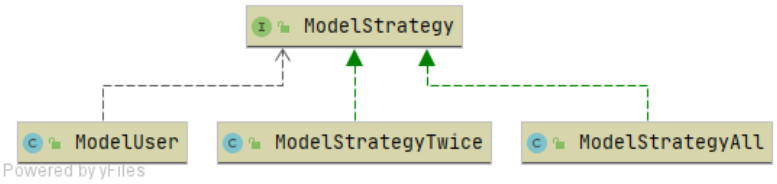

Figure 2 Strategy - Interface Fonctionnelle

Le design pattern *strategy* nous permet de réaliser cela au travers d'un interface fonctionnelle.

- **Expert**

La plupart des informations importantes dans ce simulateur sont fonction de quel utilisateur on considère (à savoir la liste des contacts d'un utilisateur et leur fréquence). Autrement dit la classe de notre modèle représentant l'utilisateur de *StopCovid* (*ModelUser*) doit être experte à ce sujet et stocker l'information.

- **Singleton**

Dans cette application, on peut (dans l'absolu) créer autant d'utilisateur que l'on souhaite, mais on souhaite s'assurer qu'un seul et unique serveur existe puisque les critères de notification d'un utilisateur cités dans la partie précédente sont dictés par celui-ci.

Afin de pouvoir s'assurer de cela nous avons appliqué le pattern du singleton, c'est-à-dire que ce serveur (correspondant à la classe *ModelServer*) n'est pas instancié par l'application. Une seule instance existe statiquement et est simplement appelée par le contrôleur. Cette instance et le constructeur sont privés (comme indiqué par le cadenas sur l'image ci-contre).

Ainsi au moment du lancement de l'application et de l'instanciation de toutes les classes, on ne fait que récupérer l'instance imposée, donc jamais plusieurs instances de cette classe ne coexisteront.

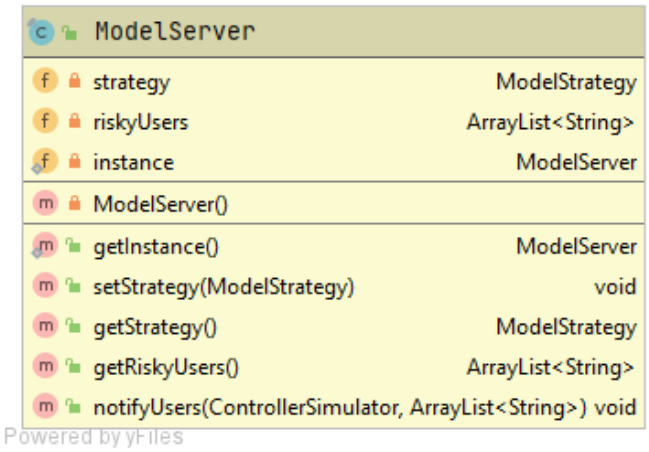

Figure 3 Contenu de la classe ModelServer

Ethique

Nous n'avons travaillé ici que sur un simulateur de la véritable application *StopCovid*, mais cela nous permet tout de même d'approcher la question de l'éthique de celle-ci. Si le principe de permettre à ses utilisateurs de savoir s'ils ont pu se trouver dans une situation à risque est louable, un tel système est contraint de collecter des données personnelles d'une manière ou d'une autre. La récupération de ces informations et leur utilisation, voir même leur utilité est sujet à discussion.

Il est intéressant de se pencher sur un pays comme la Chine qui a lancé une solution analogue à *StopCovid* dès février. Dans un article sorti alors que l'application française était encore un projet en cours, Louis Neveu de Futura Science nous explique que ces solutions de traçage étaient déjà obligatoires dans plusieurs villes au moment de l'écriture[1]. De plus un utilisateur est identifié par un code QR généré avec des données telles que le numéro présent sur une pièce d'identité.

D'après un article de France info[2], la Commission Nationale de l'informatique et des libertés s'est exprimée à plusieurs reprise dans les derniers mois. Elle exprimait le 26 avril qu'une telle application

[1] https://www.futura-sciences.com/tech/actualites/smartphone-fonctionne-application-covid-19-chine-80593/

[2] https://www.francetvinfo.fr/sante/maladie/coronavirus/stopcovid-quelles-sont-exactement-les-recommandations-de-la-cnil-concernant-l-application-de-tracage-de-contacts_4110337.html

ne devait pas influer l'accès aux tests ou aux soins, c'est-à-dire ne pas rendre moins prioritaires ceux qui n'auraient pas souhaité l'utiliser, et a ajouté le 26 mais que cette mesure doit être temporaire et exclusivement sur la base du volontariat. Le 20 juillet, la CNIL a aussi saisi le gouvernement français au sujet de « plusieurs irrégularités » qu'elle a constaté (qui aurait été corrigées, les demandes ayant cessé depuis début septembre). Un article du Monde du 16 juin[3] déclare que l'application enregistrait tous les contacts, indépendamment du temps durant lequel les utilisateurs étaient à proximité contrairement à ce qui avait été annoncé.

StopCovid n'est d'ailleurs pas le seul projet de ce type. En Allemagne une application open source du nom de *Corona-Warn* est actuellement développée[4], permettant la même chose que son homologue française. Le fait que celle-ci soit open source rend tout de même la collecte de donnée plus transparente. Ces applications se basent en grande partie sur la technologie *bluetooth*[5]. Dans le cas particulier du *StopCovid* français les personnes l'utilisant sont identifiées par des pseudonymes. Une autre manière de faire proposée par les Google et Apple est de rendre cet identifiant dynamique dans l'optique de limiter la possibilité de déterminer l'identité d'un utilisateur. Cependant cela pose des questions de stockage puisqu'un utilisateur peut être notifié dans les deux semaines suivant le contact, il faut donc conserver (sur un serveur ou localement) l'historique de ces identifiants.

Il faut cependant reconnaître qu'aucune méthode ne permet de dissimuler de manière absolue l'identité d'un utilisateur, et que bien que difficile, celle-ci est toujours possible à recouper.

Si l'on doit considérer tout ce qui précède et juger le simulateur que nous avons conçu, on peut souligner plusieurs faits. Premièrement les utilisateurs ne sont identifiés que par un identifiant de la forme *userX*, garantissant l'anonymité de ceux-ci. De plus, le serveur n'est sollicité que dans les cas où un utilisateur doit être notifié et ce sans transmettre l'utilisateur qui se déclare comme infecté. On limite ainsi la collecte de données effectuée au stricte nécessaire.

Cela étant notre version n'est encore une fois qu'un simulateur assez limité, et nous n'avons par exemple pas eu à se poser la question de la solution matérielle.

Tests

Tous les tests n'ont pas pu être implémentés faute de temps. Nous avons dû nous restreindre aux packages *model* et *controller*.

Pour chacune des classes destinées à être instanciées (*ModelUser*, *ModelServer*, ModelContact ou *ControllerSimulator*), celles-ci sont d'abord testées indépendamment. Le fichier de tests associé à une classe, typiquement *ModelUserTest* pour *ModelUser*, possède une méthode destinée à être appelée avant d'effectuer les vérifications.

Elle remplit basiquement le rôle d'appel du constructeur (ou de récupération de l'instance dans le cas de *ModelServer*) avec des paramètres fixés. Cela nous permet de connaître précisément l'état de l'objet testé.

On peut ensuite vérifier le résultat produit par chacune des méthodes de la classe, en fournissant encore une fois des paramètres pour lesquels on connaît le comportement.

[3] https://www.lemonde.fr/pixels/article/2020/06/16/l-application-stopcovid-collecte-plus-de-donnees-qu-annonce_6043038_4408996.html

[4] https://www.coronawarn.app/en/

[5] https://linuxfr.org/users/codefish/journaux/a-propos-des-protocoles-de-tracage-pour-le-covid-19-google-apple-vs-inria-fraunhofer

Conclusion

Ce projet a été réalisé sur seulement deux semaines, et par conséquent certaines fonctionnalités n'ont pas pu être mises en œuvre (à savoir une stratégie supplémentaire pour ne communiquer au serveur que les dix contacts les plus fréquents d'un utilisateur et l'implémentation du pattern *builder* pour la création des utilisateurs).

Cela étant, nous avons pu mettre en application les notions que nous avons vues lors des cours, en parallèle de ce projet, que ce soit de concevoir une application et les questions qui se soulèvent autant en terme technique qu'éthique, l'utilisation de patterns préexistants dans ce processus.

Annexe 1 : Table des figures

Annexe 2 : Diagramme des classes complet