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

Mémoire d'approbation de projet

DEGEL 0.1

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 – 23 mai 2018

Table des matières

1 Introduction

Le cours de projet de la session 6 du baccalauréat en génie informatique vise la conception d'une application améliorant les services offerts aux étudiants du département. Le projet DEGEL se veut une application mobile comprenant une interface conviviale pour l'horaire individuel, une authentification sécurisée à long terme, des notifications intégrées avec une équipe conjointe et optionnellement, une interface pour les notes d'évaluation et les nouvelles du département.

Ce rapport présente le mémoire d'approbation de projet (MAP) à l'équipe professorale en charge du cours de projet. En ordre, on y décrit le cadre logique du projet, les stratégies de gestion, les stratégies de développement, le contenu technique ainsi que les méthodes de test en place.

2 Organisation du projet

2.1 Mission et objectifs

Pour le projet de S6, l'équipe a 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 vise à apporter de la mobilité et de l'accessibilité à l'utilisateur. De plus, cette application permettra au département de génie électrique et de génie informatique de s'adapter aux nouvelles technologies tout en réduisant le taux d'absentéisme en cas de changement d'horaire. Ultimement, celle-ci améliorera la communication entre le département et ses étudiants en réduisant les délais et en s'assurant que tous les éléments importants à communiquer seront transmis efficacement.

Plusieurs objectifs sont visés en ce qui concerne ce projet. Tout d'abord, l'utilisateur utilisera une application mobile lui permettant de se connecter qu'une seule fois par trimestre et aura accès à la dernière version de son horaire. Celui-ci aura aussi accès à un service de notifications avec plusieurs options afin d'être au courant de tout changement dans l'horaire. L'utilisateur configurera ses préférences pour recevoir ses notifications. Il pourra être notifié par l'application mobile, par courriel et/ou par messagerie texte. Un second objectif sera d'offrir le même service pour la grille de notes d'évaluation. Advenant un déroulement rapide du projet, l'équipe offrira un service identique s'appliquant aux messages écrits par les professeurs.

2.2 Intrants

Plusieurs intrants sont prévus pour l'application mobile. Il y a bien entendu l'utilisateur de l'application mobile, pour lequel il faudra développer une interface utilisateur simple, efficace et solide qui saura réagir face à tout comportement imprévu. Un autre intrant nécessaire est le serveur CAS utilisé par l'université de Sherbrooke, lequel permet à un étudiant de s'authentifier. Ensuite, comme l'équipe souhaite offrir à l'utilisateur un accès à son horaire le plus récent, elle s'est entendue avec M. Bernard Beaulieu pour avoir la base de données incluant les horaires des étudiants comme intrant à son projet. Finalement, une autre équipe a décidé de concevoir un service de notifications généralisées. Alors, notre équipe a décidé d'utiliser leur service comme intrant au projet. Toutefois, le succès de notre projet dépendra pas directement de la réussite de leur service puisqu'une gestion des notifications sera présente directement dans l'application.

2.3 Échéancier préliminaire

Le projet se déroulera en 13 semaines, organisées en 3 sprints suivant la méthode agile SCRUM.

2.4 Conditions critiques

La première condition critique est la mise en place des machines virtuelles sur les serveurs de l'UdeS qui hébergeront les différents services de l'application et sa base de données. Sans cette condition, le déploiement de l'application ne pourra pas se faire et le projet n'aura pas lieu d'être.

La seconde condition critique est l'accès au service d'authentification mis en place par l'université afin d'assurer une sécurité équivalente à celle déjà existante. Ensuite, l'accès aux données du site

TABLEAU 2-1 – Échéancier préliminaire

Semaine	Date	Tâches
1	03/05 - 06/05	Organisation préliminaire du projet
2	07/05 - 13/05	Définition du projet - Création du MAP
3	21/05 - 27/05	23/05 : Remise du MAP Début de la première itération
4	28/05 - 03/06	31/05 : Retour sur le MAP
5	04/06 - 10/06	07/06 : Fin de la première itération - Démo - Rapport d'avancement Début de la deuxième itération
6	11/06 - 17/06	
7	18/06 - 24/06	21/06 : Fin de la deuxième itération - Démo - Rapport d'avancement
8	25/06 - 01/07	Début de la troisième itération
9	02/07 - 08/07	
10	09/07 - 15/07	12/07 : Fin de la troisième itération - Démo
11	16/07 - 22/07	19/07 : Test bêta
12	23/07 - 29/07	26/07 : Revue de code
13	30/07 - 05/08	Remise du rapport final - Présentation du projet

Horarius permettra de constater l'efficacité du système de notifications. Ainsi, si le projet sera indépendant des applications desquelles il tirera les données utiles d'un point de vue fonctionnel, son utilité dépendra en partie des services qui tireront partie du système de notifications. *pas clair...*

Enfin, la mise en place d'un serveur d'authentification interne à l'application qui assurera la connexion sur une période de temps plus longue que celle fournie par le CAS sera essentielle dans le sens où l'application devra fournir un service rapide sur toute la durée d'une session d'étude pour ne pas perdre de son intérêt.

2.5 Extrants, indicateurs d'achèvement et vérification

Le premier extrant de la réussite du projet est une connexion prolongée de l'utilisateur. En effet, lorsque l'utilisateur sera en mesure de se connecter qu'une fois par session et d'utiliser l'application sans perdre son accès, ce service sera considéré comme réussi. Bien-sûr, il ne faut pas supprimer l'application mobile ou se déconnecter.

Le deuxième extrant est l'accès à l'horaire et éventuellement, aux notes et aux messages des professeurs les plus récents. L'utilisateur devra pouvoir utiliser l'application mobile sans qu'il n'y ait de différences avec les données sur le site web GEL.

Finalement, la réception de notifications rapides signalant tout changement dans les différents éléments fournis par l'application est un extrant critique pour le succès du projet. L'utilisateur n'aura pas à aller sans cesse sur son téléphone pour être à jour dans les éléments transmis par son département.

2.6 Risques, déclencheurs et plan de contingence

Risque	Déclencheurs	Plan de contingence	Sévérité	Probabilité
Manque de main-d'oeuvre	Un des développeurs a un accident grave, ou vient à manquer pour diverses raisons	Les membres de l'équipe restants se réunissent pour répartir à nouveau les tâches qui étaient attribuées au développeur manquant	Grave	Moyen
Accès impossibles aux services de l'université	Un problème technique majeur survient avec les serveurs de l'UdeS	L'équipe prend les mesures adéquates pour informer les utilisateurs qu'un problème externe à l'application est survenu	Grave	Faible
Problème avec la gestion de version	La license Gitlab utilisée pour la gestion de version est retirée ou n'est plus valide	L'équipe explore les options de récupération possibles avec Gitlab	Grave	Faible
Perte de données dans Gitlab	Défaillance dans les VMs fournies	Toutes les *issues* sont synchronisées avec un logiciel tiers (*backup*)	Moyen	Moyen
Echec aux tests d'intégration	Un des modules de l'application ne peut pas être intégré au reste du projet	L'équipe identifie les raisons de ce problème d'intégration et les développeurs en charge du module recommencent, ou d'autres développeurs se chargent de ce module	Moyen	Moyen
Maîtrise d'une technologie utilisée	Les développeurs ne parviennent pas à maîtriser le *framework* React Native	L'équipe prend la décision de changer de *framework* ou répartit à nouveau les tâches à d'autres développeurs	Moyen	Faible

TABLEAU 2-2 – Concepts de risque par rapport au projet

3 Gestion (stratégies retenues)

3.1 Méthodes de mesure de l'avancement

La gestion de l'avancement du projet est faite à l'aide de la méthodologie SCRUM. Pour ce faire, on utilise le logiciel Gitlab, qui nous permet de faire la gestion des tâches, la gestion des heures et de produire facilement les courbes nécessaires pour faire une gestion efficace de notre projet.

Pour être en mesure de faire une gestion efficace lors de la session, l'équipe doit préalablement diviser le projet en plusieurs tâches et faire une estimation de temps nécessaire pour chacune. Ainsi, l'équipe sait combien d'heures estimées il reste avant chaque revue en fonction du temps déjà effectué.

Définir adéquatement les tâches avant de commencer le projet est un point important pour être en mesure de faire une gestion efficace, mais l'assiduité des membres est nécessaire à la réussite de la gestion. En effet, il est important que les membres entrent leurs heures travaillées dans l'application et qu'ils s'assurent que le projet avance comme prévu. Si l'équipe juge que le projet est en retard et que l'estimation n'était pas bonne, le remaniement des tâches est possible en cours de projet.

3.2 Processus de gestion des décisions

Aucun membre n'a plus de pouvoir qu'un autre dans l'équipe, ainsi les décisions importantes se font lors des réunions d'équipe. Chaque membre travaillant sur cette section doit être minimalement présent. Idéalement, toute l'équipe doit être présente pour prendre la décision.

Lors d'une prise de décision, il doit y avoir au minimum 4 membres de l'équipe. Le vote est à découvert et la résolution doit obtenir 50 % plus un pour être adoptée.

3.3 Organisation de l'information

Chaque tâche doit avoir une description claire et précise pour permettre au membre de savoir en quoi elle consiste. Si jamais la tâche n'est pas claire, il en revient au membre à qui la tâche est assignée de la définir de façon adéquate. En effet, chaque tâche doit avoir une description assez précise pour permettre à n'importe quel membre de la reprendre.

Pour la gestion des documents, comme les rapports et les graphiques on utilise un Google drive commun. On utilise Google Docs pour l'écriture des rapports, ce qui permet à tous les membres de travailler sur le document en même temps.

Pour la gestion du code, on utilise le système de gestion de versions GIT. On garde ainsi une histoire si jamais un pépin arrive. On utilise un Gitlab installé directement sur les machines de l'Université.

3.4 Gestion et organisation de l'équipe

Étant donné qu'il s'agit d'un projet dans un cadre universitaire, aucun membre de l'équipe n'est supérieur à un autre. Chaque membre a la responsabilité de fournir assez de temps et d'effort pour mener le projet à terme dans les temps établis par le cadre professoral.

Toutefois, il y a certaines personnes qui sont responsable de suivre l'avancement du projet et de

s'assurer que les membres de l'équipe n'oublient pas des parties importantes. Premièrement, un membre doit suivre la gestion du projet. Ce dernier a la responsabilité de s'assurer que toutes les membres utilisent Gitlab convenablement. Il doit aussi s'assurer que les différentes courbes reliées à la méthodologie SCRUM soient faites. Deuxièmement, chaque grande section du projet (BD, serveur, HTML, etc.) doit avoir un responsable qui s'assure que la section est faite convenablement (code et tests) et dans les temps.

3.5 Organisation des rencontres

L'équipe prévoit une rencontre hebdomadaire lors des cours de projet. Ainsi, les membres de l'équipe ne sont pas obligés de se déplacer pour la rencontre d'équipe. Toutefois, les membres de l'équipe peuvent organiser une rencontre à d'autres moments dans la semaine s'ils jugent la réunion pertinente. Pour s'assurer que le plus de personnes soient présentes, il doit consulter les autres membres afin de déterminer l'heure et l'endroit.

3.6 Gestion et révision des objectifs

Lors de chaque réunion hebdomadaire, l'équipe doit convenir d'un certain nombre de tâches. Lors de la semaine suivante, chaque membre révèle comment a été sa semaine, s'il a terminé ses tâches et s'il a des bloquants pour la semaine à venir. Ainsi, on est en mesure de bien partager les tâches entre les membres et de s'assurer d'atteindre nos objectifs avant une remise.

Aussi, on doit réserver un temps après chaque remise importante pour faire une rétroaction d'équipe sur le déroulement précédent. Ainsi, on est capable de sortir les points forts et les points faibles et de changer nos méthodes pour obtenir de meilleurs résultats lors des remises subséquentes.

4 Stratégies techniques de développement

4.1 Standard de programmation

Le plus difficile aspect de l'utilisation d'un standard est de ne pas oublier de l'utiliser. Afin d'éviter ce problème, nous avons décidé d'adopter des standards de programmation de base qui sont utilisés par une majorité de personnes et dont le support est bon. Nous allons également utiliser un maximum d'outils pour automatiser cette tâche.

Pour le Kotlin, nous allons utiliser le standard défini par Jetbrains. Ce standard est appliqué par défaut lorsque le plugin de Kotlin est installé dans IntelliJ. Plusieurs éléments s'inspirent des conventions de Java. Le standard est explicité dans la documentation du langage :

`https://kotlinlang.org/docs/reference/coding-conventions.html`

Comme tous les développeurs utiliserons IntelliJ, il sera relativement facile d'appliquer ce standard en utilisant le raccourci `Ctrl+Shift+Alt+L`.

Pour le Javascript, nous allons utiliser le formateur Prettier (`https://prettier.io/`), très utilisé par la communauté et supporté par de nombreux IDE. Il est possible de changer les règles de formatage au besoin, mais nous allons commencer avec les règles de base. Également, le formateur sera automatiquement roulé sur tous les fichiers du projet avant chaque commit (« *pre-commit hook* »). Cela permet de s'assurer que le format est toujours respecté.

4.2 Gestion des revues de code

Tel que mentionné, l'équipe utilisera Gitlab afin de gérer le code du projet. Chaque fonctionnalité sera développée dans sa propre branche. Lorsqu'une fonctionnalité sera terminée et prête à être ajoutée au code principal, le développeur utilisera la fonctionnalité « *Merge Request* » de Gitlab et désignera au moins un autre membre pour effectuer une revue de code. Si les modifications sont approuvées (plusieurs itérations possibles) et que les tests passent, le code sera fusionné au code principal. Dans cette revue, nous allons vérifier tous les aspects du code, de la nomenclature aux algorithmes.

Avec cette manière de procéder, aucun développeur ne peut modifier directement le code principal. On s'assure ainsi qu'aucun code non testé n'est ajouté au code principal. On réduit aussi fortement le risque de briser notre code de production et on maintient toujours un code principal qui est utilisable.

4.3 Gestion des revues de design

Pour chaque fonctionnalité technique importante (que nous nommerons « *epics* »), nous aurons d'abord une réunion de design avec les développeurs concernés afin d'établir et de documenter les lignes directrices qui devront être respectées.

Ensuite, pour chaque fonctionnalité (que nous nommerons « *features* »), le développeur sera responsable du design. Ce design devra être cohérent avec celui défini par l'équipe pour l'*epic* correspondant. Si le design n'est pas explicite ou étrange à première vue, il devra être explicité textuellement dans le *merge request*. Le design sera un élément sujet à la critique dans la revue de

code.

Finalement, nous aurons une revue de design qui sera partie intégrante de la réunion de fin de sprint afin de s'assurer du respect de la direction de design. Le but est que le code reste extensible et soutenable sur une longue période. Chaque déviation devra être documentée afin de garder une trace de l'évolution du design de l'application.

4.4 Gestion des revues de correction de bogues

Pour les *bugfixes*, tout commencera par l'ouverture d'une *issue* contenant notamment des étapes de reproductibilité (si possible). Lorsque le bug sera confirmé, un développeur sera assigné à sa résolution. Le processus ensuite sera très similaire à l'ajout d'une fonctionnalité ordinaire (tests, *merge request*, revue). Lorsque le *fix* sera déployé, la personne ayant ouvert le bug s'assurera du suivi en vérifiant si le bug a bien été corrigé et fermera l'*issue* si tel est le cas.

4.5 Outils de développement utilisés

4.5.1 Code et issues : Gitlab

Nous allons explorer les fonctionnalités afin de s'assurer que l'entièreté de la gestion de projet puisse être faite à l'aide de cet outil, mais nous sommes confiants que c'est le cas.

Cet outil possède aussi les fonctionnalités d'intégration continue et de déploiement continu qui nous permettent de rouler nos tests automatiquement lors de chaque *merge request* et de mettre le programme à jour sur nos serveurs de production.

4.5.2 Développement Kotlin

IDE IntelliJ, car c'est simplement le meilleur IDE afin d'offrir un support Kotlin intégré.

Dépendances et *builds* Gradle, car cet outil est moderne, de plus en plus utilisé en industrie et plus agréable que *maven*.

Tests JUnit (référence pour les tests sur la JVM) et Mockito (référence pour les *mocks* et intégré à JUnit).

4.5.3 Développement Javascript

IDE Au choix, car beaucoup supportent bien Javascript. Une personne devra avoir XCode et Android Studio afin de faire les *builds* finaux.

Tests Jest, car c'est la référence pour les tests en Javascript.

4.6 Autres stratégies

Une autre stratégie de développement pour le projet sera d'utiliser des migrations pour muter automatiquement le schéma de nos bases de données. En effet, ceci aura pour avantage de pouvoir

gérer des versions de notre base de données et de pouvoir revenir dans un état précédent si nécessaire. Cela permet aussi d'associer des changements de schéma à des changement de code dans un *merge request* et de les exécuter au même moment. Cela permet de garder un grand synchronisme entre la base de donnée et le code qui l'utilise.

L'équipe tentera de limiter les changements de la base de données qui brisent le code, tel que la suppression d'une colonne. Pour ne pas perdre de données, des migrations en trois étapes pourront être utilisés. Par exemple, pour la suppression d'une colonne, une première migration ajoute une nouvelle colonne et copie les données de l'ancienne colonne dans la nouvelle, puis une deuxième migration change le code pour utiliser la nouvelle colonne et une dernière migration supprime la vieille colonne.

Pour les diagramme l'équipe utilisera l'outil Draw.io. Cet outil est complet et permet de faire toutes les sortes de diagrammes nécessaires pour le projet.

Finalement, la base de données sera accédée à l'aide d'un *object-relational mapping* (ORM). L'option de l'ORM JOOQ sera évalué. Un ORM est une solution idéale pour accéder une base de données à partir d'un langage orienté objet tel Kotlin/Java. Il faudra faire attention à la disparité objet-relationnel.

5 Contenu technique

5.1 Contenu du « Project Backlog » initial

TABLEAU 5-3 – Project Backlog

ID	Titre	Responsable	Milestone	Durée
38	Release managers should own coordination of CE to EE merge			0
39	Slower rotation of Release managers			0
55	Document exception request process			0
61	Documentation changes for release process			0
68	Remove manual QA checklist steps from the QA template			0
150	On-boarding Mek as release trainee	Mek Stittri		0
169	Communicating errors introduced by a new deployment			0
172	Testplan for GDPR (General Data Protection Regulation) feature		10.8	0
173	Creating RC's and patching GitLab.com			0
175	[Meta] Tooling improvements			0
215	Outline an ideal release process			0

222	2018-05-18 : 10.8.1 exception request for gitlab-org/gitlab-ce!19021	0
226	2018-05-22 : 10.8.1 exception request for gitlab-org/gitlab-ce!19087	0

5.2 Contenu du backlog du 1er sprint

5.3 Diagrammes de séquences

Lorsque l'utilisateur veut utiliser le service, ce dernier valide son identité auprès du système d'authentification, lequel demande et vérifie avec les serveur CAS les informations de connexion de l'utilisateur. Le serveur CAS renvoie un *token* qui est remis au service. Pour faire durer la connexion, le système d'authentification gère le *token* CAS comme connexion permanente et renvoie un *token* AUTH au service. Le service utilise le token AUTH pour vérifier l'identité de l'utilisateur et valider les notifications à émettre. Lorsque le *token* AUTH est valide et qu'une notifications est reçue, la notification est transmise à l'utilisateur. Le diagramme de séquences (figure 5-1) illustre ces étapes.

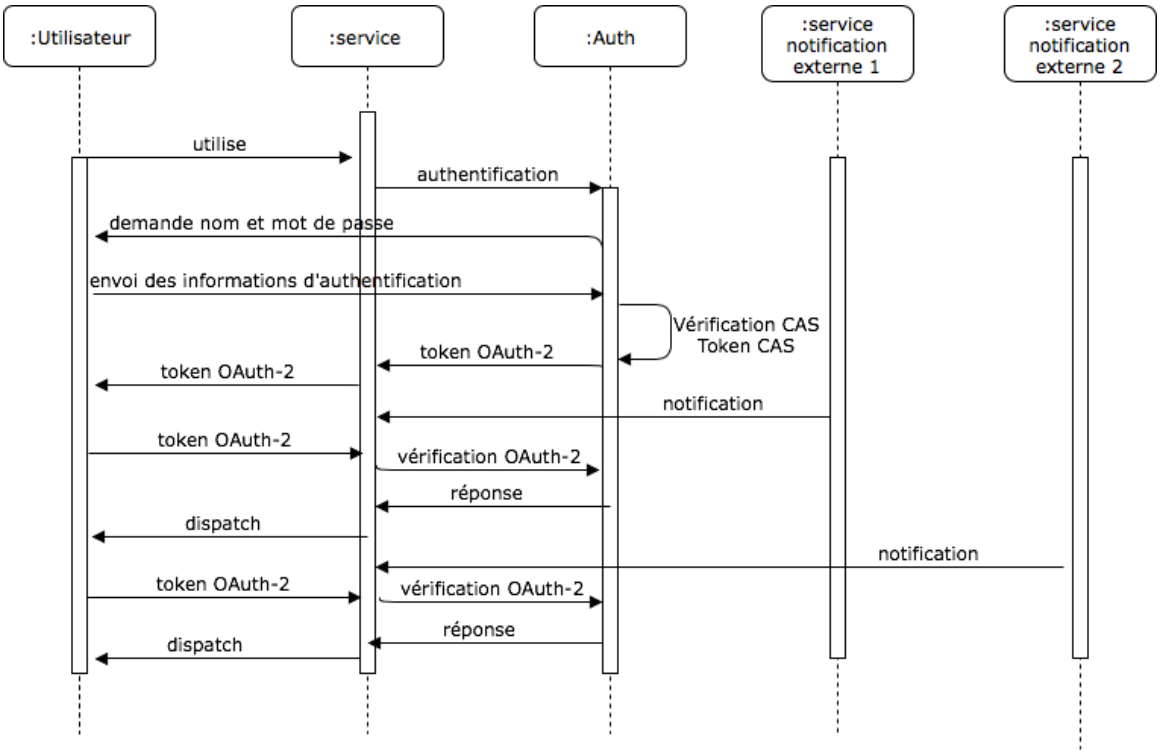

FIGURE 5-1 – Diagramme de séquences par rapport à l'authentification à mémoire long terme

5.4 Architecture du système

L'architecture du système (figure 5-2) se compose en 4 parties distinctes, soient l'application mobile, le service mobile, l'authentification et le micro-service. Tout d'abord, l'application mobile permettra à un utilisateur d'Android et/ou d'iPhone d'utiliser le service mobile à partir d'une application disponible dans le magasin d'application du téléphone. En ce qui concerne le service mobile, un serveur gérera les notifications du micro-service et des autres équipes de notifications à condition que l'utilisateur soit authentifié. Pour s'authentifier, un utilisateur doit s'authentifier sur le serveur de CAS. Lorsque le token est valide et que les permissions le permettent, des micro-services peuvent être utilisés.

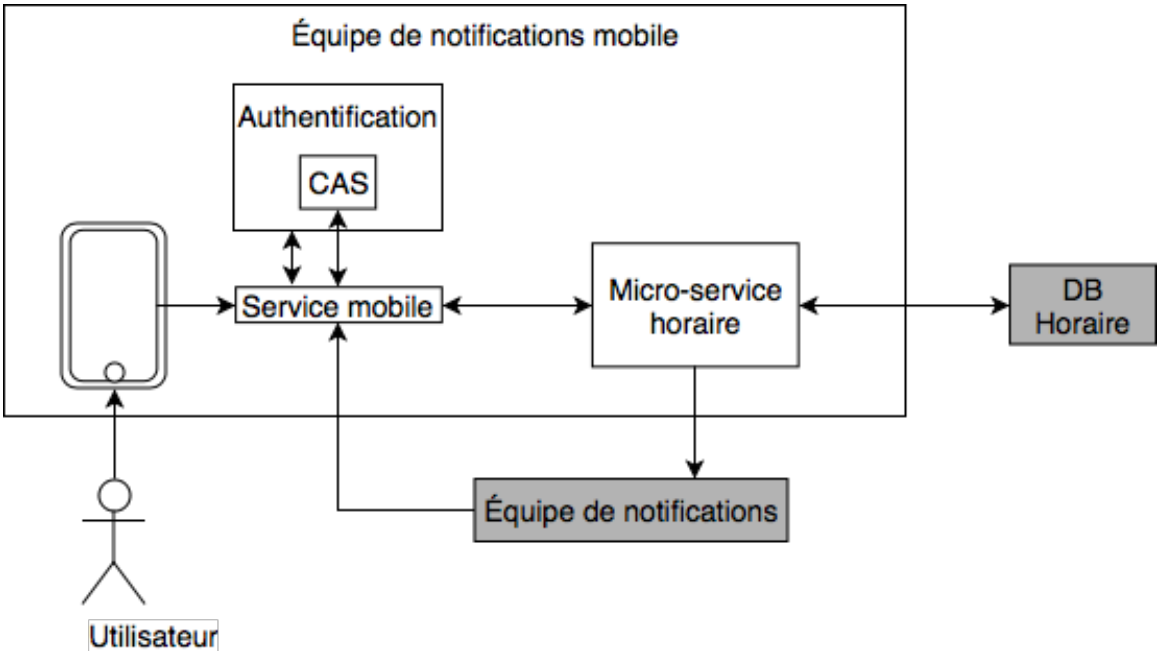

FIGURE 5-2 – Architecture du service mobile de notifications pour l'horaire

5.5 Modèles de conception

Beaucoup de *design patterns* sont abstraits par les *frameworks* qui ont été choisis pour le projet, donc nous nous sommes limités ici à ceux qui seront utilisés explicitement.

Dependency injection Le *framework* Spring repose fortement sur l'injection de dépendance et nous l'utiliserons à son maximum. Il suffit d'ajouter l'annotation `@AutoWired` à un membre pour indiquer à Spring qu'il doit injecter la valeur et le reste se fait automatiquement.

Singleton Dans Spring, ce pattern va de pair avec l'injection de dépendance. En effet, pour injecter la bonne dépendance, Spring recherche une fonction annotée `@Bean` qui retourne un objet du bon type. Par défaut, cet objet sera réutilisé pour toutes les injections et est donc de fait un *singleton*.

MVC Notre architecture fait un fort usage du *design pattern* MVC. En effet, notre vue est l'application mobile qui ne contient pas de logique d'affaire et communique avec les contrôleurs du *backend* avec des modèles précis sérialisés en JSON.

6 Description des méthodes de test

6.1 Tests unitaires

Pour les tests unitaires, l'équipe a prévu de tester les services Kotlin de notre *backend*. Un service est une unité de logique d'entreprise (*Business logic*) avec des intrants et des extrants.

L'équipe ne va pas tester au niveau unitaire les contrôleurs, les requêtes HTTP et l'interface avec la base de donnée. Ceci va être testé dans les tests d'intégrations.

Les tests unitaires seront créés avec le framework JUnit. Leur exécution sera automatisée sur les postes locaux des développeurs et sur nos serveurs de déploiement.

Du côté de l'application mobile, les tests unitaires seront écrits à l'aide de Jest. Nous testerons majoritairement la transformation des résultats des API appelés par l'application mobile.

6.2 Tests d'intégration

Le principal élément à tester au niveau de l'intégration est l'API de notre système. L'équipe devra automatiser ces tests également. Cette tâche est complexe, car il faut intégrer plusieurs systèmes ensemble dans un environnement de tests. Il faut qu'il soit productif d'écrire les tests d'intégrations, sinon aucun développeur ne désirera en écrire et nous aurons une couverture incomplète de tests.

Pour rendre l'écriture des tests d'intégrations productive, l'équipe prévoit utiliser une combinaison de JUnit, de la technique d'injection de dépendances et de *Fixtures*. JUnit permettra de faire des assertions pour déterminer si un test d'intégration passe ou non. L'injection de dépendances permettra de *mocker* certains éléments cachés du système pour limiter les effets collatéraux et bien isoler les fonctionnalités que nous voulons tester. Finalement, les *fixtures* sont un outil pratique gérer l'état de notre base de données et comparer au début et à la fin des tests si l'état de la base de données a changé comme prévu.

Ces tests seront automatisés sur les postes des développeurs et sur les serveurs de déploiement.

Pour l'application mobile, Jest sera également utilisé pour les tests d'intégration. Ceux-ci vont consister principalement en des tests de UI pour s'assurer que les vues sont générées correctement.

6.3 Processus de test prévus

Pour exécuter les tests sur les postes locaux des développeurs, ils devront lancer une commande qui exécutera l'ensemble ou un sous-ensemble des tests. Lorsqu'un développeur écrit une nouvelle fonctionnalité, il doit écrire un ou des tests qui s'assurent de la présence et du bon fonctionnement de cette fonctionnalité.

Ensuite, le développeur peut envoyer son code en révision via un *merge request* (MR). Sur le MR, la totalité des tests sera exécutée, puis d'autres membres de l'équipe pourront commenter les changements apportés au code. Les tests seront donc exécutés à très haute fréquence, puisque pour chaque changement au code principal (branche *master*) il y aura exécution de la totalité des tests.

Pour l'application mobile, le processus sera plus manuel. Les tests seront exécutés avec une commande sur le poste du développeur. Puis celui-ci va déployer de manuellement l'application sur la plateforme de son choix.

7 Recommandations et conclusions

Le projet de S6 intégrera un système de notifications mobiles à l'aide d'intrants de l'université de Sherbrooke. Puisque ce système sera compatible avec les notifications envoyées par d'autres équipes, nous devons nous assurer d'une bonne communication avec les autres équipes. De plus, il sera nécessaire d'établir des protocoles d'échanges standardisés pour garantir la compatibilité entre systèmes.

La gestion de connexion prolongée, l'accessibilité aux services et la rapidité de réception de notifications sont les principaux extrants du projet. Il va sans doute que les conditions critiques en lien avec le projet devront être surveillées attentivement, le projet dépendant en partie de la collaboration de personnel de l'université.