UNIVERSITÉ DE SHERBROOKE
Faculté de génie
Département de génie électrique et de génie informatique

Rapport final

Projet DEGEL #4.0

Conception d'un système informatique distribué
GIF600

Présenté à
M. Bernard Beaulieu
Pr Frédéric Mailhot

Présenté par l'équipe 2 :

BOLX2201	Xavier BOLDUC-MEILLEUR
DOSM2902	Mathieu DOSTIE
FUGE2701	Émile FUGULIN
GIRP2705	Philippe GIRARD
HIPT2501	Théo HIPAUT
LARJ2526	Julien LAROCHELLE
MARD1206	Donavan MARTIN

Sherbrooke – 15 août 2018

Table des matières

1 Introduction

Toute université possède diverses plateformes destinées à connecter le corps professoral à la communauté étudiante. Ces plateformes rassemblent par exemple l'horaire, les résultats académiques, les travaux, etc. Au sein du département de génie électrique et de génie informatique de l'Université de Sherbrooke, la plateforme *Horarius* sert à afficher un horaire individuel aux étudiants, lequel est administré par les coordonnateurs du programme. Sur un fureteur d'ordinateur, l'horaire s'affiche sous un format lisible et clair ; sur mobile, le même contenu peine à s'afficher sur un écran rikiki. Qui plus est, aucune notification n'est envoyée en cas de changement d'horaire, et il faut entrer ses identifiants à chaque connexion, ce qui constitue un irritant supplémentaire.

Dans le cadre du cours de conception d'un système distribué, l'équipe a décidé de s'attaquer à ces trois problématiques en créant une application mobile qui afficherait l'horaire individuel dans un format plus adapté à cette taille d'écran, qui enverrait une notification lorsqu'un changement à l'horaire survient, et qui éviterait à l'utilisateur d'avoir à se reconnecter pour la durée d'un trimestre entier. Au moment d'écrire ces lignes, l'application est fonctionnelle, disponible sur le Play Store et en attente d'approbation par les évaluateurs de l'App Store.

Ce rapport présente la démarche entreprise pour réaliser le projet de conception dans le temps alloué. On y décrit les outils de gestion utilisés, on y explique les décisions prises quant aux technologies de développement possibles, on y décrit les méthodes de test appliquées tout au long du projet, et enfin, on porte un regard post-mortem sur le travail réalisé. Des recommandations s'ensuivent pour guider toute envie par un tiers d'amener l'application plus loin.

2 Gestion

2.1 Sommaire de l'avancement

Au début du projet, l'équipe a décidé de faire son suivi de gestion sur Gitlab. C'était un nouvel outil de gestion pour tous les membres de l'équipe. Avec la version d'essai de 30 jours, le logiciel répondait aux besoins de l'équipe. Malheureusement, lorsque la période d'essai fut terminée, la version gratuite de Gitlab n'offrait aucun outil de gestion intéressant. Ainsi, l'équipe décida d'utiliser le logiciel de gestion Jira. Ce dernier était très bien fait pour suivre la gestion et l'avancement du projet. Le changement d'un outil de gestion vers un autre s'est très bien fait, et n'a aucunement affecté le bon déroulement projet.

Pour faire le suivi de l'avancement, l'équipe se rencontrait chaque jeudi pour connaître l'avancement des tâches du sprint courant. En effet, chaque membre parlait de l'avancement de ses tâches en cours et des problèmes qu'il a rencontrés. Ainsi, l'équipe était en mesure de transférer des ressources ou des tâches si un membre avait trop de tâches. Lorsqu'un jeudi tombait en fin de sprint, l'équipe se décidait des tâches présentes dans le prochain sprint et discutait de ce qui avait bien ou mal été lors du dernier sprint.

Chaque grande section du projet avec un responsable pour suivre l'avancement des tâches. De cette manière, chaque section avait un responsable qui s'assurait qu'à long terme, tout serait prêt pour les différentes remises. Son rôle n'était pas de faire toute la section, mais plutôt de s'assurer que toutes les ressources nécessaires étaient à la disposition des membres.

2.2 Sommaire des heures travaillées par rapport aux heures planifiées

Étant donné que les membres n'avaient pas le même niveau de compétence, l'équipe a décidé de faire le suivant de l'avancement en points et non en heures. De cette manière, le nombre d'heures travaillées n'était pas une mesure prioritaire pour l'équipe.

Chaque membre avait environ le même nombre de points à chaque sprint, ce qui veut dire que l'avancement du projet dépendait davantage d'une complexité que d'un nombre d'heures travaillées. Ce choix a été fait pour pallier les différences de niveaux techniques au début de la session. De plus, étant donné qu'un membre avait déjà fait ce genre de projet, évaluer en termes de complexité était plus facile qu'en termes de durée. Nos évaluations s'avéraient ainsi plus précises.

Néanmoins, les points étaient reliés à une intervalle d'heures selon un barème (tableau 2-1) établi par l'équipe afin d'avoir une idée générale du temps de travail effectué. Généralement un sprint contenait 50 à 60 points, représentant 100 à 150 heures de travail.

TABLEAU 2-1 – Barème de gestion des points

Nombre de points	Durée (heures)
1	0,5 - 1
2	3 - 4
3	5 - 8
4	9 - 12
5	13+

2.3 Sommaire des revues de code et de design

Les revues de code et de design ont été faites au fur et à mesure qu'elles devaient avoir lieu. En effet, notre outil de gestion Gitlab nous a permis de suivre l'avancement du projet en temps réel. Lorsqu'un membre de l'équipe voulait joindre sa branche dans la branche principale, son code devait être passé en revue et être approuvé par au moins un autre membre de l'équipe. Lorsqu'un membre faisait une revue de code ou de design, ce dernier pouvait facilement poser des questions à l'aide de notre outil de gestion de code. De cette manière, l'équipe arrivait à avoir la meilleure pratique méthodique en fonction de l'expérience de tous les membres.

Les revues de design ont surtout eu lieu au début de la session. Une fois le standard de code établi, nous utilisions toujours ce dernier pour ajouter une nouvelle fonctionnalité au projet. Aussi, les revues de code étaient plus facilement acceptées à la fin de projet, car tous utilisaient très bien les standards de l'équipe.

2.4 Suivi des bugfixes

La gestion des bugs a été très efficace tout au long du projet. Lorsqu'un bug était trouvé, un membre de l'équipe était assigné à la tâche de régler le bug. La personne assignée était très souvent la personne qui avait écrit le code contenant le bug. Ainsi, elle était déjà au courant du code et de la logique implémentée. La personne responsable du bug avait comme devoir de s'assurer que le bug était réglé le plus rapidement possible et de tester que le problème avait été résolu. Aussi, dès le début, l'équipe a établi un standard établissant que chaque bug devait avoir un test unitaire pour s'assurer que le bug ne revienne pas dans le futur.

2.5 Sommaire des objectifs du projet

Pour le projet de S6, l'équipe avait pour mission de créer une application mobile afin de permettre aux étudiants d'accéder aux services du site web GEL via un téléphone mobile. Ce projet visait à apporter de la mobilité et de l'accessibilité à l'utilisateur. L'application est un premier pas permettant au département de génie électrique et de génie informatique de s'adapter aux nouvelles technologies tout en réduisant le taux d'absence ou de retard en cas de changement d'horaire.

Plusieurs objectifs étaient visés en ce qui concerne ce projet. Tout d'abord, l'utilisateur devait avoir une application mobile lui permettant de se connecter qu'une seule fois par session et d'accéder à la dernière version de son horaire. Celui-ci devait aussi avoir accès à un service de notifications avec plusieurs options afin d'être au courant de tout changement dans l'horaire.

Un second objectif était d'offrir le même service pour d'autres services. Si le temps nous le permettait, l'équipe devait offrir un service identique s'appliquant aux messages écrits par les professeurs sur la page d'accueil et pour les notes. Cet objectif a été partiellement abandonné au profit d'une intégration à l'équipe de notifications.

2.6 Extrants, indicateurs d'achèvement atteints et processus de vérification

Le premier extrant de la réussite du projet était la connexion prolongée de l'utilisateur. Le système actuel comprend une authentification sur le serveur CAS de l'Université de Sherbrooke, tel que requis. Puisque l'utilisateur est en mesure de se connecter qu'une seul fois par session et d'utiliser l'application sans perdre son accès, ce service est donc considéré réussi.

Le deuxième extrant était l'accès à l'horaire et, éventuellement, aux notes et aux plus récents messages des professeurs. Actuellement, on peut utiliser l'application mobile dans le but de consulter son horaire. Le système modulaire mis en place permet une intégration éventuelle des autres fonctionnalités. Ce deuxième extrant est donc aussi considéré réussi.

Finalement, la réception rapide de notifications signalant tout changement dans les différents éléments fournis par l'application était un extrant critique pour le succès du projet. Celui-ci est aussi un succès puisque l'utilisateur peut recevoir des notifications sur son téléphone dans un délai maximal de 5 minutes après un changement à l'horaire et lorsque des services externes publient des notifications par l'équipe des notifications.

3 Contenu technique

3.1 Cas d'utilisation

Le premier cas d'usage (figure 3-1) consiste en une application d'horaire spécialement conçue pour les étudiants en génie de l'Université de Sherbrooke. La simplicité, la mobilité et l'accessibilité de la gestion d'horaire en sont les caractéristiques principales. En effet, l'application mobile permet à l'étudiant de visualiser son horaire et de recevoir des notifications en cas de changement d'horaire. L'application propose plusieurs paramètres pour personnaliser la réception de notifications

Le deuxième cas d'usage (figure 3-2) consiste en une plateforme de communication sociale entre les étudiants et les professeurs. Présentement, il n'y a que le changement d'horaire qui est lié à cette forme de communication, mais l'infrastructure de l'application permet l'intégration de notifications tierces. À cet effet, la communication d'éléments jugés importants par l'équipe professorale peut être effectuée via le centre de notifications de l'application dans le but de divulguer cette information aux étudiants concernés. On distingue trois types d'utilisateurs :

Un(e) étudiant(e) de l'université de Sherbrooke Il s'agit des utilisateurs les plus nombreux. On cherchera à maximiser la transparence de l'application pour ces utilisateurs afin qu'ils puissent se connecter et se déconnecter, consulter les horaires de cours, recevoir les notifications et modifier les paramètres associées à ces mêmes notifications sans obtenir de détails sur le fonctionnement interne du système.

Un administrateur du système Il devra s'authentifier, aura accès à toutes les permissions des utilisateurs et sera en mesure d'envoyer des notifications à l'application.

L'équipe de notifications Le système se voulant être distribué et modulaire, un type d'utilisateur est réservé pour les autre équipes de projet qui sont susceptibles de gérer des notifications et de les envoyer à notre application. Ce type d'utilisateur particulier devra s'authentifier comme les autres et n'aura accès qu'à une partie restreinte du système, laquelle lui accorde la permission d'envoyer des notifications.

3.2 Design patterns utilisés

La structure du code de ce projet suit plusieurs *design patterns* différents. Pour ce qui est du système dans son ensemble, le *pattern Model View Controller* a été appliqué : l'application mobile fait office de vue en présentant l'information dans son interface, la base de donnée est le modèle qui contient les données et communique à l'aide d'entités avec le serveur. Le backend fait le lien entre la vue et le modèle en tant que contrôleur à l'aide de *data objects sérialisés* en JSON.

Concernant le backend, l'utilisation du *framework* Spring et son fonctionnement par annotations nous obligent à utiliser l'*injection de dépendances* avec les balises `@AutoWired` et le *Singleton* (`@Bean`), car ce *framework* va de pair avec l'injection de dépendances.

Enfin, afin de permettre un filtrage des modifications d'horaire et de ne garder que des notifications d'événements à venir, nous avons mis en place un patron de spécifications (*criteria pattern*). Ce patron se veut une chaîne de critères dans laquelle passent les événements et où seulement la sortie est utilisée pour envoyer les notifications. Les critères implémentés sont notamment l'élimination des événements passés et ceux qui sont plus d'un mois dans le futur.

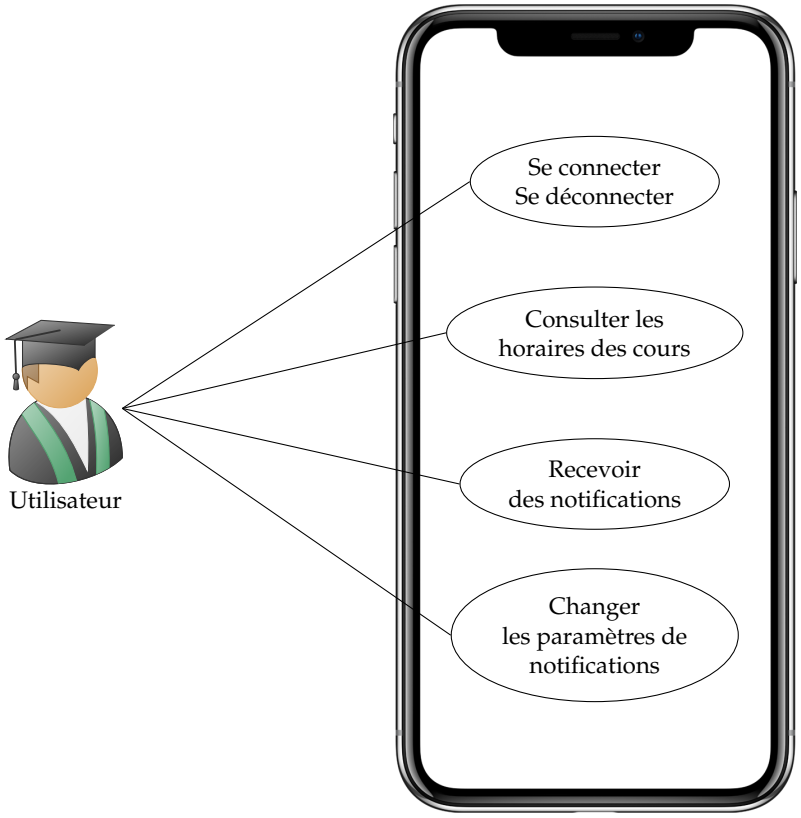

FIGURE 3-1 – Diagramme du premier cas d'usage de l'application

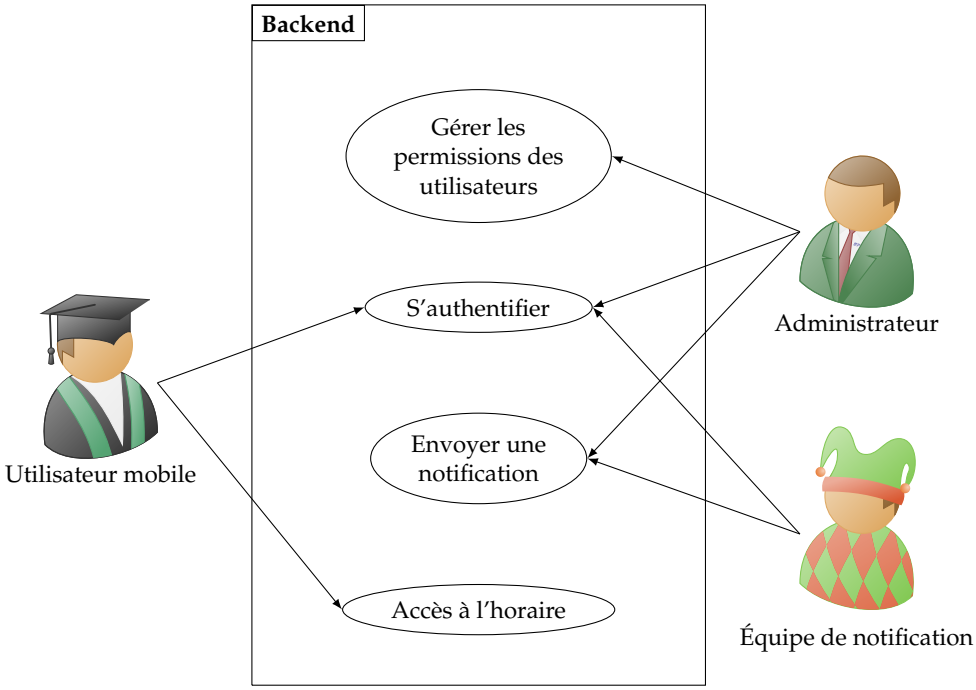

FIGURE 3-2 – Diagramme du second cas d'usage de l'application

3.3 Architecture du système

L'architecture du système (figure 3-3) est divisée en différentes sections :

Éléments bleus Services internes à l'Université

Éléments rouges Services externes à l'Université

Éléments verts Services fournis par l'équipe DEGEL

Les services fournis par DEGEL sont les suivants :

Authentification Module qui regroupe tout ce qui touche de près ou de loin à l'authentification CAS et OAuth2. Ce module permet aussi de gérer les permissions des utilisateurs.

Horaire Module qui s'occupe des communications avec *Horarius*. Il gère les clés des utilisateurs et vérifie les changements d'horaire.

Notifications mobiles Module qui reçoit les requêtes de l'équipe de notifications, lesquelles proviennent soit du module d'horaire, soit d'autres services externes. Il communique les notification aux utilisateurs par le biais des serveurs de Facebook (Expo).

3.4 Modèle d'architecture logicielle

3.4.1 Backend

Pour le backend, l'architecture choisie est le modèle en couches (figure 3-4) :

Controllers Ils définissent les *endpoints* et les permissions associées à ceux-ci. Ils contiennent peu ou aucune logique d'affaire.

Services C'est dans ceux-ci que la grande majorité de la logique d'affaire se retrouve. Ainsi, les tests se concentrent sur eux.

Repositories C'est la couche ORM de l'application qui permet l'accès aux données persistantes.

Database C'est l'instance qui s'occupe des données, Postgres dans notre cas.

Il y a également deux autres parties secondaires dans le modèle :

Tasks C'est l'ensemble des tâches qui roulent en arrière-plan pour notamment aller chercher les modifications aux horaires.

Clients Ceux-ci permettent de communiquer à d'autres services par HTTP.

Notons que les couches *Services* et *Repositories* communiquent entre elles avec des *Entities* qui sont liées aux tables de la base de données. Ces entités sont transformées dans les services en *Models* (et vice versa). Les services et les contrôleurs communiquent à l'aide de ces modèles.

3.4.2 Mobile

Pour l'architecture mobile (figure 3-5), l'ensemble tourne principalement autour des composantes de la partie vue de notre *pattern* MVC :

Components Parties visuelles réutilisables un peu partout dans l'application.

Screens Les différentes vues qui seront présentées à l'utilisateur.

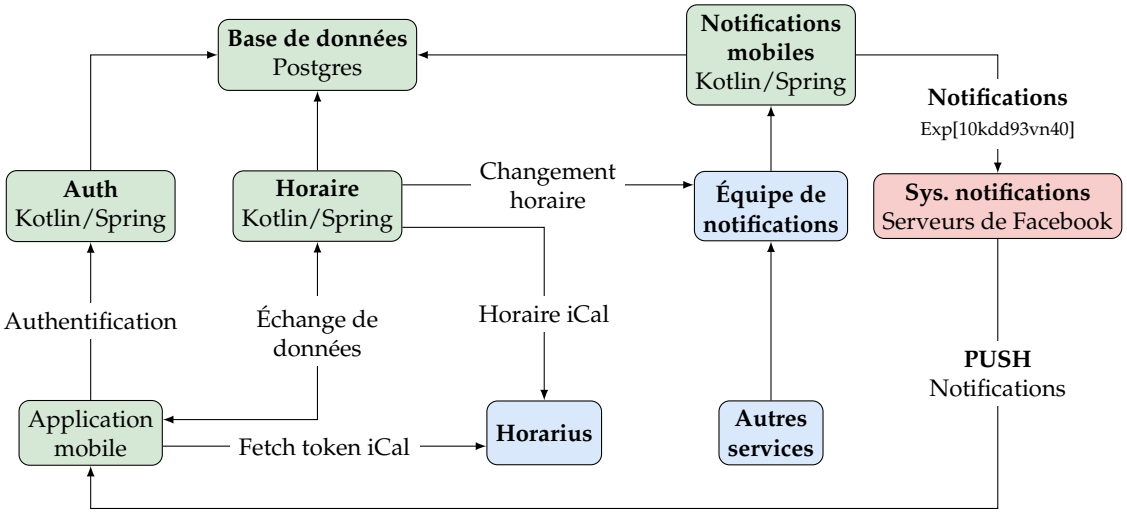

FIGURE 3-3 – Diagramme de l'architecture du système

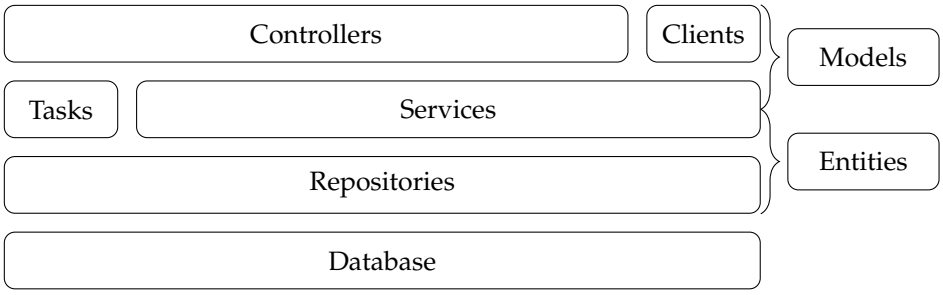

FIGURE 3-4 – Diagramme de l'architecture du backend

```
BL
  __test__
    /* degelClient.test.js
    /* session.test.js
  /* degelClient.js
  /* session.js
  /* storageHelper.js
components
constants
locales
navigation
screens
```

FIGURE 3-5 – Arborescence de l'architecture mobile

Navigation L'agencement des différentes vues et les conditions pour passer de l'une à l'autre.

Il y également trois autres parties :

Constants Utile pour regrouper les constantes de l'application, comme l'URL du serveur.

Locales L'application étant localisée, tous les textes (français et anglais) s'y trouvent.

BL La petite partie de logique d'affaire de l'application, notamment pour communiquer avec le serveur. C'est principalement cette partie qui sera testée.

3.5 Modèle conceptuel de données

Le modèle reste plutôt simple : quatre entités indépendantes (figure 3-6) requises par Spring :

flyway_schema_history Contient l'histoire des migrations SQL effectuées.

oauth_refresh_token Contient les jetons de rafraîchissement des utilisateurs.

oauth_access_token Contient les jetons d'accès des utilisateurs.

oauth_client_details Contient les informations sur les clients OAuth2 (qui sont l'application mobile et l'équipe de notifications).

3.6 Persistance

L'ensemble des entités restantes sont nécessaires à l'application et sont toutes liées à l'utilisateur (figure 3-7).

users Contient l'information de base sur chaque utilisateur (cip, actif, id).

authorities Contient les rôles de chaque utilisateur. Un utilisateur peut avoir plusieurs rôle (ADMIN et USER par exemple).

notification_token Contient les divers tokens Expo de l'utilisateur. Un utilisateur a un token unique par appareil qu'il possède.

calendar Contient les informations du calendrier de l'utilisateur dont notamment sa clé ical et le dernier calendrier qui a été retourné par horarius (pour faire les diff).

settings Contient les préférences de l'utilisateur (une par colonne comme il n'y en pas beaucoup) notamment pour les notifications.

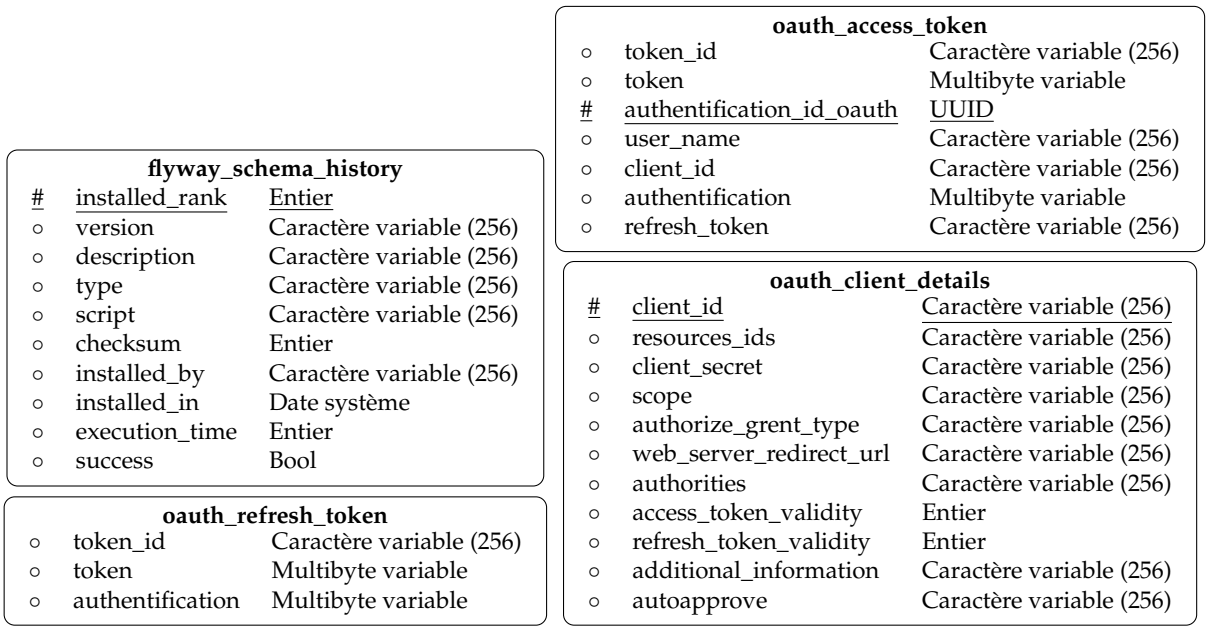

FIGURE 3-6 – Entités indépendantes requises par Spring

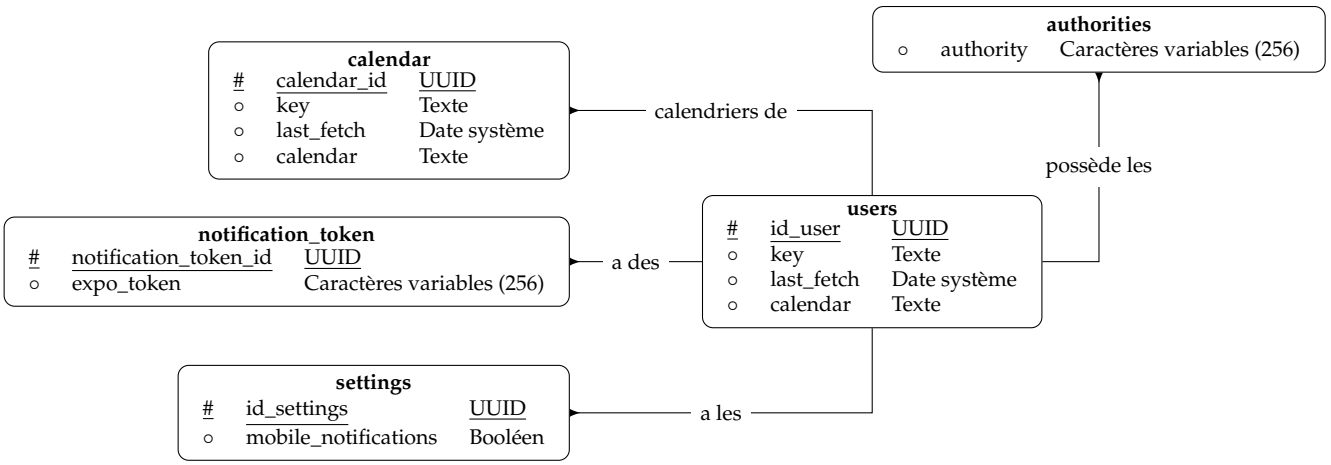

FIGURE 3-7 – Entités liées à l'application et l'utilisateur

3.7 Développement global

3.7.1 Éléments réalisés

Le principal élément réalisé lors du projet de session est l'*utilisation de nombreuses technologies*. Évidemment, cela est fait pour la simplification et l'augmentation de la productivité, mais cela nous a demandé plus de temps au commencement du projet, car la majorité de celles-ci n'était pas connue par l'équipe. Néanmoins, l'effort apporté pour l'apprentissage a eu beaucoup d'avantages, dont l'environnement de développement qui est devenu de plus en plus robuste. En autres, le déploiement automatisé nous a permis de sauver quelques minutes à chaque fois qu'une fusion de branche s'effectuait avec « master ».

L'application en soi propose une interface pour la visualisation de l'horaire et une gestion des notifications. Le problème que résout cette application provient de la réalité actuelle des étudiants en génie à l'Université de Sherbrooke, en ce qui concerne les changements d'horaire et de divulgation de nouvelle information sur l'intranet. En effet, l'horaire affecte directement l'organisation temporelle des étudiants et l'application a pour avantage de divulguer le nouvel horaire le plus tôt possible, soit environ 5 minutes après un changement effectué. Ainsi, l'application est un avancement vers *une meilleure gestion du temps pour les étudiants en génie* et constitue donc un élément dont l'équipe est fière d'avoir réalisé. L'utilisateur désirant réduire les notifications reçues peut désactiver la fonctionnalité dans la page de configuration de l'application. Finalement, l'application a été conçue de manière à améliorer l'accessibilité des personnes atteintes de malvoyance ou de cécité. Chaque mot de l'application est compatible avec la fonctionnalité de l'énonciation de texte intégrée par défaut sur chaque téléphone.

Au niveau du serveur, l'équipe a implémenté un système d'autorisation et de permissions utilisant OAuth2 (figure 3-8). Cela était nécessaire puisque CAS n'offre pas de connexion longue durée sur plusieurs semaines. Cette approche a été très complexe à mettre en œuvre, mais fonctionne très bien. L'utilisateur s'identifie d'abord avec le serveur CAS afin d'avoir une session sur notre serveur, ce qui lui permet de récupérer un *access token* et un *refresh token*. Le premier lui permet d'accéder à certains API (selon ses permissions) et est valide quelques heures. Le deuxième lui permet de récupérer un nouvel *access token* et est valide trois mois. On peut vérifier les permissions à chaque *refresh* de *token*, donc il est très facile de révoquer ou de changer les accès au besoin.

3.7.2 Solutions utilisées

Environnement de développement Pour le développement de l'application, la technologie React-Native avec Expo nous a permis une intégration multi-plateforme pour l'environnement de développement. À vrai dire, ces deux technologies nous ont permis de développer efficacement, puisque le code était compilé sur les serveurs d'Expo plutôt que sur la machine du développeur. D'ailleurs, React-Native permet le développement sur Android et iOS en utilisant un unique code source. Comme si ce n'était pas suffisant, Expo nous permet de compiler l'application automatiquement lorsqu'on sauvegarde l'un des fichiers de l'application. Ainsi, nous avons réalisé un environnement de développement applicatif multi-plateforme avec un déploiement mobile Android et iOS en temps réel, ce qui nous a permis d'optimiser notre temps de travail.

Gitlab Le choix de Gitlab a été principalement motivé par la volonté de l'équipe d'utiliser l'intégration et le déploiement continu dans le projet (figure 3-9). Lors de chaque commit, Gitlab faisait un *build* de l'application et roulait les tests. Cela a permis aux développeurs de savoir très rapidement s'ils avaient brisé quelque chose, ce qui est une raison pour refuser un *merge request*. En supplément, pour le serveur, il y avait deux tâches supplémentaires lors de changements dans le *master*. Premièrement, Gitlab faisait une *release*, qui consiste à construire l'image Docker. Ensuite, Gitlab se connectait en SSH sur la machine de production, arrêtait le service, téléchargeait la nouvelle image et repartait le service. Le tout prenait environ une dizaine de minute et nous avions toujours la dernière version à jour sur le serveur de production sans que l'équipe n'ait à s'en soucier.

Docker L'équipe étant très friande de nouvelles technologies, il a été décidé d'utiliser Docker afin d'encapsuler notre serveur Spring (figure 3-10). Le but est vraiment de créer une boîte noire facile à déployer et qui ne requiert pas de configuration. En plus de cela, l'équipe a utilisé un Docker-Compose afin d'encapsuler l'ensemble des services du backend. Cela crée un réseau virtuel séparé dont le seul point d'entré est un NGINX reverse proxy en HTTPS. Le reste des services peut utiliser des protocoles en clair comme il n'est pas possible d'y accéder de l'extérieur.

Spring Spring est le *framework* web Java le plus utilisé mondialement et en entreprise. Il contient énormément d'intégrations (injection de dépendances, CAS, OAuth2, REST, Hibernate, FlywayDB, OpenFeign, etc.) qui nous ont notamment permis d'implémenter notre authentification et notre gestion complexe des permissions sans avoir à ajouter d'autres dépendances externes. La documentation et le support sont également excellents puisque la communauté est très vaste et active.

Kotlin Kotlin est un langage récent développé par JetBrains qui roule sur la JVM. Sa grande force est qu'il est interopérable à 100 % avec Java, ce qui nous a permis d'utiliser toutes les libraires déjà présentes dans l'écosystème Java. Un peu comme le Java depuis la version 8, Kotlin est un langage qui est influencé par le fonctionnel tout en gardant une forte base orientée-objet. De plus, le langage est *null-safe* et emprunte beaucoup de concepts d'immuabilité, ce qui le rend très sécuritaire. Finalement, le langage possède une syntaxe moderne et est peu verbose, ce qui le rend très agréable à utiliser. Pour toutes ces raisons, l'entièreté du backend a été codée avec Kotlin.

Intégration Horarius Par défaut, l'utilisation d'*Horarius* n'a pas d'API en place pour la détection d'un changement à l'horaire. Nous avons solutionné ce problème à l'aide d'un ajout de fonctionnalité dans le backend qui différencie les horaires à chaque 5 minutes. Plus précisément, chaque étudiant utilise un lien iCal dont l'horaire complet peut être téléchargé. En émettant ce lien iCal, nous avons évalué l'évolution de chaque horaire dans un intervalle de temps. De manière optimale, l'intégration de cette vérification doit être conçue à partir de l'administration des horaires puisque chaque requête de téléchargement au serveur de l'université est lourde et demande beaucoup de bande passante pour une simple application. Nous avons donc une solution pour la preuve de concept, et il suffirait d'un engagement plus concret de la part du service d'administration des locaux de l'université pour optimiser le processus.

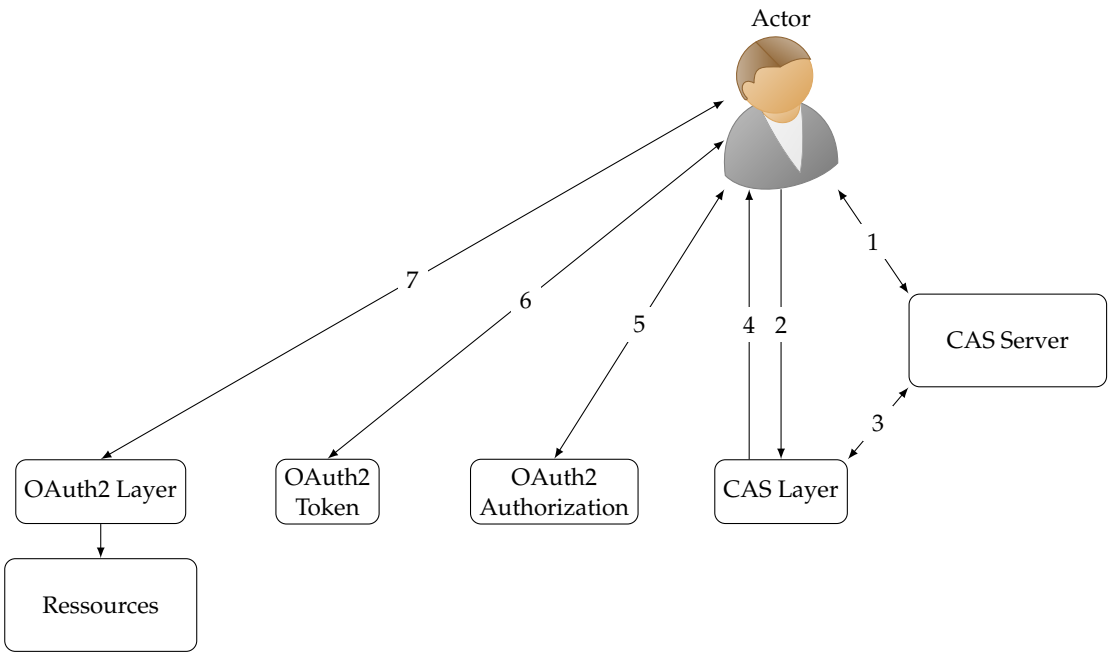

FIGURE 3-8 – Modèle d'autorisation implémenté

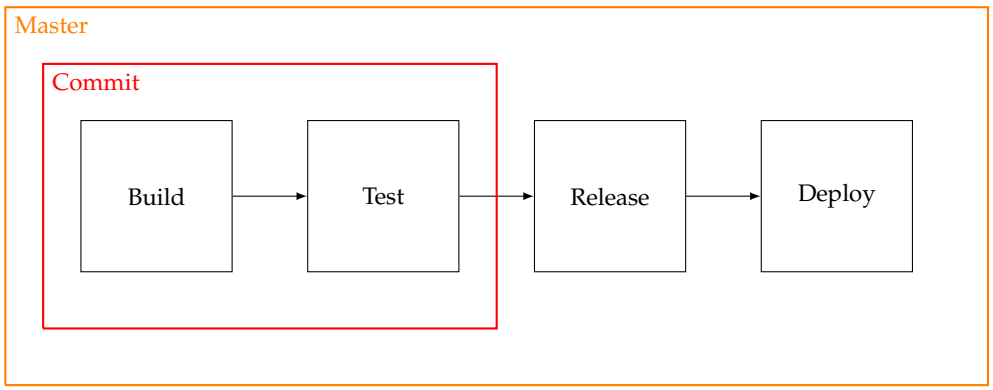

FIGURE 3-9 – Intégration et déploiement continu

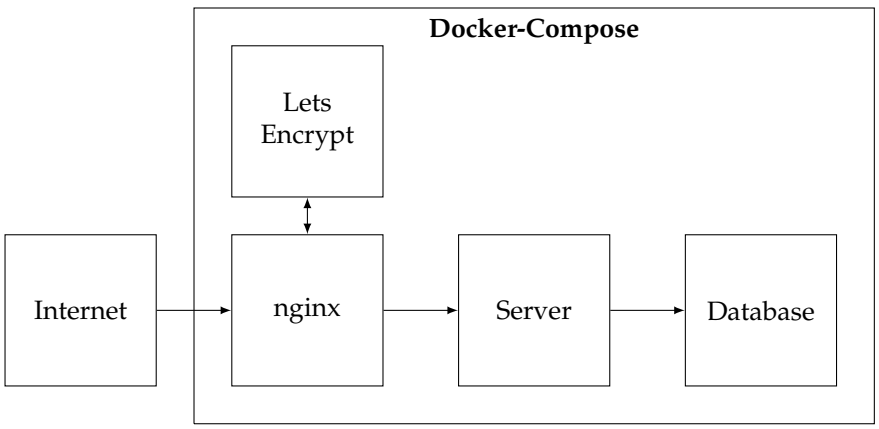

FIGURE 3-10 – Services du backend avec Docker-Compose

Notification Lorsqu'une information pertinente doit être communiquée à l'utilisateur, le système de notifications de Facebook propose un moyen efficace pour l'envoi de notifications. Ces alertes peuvent devenir très complexes à gérer puisque la diversité du type de notifications, la compatibilité multi-plateforme et la permissivité sont des facteurs influençant la qualité du sous-logiciel. Dans l'optique d'une meilleure efficacité, on utilise le système de notifications de Facebook, qui gère très bien les notifications pour plusieurs raisons : multi-plateforme, simple d'utilisation, exemplifié concrètement et pré-configuré pour Expo.

3.8 Problèmes rencontrés

L'un des problèmes rencontrés lors du développement mobile est le manque de documentation pour la technologie Expo. À vrai dire, les exemples d'application mobile Expo similaires sur le web ont été plus bénéfiques pour l'avancement du logiciel que la documentation en soi. D'ailleurs, une étude sur la technologie React-Native estime que le temps passé à écrire le code React-Native est inférieur à 1 %, [1]. React native propose donc un moyen simple d'intégration et de compatibilité multiplateformes, mais ce travail est un casse-tête constant pour les débutants.

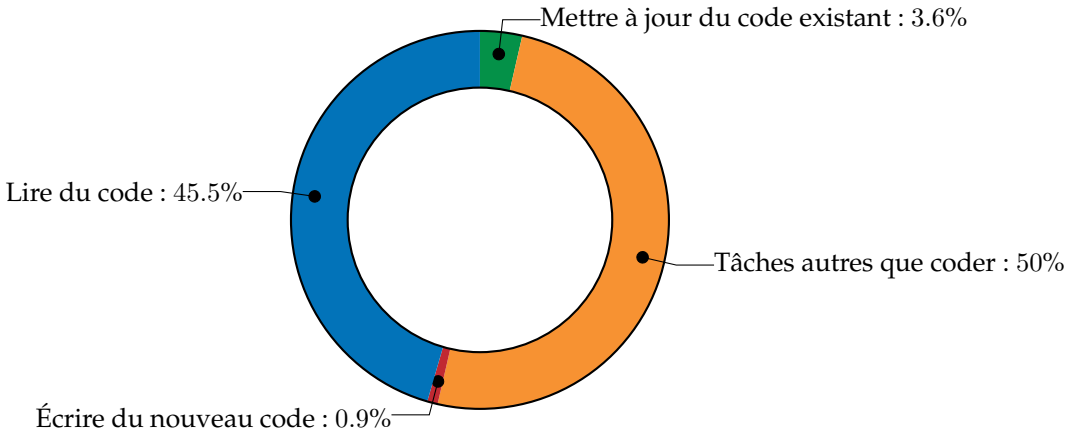

FIGURE 3-11 – Répartition du temps dans un projet React Native

Comme mentionné précédemment, l'absence d'API pour la notification lors d'un changement à l'horaire a été un problème important pendant le développement. Tout d'abord, nous avons tenté de résoudre ce manque avec le groupe professoral, mais en vain. Voyant l'échéancier arriver à grands pas, nous avons décidé de concevoir notre propre logique d'évaluation de changement temporelle d'horaire. Évidemment, cela a généré d'autres problèmes connexes à cet ajout de fonctionnalité. Entre autres, le temps de calcul et la bande passante disponible sont inévitablement des paramètres qui ont été analysés pendant la conception.

Finalement, nous avons rencontré de nombreux problèmes lors de la mise en œuvre de l'authentification. En effet, personne ne fait ce genre d'authentification CAS et OAuth2, car c'est redondant. On choisit généralement l'un ou l'autre et/ou on a le contrôle sur le serveur d'authentification. Il a fallu lire de nombreux tutoriels et expérimenter longtemps pour arriver à une solution fonctionnelle et sécuritaire. Nous avons même écrit un document technique assez poussé détaillant précisément comment CAS et OAuth2 fonctionne en général et dans l'application afin de garder une trace de nos recherches.

4 Description des méthodes de test

4.1 Mobile

4.1.1 Tests unitaires

Une panoplies de tests unitaires automatisés a été effectuée (tableaux[1] 4-2, 4-3, 4-4). Ces tests résident dans les trois fichiers suivants du dossier `BL/__test__` de notre *codebase* :

- `degelClient.test.js`
- `session.test.js`
- `storageHelper.test.js`

Les libraires de tests utilisées sont les suivantes :

Jest expo Effectue les assertions de base, le mock de fonctions.

Jest-fetch-mock Effectue le mock de requêtes HTTP.

Fetch-vcr Enregistre des requêtes HTTP pour les rejouer lors des tests.

Mock-async-storage Simule l'environnement de storage du téléphone.

4.1.2 Tests d'intégration

Les scénarios de tests d'intégrations mobile sont fait et vérifiés manuellement (tableau 4-5).

4.2 Backend

4.2.1 Tests unitaires

Plusieurs *frameworks* tels que JUnit et MockK (équivalent Kotlin de Mockito) ont été utilisés du côté backend pour effectuer une série de tests unitaires (tableau 4-6). Ceux-ci reposent sur l'injection de dépendances par souci de praticité, ce qui permet de ne tester qu'un module à la fois. Comme la logique d'affaire se retrouve surtout dans les *services* et les *helpers*, ce sont eux qui ont été principalement testés. Les tests se trouvent dans le dossier `src/tests/kotlin/ca/usherbrooke.degel/`.

4.2.2 Tests d'intégration

Une interface Swagger a été installée sur notre backend (figure 4-1). Celle-ci permet d'effectuer facilement des tests d'intégration pour tout nos *endpoints*. Swagger est une belle interface graphique qui permet de faire des requêtes HTTP autorisées à des *endpoints* en injectant les bonnes données. Il reste ensuite à observer le résultat pour savoir si la fonctionnalité fonctionne ou non. Les tests d'intégration du backend ont été effectués manuellement.

1. Le titre des tests unitaires est censé décrire ses critères de passation, cependant pour plus de détails concernant les tests des effets de bord (*side effects*), il faut se référer au code source des tests de l'application mobile.

TABLEAU 4-2 – DEGEL Client

Nom	Description
#requestAndSaveAccessTokensWithCode authorized	Requête des access tokens réussite
#requestAndSaveAccessTokensWithCode unauthorized	Requête des access tokens non réussite
#refreshAccessToken	Requête des refresh tokens réussite
#refreshAccessToken unauthorized	Requête des refresh tokens non réussite
#getCurrentUser authorized	Obtenir l'utilisateur courant réussi
#getCurrentUser unauthorized	Obtenir l'utilisateur courant non réussi
#getSettingsStatus authorized	Obtenir l'état des paramètres réussi
#getSettingsStatus unauthorized	Obtenir l'état des paramètres non réussi
#getSettingsStatus unauthorized when id is undefined	Obtenir l'état des paramètres non réussi lorsque le id de l'usager est absent
#setSettingsStatus unauthorized	Requête de changement de paramètres non réussite
#setSettingsStatus unauthorized when id is undefined	Requête de changement de paramètres non réussite lorsque le id de l'usageer est absent
#setSettingsStatus notification mobile = true	Requête de changement de paramètre vers le positif réussite
#setSettingsStatus notification mobile = false	Requête de changement de paramètres vers le négatif réussite
Sends registration token when permission is granted by the user	Envoi des tokens de notification réussi
Sends registration token when permission is already granted	Envoi des tokens de notification réussi même si déjà envoyés
Does not send registration token when permission is denied by the user	Non envoi des tokens de notification lorsque l'usager refuse
registerForPushNotificationsAsync authorized	Enregistrement pour des notifications réussite

TABLEAU 4-3 – Session Tests

Nom	Description
#logIn authorized	Test d'un login autorisé
#logIn unauthorized	Test d'un login non autorisé
#logIn unauthorized expired refresh token	Test d'un login avec un refresh token expiré
#logOut remove all stored values	Test que le logout supprime toutes les informations de l'utilisateur

TABLEAU 4-4 – Storage Helper

Nom	Description
#set save the value when the key does not exists	Sauvegarde d'une clé fonctionne lorsqu'elle n'existe psa
#set override the value when the key exists by removing existing key	Sauvegarde d'une clé existante fonctionne
#set does not save undefined values	Ne sauvegarde pas les valeurs non définies
#set does not save null values	Ne sauvegarde pas les valeurs nulles
#get returns the value	Retour de la valeur fonctionne
#remove remove the value	Suppression de la valeur fonctionne

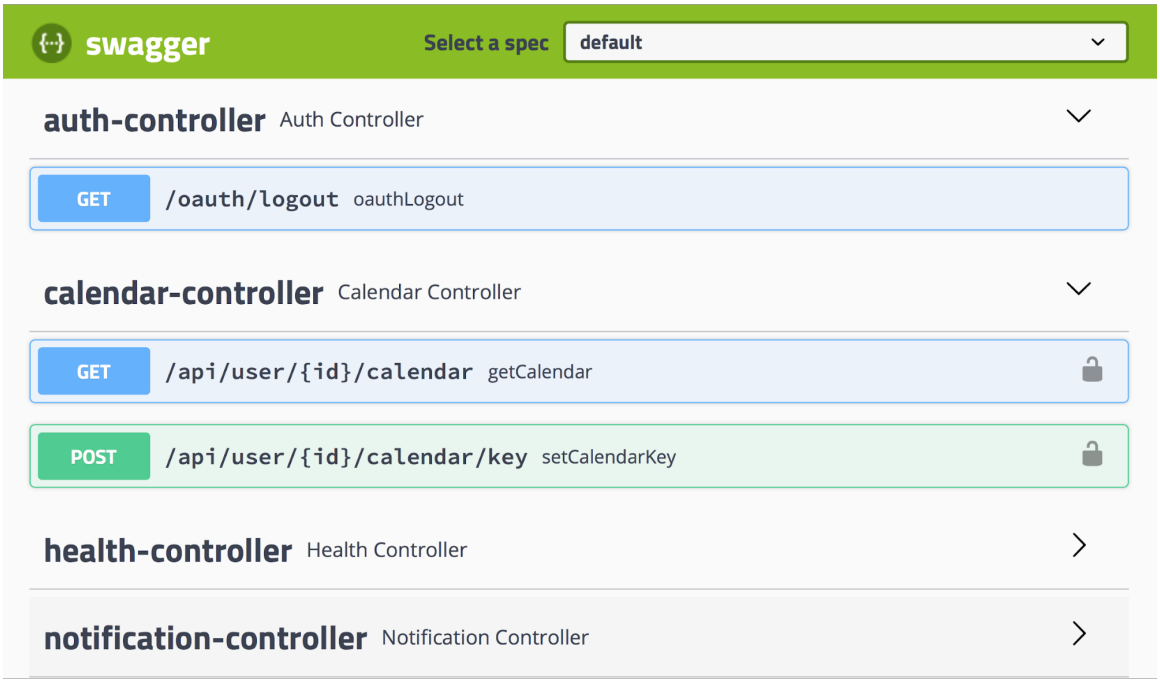

FIGURE 4-1 – Interface swagger du backend

TABLEAU 4-5 – Tests d'intégration sur mobile

Scénario	**Résultat escompté**
Un usager ouvre l'application pour la première fois	Il observe une page qui l'invite à se connecter
L'usager appui sur le bouton de connexion	L'usager est redirigé vers CAS
L'usager entre des informations sur CAS puis se connecter	L'usager est redirigé vers l'application
L'usager appui sur le bouton de retour dans la vue CAS	L'usager est redirigé vers la page précédente
L'usager appui sur le bouton X de la vue CAS	L'usager revient à la page d'accueil
L'usager consulte son horaire une fois connecté	Les événements de son calendrier horarius s'affichent
L'usager se déplace de journée	L'usager peut voir les événements de plusieurs semaines qui sont chargés dynamiquement
L'usager appui sur le bouton de retour à la journée présente	L'usager retourne à la journée présente
L'usager appui sur le bouton des paramètres	L'usager se voit rediriger vers la vue paramètre
L'usager se désinscrit des notifications	L'usager ne reçoit plus de notifications
L'usager s'inscrit aux notifications	L'usager peut recevoir des notifications
L'usager est inscrit aux notifications	L'usager reçoit des notifications lors d'un changement d'horaire
L'usager se déconnecte	Toute les informations sauvegardés de l'usager sont supprimés du téléphone

TABLEAU 4-6 – Tests backend

Nom	Description
[Helper] Test diff calendars	Test du helper de différences de calendriers (ajout, suppression, modification)
[Helper] Test same events	Test du helper de différences d'événements (dates, local, professeur)
[Settings] Get user settings	Test le fetch des préférences de l'utilisateur
[Settings] When no settings get default settings	Test que les préférences par défaut sont retournées pour un nouvel utilisateur
[Settings] If no settings insert them	Test que les nouvelles préférences de l'utilisateur sont enregistrées
[Settings] If settings present update them	Test que les préférences de l'utilisateur sont mises à jour
[Calendrier] If no calendar insert it	Test l'insertion de l'entité calendrier si non présent
[Calendrier] If calendar present update the key	Test la mise à jour de la clé horarius
[Calendrier] If no calendar throw	Test l'exception si le système n'a pas de calendar lorsque demandé
[Calendrier] If empty calendar key throw	Test l'exception si le système à une clé horarius vide
[Calendrier] If invalid key throw	Test l'exception si le système détecte une clé invalide
[Calendrier] Get user calendar	Test l'utilisation normale du client horarius
[Calendrier] If fetching error throw	Test l'exception si horarius retourne une erreur
[Calendrier] Get stored calendar	Test l'extraction du calendrier ical enregistré dans la BD
[Calendrier] Update calendar for user	Test l'update du calendrier ical dans la BD
[Calendrier] Cannot insert invalid calendar key	Test l'exception si l'utilisateur tente d'insérer une clé invalide
[Notifications] When new token add token	Test d'ajout d'un nouveau token Expo
[Notifications] When duplicate token delete old token and then add new token	Test le remplacement d'un Token en cas de duplicata
[Général] Test context load	Test que l'application peut démarrer (long test)

5 Analyse post-mortem

5.1 Technologies

5.1.1 Kotlin

L'utilisation de Kotlin a été un énorme point positif. En effet, la syntaxe simplifiée de celui-ci a accéléré le développement, amélioré la lisibilité du code et facilité la maintenance du code. Par exemple, les *data classes* de Kotlin permettent d'abstraire en quelques lignes de code des centaines de lignes de Java. De plus, Kotlin est en voie de devenir le langage officiel d'Android ; nous étions donc réjouis d'avoir cette occasion d'apprentissage.

5.1.2 Spring

L'utilisation du *framework* Spring n'a pas été appréciée dans le projet, car il y avait trop de magie et de configuration. De plus, ce *framework* était trop lourd. Le temps de démarrage était très élevé malgré que notre projet soit minuscule comparativement à ce qu'un projet avec Spring peut devenir. En rétrospective, l'équipe aurait dû choisir un *framework* Kotlin plus simpliste et plus approprié.

5.1.3 Gitlab en déploiement continu

Gitlab en déploiement continu fut un élément nettement positif dans notre équipe. Il était si simple de déployer, car tout était automatisé. En effet, il suffisait de fusionner une branche à *master* et Gitlab se chargeait de déployer automatiquement sur les VMs de l'université. En contrepartie, il a été extrêmement complexe d'instaurer cette automatisation. Cependant, nous pensons que l'effort en valait la peine à cause de l'utilisation intensive que nous en avons fait.

5.1.4 React Native

React Native fut un choix controversé dans l'équipe. En effet, celui-ci permet de développer rapidement sur des ordinateurs Windows et Mac pour les plateformes Android et iOS avec un seul *codebase* de manière native. Cependant, React Native est une sorte de dette technique, car éventuellement nous devrons reprogrammer certaines parties de notre application à l'aide des langages natifs si nous voulons atteindre de meilleures performances et plus de personnalisation. En définitive, dans le cadre du projet de session, React Native constituait un avantage net, car notre objectif était de livrer un produit fini avec le plus de fonctionnalités possible en 15 semaines. React Native était le choix parfait pour atteindre les objectifs que nous nous étions fixés.

5.1.5 Expo

Expo est un outil gratuit et facile d'utilisation qui nous a permis d'accélérer le développement de plusieurs fonctionnalités telles que les notifications. De plus, Expo nous permet de compiler les applications iOS et Android à l'aide d'une seule commande sur Windows ou Mac sans avoir à installer des *toolchains* sur les machines de développement. Expo fut très apprécié par l'équipe et serait à réutiliser si nous refaisions un projet similaire.

5.2 Gestion

Au début du projet nous utilisions Gitlab pour faire la gestion. Cependant, après 30 jours, la version d'essai a expiré et nous nous sommes aperçus bien vite que les fonctionnalités payantes étaient plus que nécessaires pour un bonne gestion de projet. Vu le prix démesuré de Gitlab, nous avons exploré les alternatives et notre choix s'est arrêté sur Jira. Celui-ci coûtait 13 $ par mois pour toute l'équipe, et nous avons découvert un logiciel qui était bien meilleur que Gitlab. Notre expérience avec Jira a été si positive que nous serions prêt à le réutiliser pour notre projet de fin de baccalauréat ou à le recommander en entreprise.

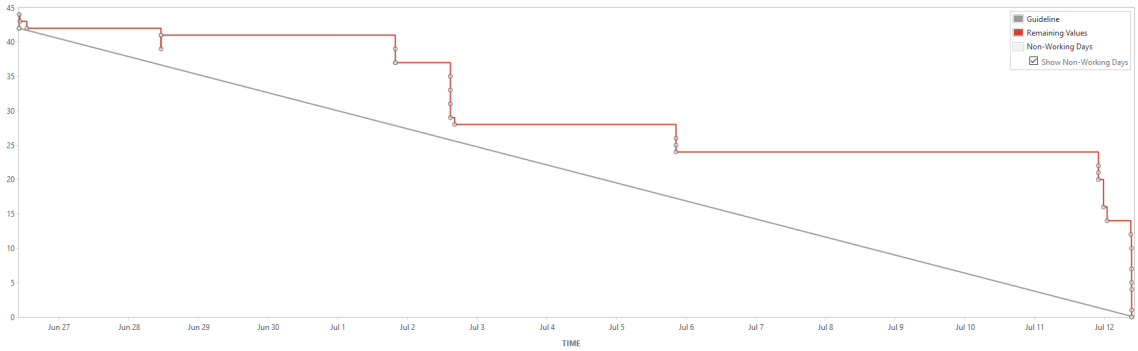

FIGURE 5-1 – Burndown Chart du troisième sprint du projet

Le problème principal rencontré par l'équipe consiste en la stagnation du statut des tâches, ce qui avait des répercussions sur le *burndown chart* (figure 5-1). Ainsi, lors des réunions, nous avions tendance à fermer beaucoup de tâches d'un coup. On peut aussi remarquer que l'équipe avait parfois de la difficulté à faire avancer le projet en même temps que certains APP plus exigeants. Une division en plus petites tâches et un suivi plus serré sauront régler ces problèmes à l'avenir.

5.3 Travail d'équipe et cohésion

Le travail d'équipe s'est bien passé tout au long du projet. Étant donnée la grande différences entre les technologies utilisées dans le projet, nous avons commencé à chacun se spécialiser dans un aspect. Par exemple, certain membres de l'équipe s'occupaient des fonctionnalités mobiles en javascript tandis que d'autres faisaient les fonctionnalités du serveur en Kotlin. De plus, même au sein des équipe mobile et backend il y avait des spécialités. Par exemple, au sein de l'équipe mobile, il y avait une personne qui était spécialiste des détails de l'UI.

Ensuite, l'équipe possédait une très bonne cohésion. En effet, malgré les différents niveaux de compétence, les membres ont travaillé ensemble pour que tout le monde atteigne le même niveau. Il y a donc eu un grand partage de connaissances entre les membres, ce qui s'est avéré essentiel.

Finalement, nous étions très motivés par notre projet, malgré un relâchement en milieu de parcours. En effet, au milieu du projet nous n'avions pas reçu les accès promis en début de session. Cependant, l'équipe a su contourner cet imprévu en trouvant un moyen alternatif d'avoir un produit fini à temps pour la remise. Cet épisode nous a préparés à la vie réelle, car souvent sur le marché du travail, nous allons faire face à des situation où des fournisseurs ne livrent pas ce qui était attendu et où il faut tout de même livrer un produit fonctionnel.

6 Recommandations et conclusions

Suivant la finalité d'un projet, il est nécessaire de réfléchir à certaines recommandations afin de prévoir la continuité du projet. La première recommandation, qui est probablement la plus importante, est la création d'*endpoints* avec les services du département de GEGI. Un exemple serait un *endpoint* de type *callback* permettant au département d'avertir le *backend* de l'équipe lorsqu'un changement est effectué à l'horaire et de lui transmettre l'information concernant ce changement. Ce simple changement permettrait au serveur de l'équipe de répondre à une plus grande quantité d'utilisateurs sans compromettre les performances générales.

Ensuite, une recommandation nécessaire pour la continuité de l'application est de donner l'accès d'une machine virtuelle permanente à l'équipe. Cela donnerait la certitude à l'équipe que leurs services ne seront pas effacés du jour au lendemain lors d'une routine de ménage.

Quant à la troisième recommandation, proposée par l'équipe professorale évaluant la présentation orale finale de l'équipe, consiste en la création d'un groupe technique à l'université qui s'occuperait des outils informatiques créés par les étudiants. Cela permettrait que plusieurs projets bien développés puissent être offerts aux étudiants de GEGI et que les étudiants aient un groupe technique pour découvrir des langages qu'ils ne voient pas durant leur baccalauréat.

Finalement, l'équipe recommande que des employés du département d'informatique prennent un moment pour discuter avec les membres de l'équipe des services disponibles sur GEL qui peuvent se retrouver dans l'application mobile. Par exemple, avertir les étudiants lorsqu'une nouvelle note est disponible ou afficher les messages disponibles sur la page d'accueil du site web.

⋆ ⋆ ⋆

En guise de conclusion, l'équipe estime que le projet est une réussite, notamment par la publication sur l'Android Store et l'App Store de l'application DEGEL. Dorénavant, tous les étudiants (et les quelques étudiantes) du département de génie électrique et de génie informatique de l'Université de Sherbrooke pourront bénéficier du travail réalisé par l'équipe dans le cadre du cours de conception d'un système distribué pour consulter leur horaire sur mobile dans un format agréable, être informés des changements d'horaire de dernière minute et conserver une connexion au CAS durant toute une session.

L'équipe s'est montrée apte à surmonter les défis qui se sont présentés au fil des semaines, que ce soit des outils de gestion trop dispendieux, de la documentation de technologies parcellaire, des accès à *Horarius* non obtenus ou bien des APP de modulation et de transmission RF interminables.

L'application des recommandations énoncées ci-haut ne peut qu'améliorer encore le produit livré et permettre à l'Université de Sherbrooke de faire bonne figure face à ses rivales montréalaises.

Références

[1] M. MALHOTRA, « React native vs flutter : Which is more startup friendly ? », *HackerNoon*, July 2018.