Le génie pour l'industrie

RAPPORT D'ACHÈVEMENT
PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE DE MONTRÉAL
DANS LE CADRE DU COURS LOG795/GTI795 - PROJET DE FIN D'ÉTUDES

PROTOGEST 2.0

ÉTUDIANTS

PHILIPPE NGO	NGOP 180393-05
ÉMILE ROBINSON	ROBE 190293-07
ALBERT DEKERMENJIAN	DEKA 191287-00
CARL-HENRI CODIO	CODC 151178-08
HUGO LAPOINTE DI GIACOMO	LAPH 081191-04

DÉPARTEMENT DE GÉNIE LOGICIEL ET DES TI

PRÉSENTÉ AU
PROFESSEUR ALAIN APRIL
SUPERVISEUR DU PFE

MONTRÉAL, 20 DÉCEMBRE 2018
SESSION D'AUTOMNE 2018

REMERCIEMENTS

L'équipe de *Protogest 2.0* tient à remercier toutes les personnes qui ont contribué au succès de ce projet et qui ont aidé lors de la rédaction de ce rapport.

Tout d'abord, l'équipe adresse ses remerciements au superviseur du projet de fin d'études (PFE), Dr. Alain April, Professeur à l'École de Technologie Supérieure (ÉTS) de Montréal. Son écoute et ses conseils ont beaucoup aidé dans le développement et la gestion de ce projet, notamment pour cibler nos objectifs.

De plus, l'équipe veut souligner le support de Walid Bezzaoui, étudiant diplômé de l'ÉTS de Montréal, pour avoir facilité la continuité de ce projet. Ses conseils et sa collaboration ont permis une meilleure symbiose dans le travail des différentes équipes qui ont travaillé sur le projet au fil de ses phases.

Également, l'équipe remercie l'aide de Mathieu Dupuis, étudiant diplômé de l'ÉTS de Montréal, pour son aide à la compréhension des différentes technologies Amazon Web Services (AWS). Ses explications et ses conseils ont été cruciaux quant à la réussite de ce projet.

Enfin, un merci spécial aux autres personnes qui, dans l'ombre, ont aidé de près ou de loin à la réalisation du projet, à la rédaction et la révision de ce rapport.

Un grand merci.

Philippe Ngo

Émile Robinson

Albert Dekermenjian

Carl-Henri Codio

Hugo Lapointe Di Giacomo

VERSIONS

Date	Description	#
13 décembre 2018	Publication initiale.	1.0

TABLE DES MATIÈRES

DOCUMENTS DE RÉFÉRENCE

Nom	Version
Rapport technique (Protogest 1.0)	1.0
Guide et gabarits de rédaction de l’ÉTS	4.6

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES ABRÉVIATIONS ET ACRONYMES

Référence	Description
STS	Spring Tools Suite.
IDE	Integrated Development Environment (Environnement de Développement).
Eclipse	Logiciel de développement produit par Eclipse Foundation.
API	Application Programming Interface (Interface de programmation).
REST	Representational State Transfer.
SPA	Single-Page Application.
MVC	Modèle-Vue-Contrôleur.
MIT	Massachusetts Institute of Technology.
GNU GPL	General Public License.
AWS	Amazon Web Services .
DTO	Data Transfer Object.
DAO	Data Access Object.
RDS	Relational Database Service.
EC2	Elastic Compute Cloud.
S3	Simple Storage Service.
OBS	Organization Breakdown Structure.
PBS	Product Breakdown Structure.
WBS	Work Breakdown Structure.
URI	Uniform Resource Identifier.
JSON	JavaScript Object Notation.

INTRODUCTION

La coordination d'un dossier légal entre les parties prenantes d'un procès a toujours été un grand défi pour les avocats. Ainsi, définir dans un court délai une plage horaire qui sera acceptée par l'ensemble des parties en utilisant les moyens de communication conventionnels (téléphone, courriel, calendrier *Outlook*, etc.) présente plusieurs enjeux et n'offre pas de résultats réellement efficaces. Dans sa quête de solution, une firme d'avocats avait présenté lors de la session d'été 2018 une liste de besoins à une première équipe d'étudiants de l'ÉTS, comme projet de fin d'études (PFE). Les analyses sur les besoins ont guidé les développeurs à concevoir et implémenter la première version de l'application ***Protogest***.

Protogest est une application Web regroupant un ensemble de fonctionnalités permettant de faire la gestion et la planification des rencontres pour les avocats, et ainsi faciliter la finalisation du protocole d'entente entre les intervenants juridiques.

Pour assurer la longévité de l'application, une nouvelle version de Protogest sera conçue durant cette session d'automne 2018. En prenant en considération les améliorations proposées et les difficultés rencontrées dans l'ancienne version, les outils originaux seront utilisés lors du développement. Ce rapport présente une analyse des risques encourus, la méthodologie utilisée dans la réalisation du projet, les modifications apportées à la conception de l'application ainsi que les outils et technologies utilisées pour ce faire.

PROJET - Mandat

Le mandat du projet est le développement d'un logiciel Web qui répond aux besoins de structure de rencontre des avocats. Il fut débuté par une autre équipe à la session universitaire précédente. Les objectifs de la phase actuelle du projet, la phase 2, étaient donc de continuer le développement et la conception du logiciel en évaluant les technologies utilisées, en validant le développement actuel et, si nécessaire, en migrant le logiciel vers d'autres technologies.

Améliorations proposées

ID	Description
AP01	Implémenter une communication entre *Front-End* et *Back-End* avec *DTOs*.
AP02	Développer un *Back-End* sans session.
AP03	Migration de la base de données *H2* vers *MySQL*.
AP04	Hébergement des ressources *Front-End* et *Back-End* en ligne.
AP05	Satisfaire les critères d'une application Web *Open source*.

Tableau 01 : Améliorations proposées

PROJET - Équipe

L'équipe a été formée automatiquement à l'aide du système de vote de la plateforme Moodle gérée par l'ÉTS. Chaque étudiant inscrit au PFE devait sélectionner les projets qui l'intéressaient, par ordre de préférence. Par la suite, le système a désigné un projet pour chaque étudiant en fonction des votes et des disponibilités. Tous les membres de l'équipe ont donc été choisis par ce système.

Le rôle de chaque étudiant a été décidé en fonction des connaissances et des expertises de chacun. Le choix des technologies que chaque membre désirait apprendre a également influencé le rôle de chacun.

En fonction de ses bonnes connaissances en Java et de son expérience suffisante dans ce langage de programmation, Hugo Lapointe Di Giacomo a été nommé développeur *Back-End*. Ce choix a été renforcé par son intérêt à apprendre la technologie Spring Boot. Également, suite à un rapide vote lors de la première rencontre, il a été désigné gestionnaire de projet.

Émile Robinson a été nommé opérateur de l'infrastructure, dû à ses bonnes connaissances dans le domaine des technologies de l'information et des bases de données.

Philippe Ngo, Carl-Henri Codio et Albert Dekermenjian ont occupé la fonction de développeur Front-End. Philippe avait déjà travaillé à ce poste et avait donc la plus grande maîtrise de la technologie Angular. Successivement, Carl-Henri avait une expérience de base avec cette technologie et Albert était désireux de l'apprendre.

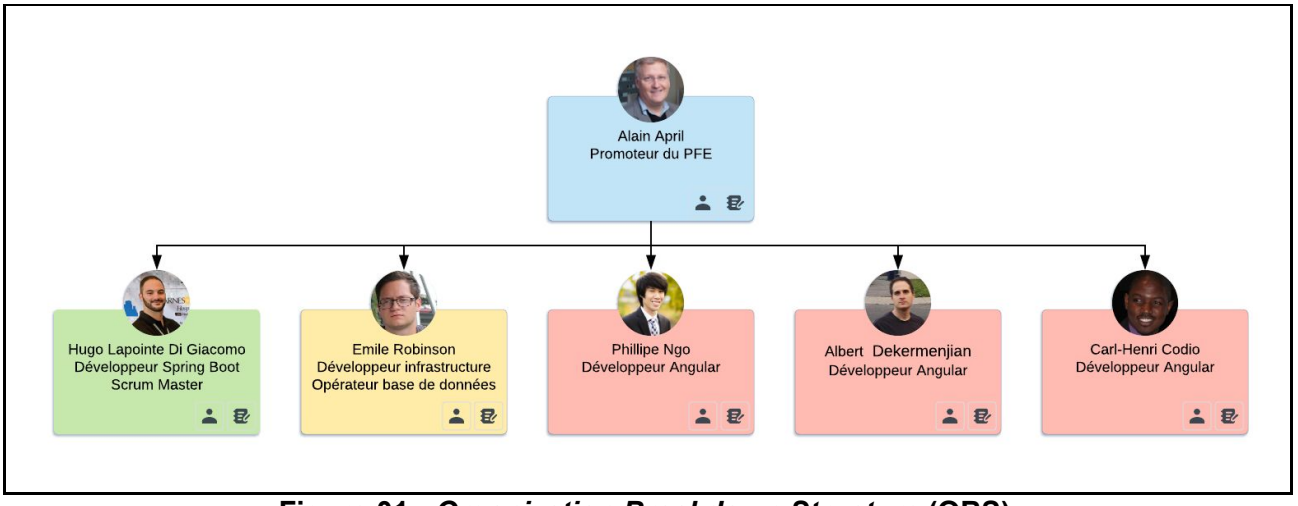


Figure 01 : *Organization Breakdown Structure* (OBS)

Ainsi, suite à la première rencontre de l'équipe, la description et la responsabilité de chaque membre de l'équipe furent déterminées afin que tous soient en accord et ainsi éviter des problèmes de communication au cours du projet.

Nom	Description	Responsabilités
Alain April	- Promoteur du PFE	- Définir les exigences du projet. - Définir les exigences du produit.
Albert Dekermenjian	- Développeur Angular	- Développer le Front-End. - Rédiger la documentation.
Carl-Henri Codio	- Développeur Angular	- Développer le Front-End.. - Rédiger la documentation.
Émile Robinson	- Opérateur de l'infrastructure	- Configurer l'infrastructure AWS. - Administrer la base de données. - Rédiger la documentation.
Philippe Ngo	- Développeur Angular	- Développer le Front-End. - Rédiger la documentation.
Hugo Lapointe Di Giacomo	- Développeur Spring Boot - Gestionnaire de projet	- Développer le Back-End. - Gérer les activités de l'équipe. - Superviser l'avancement du projet.

Tableau 02: Description et responsabilités des membres de l'équipe

PROJET - Livrables

Tous les livrables de projets listés dans le tableau ci-dessous étaient obligatoires dans le cadre du projet de fin d'études. Le rapport de projet se distingue des autres livrables par le fait qu'il s'agit d'un texte à être lu par un individu et qui ne sert pas à l'exécution d'un programme informatique.

Subséquemment vient le code source de l'application qui se divise entre la partie Front-End et la partie Back-End. Il a été décidé que le *Front-End* et le *Back-End* aient chacun un dépôt *Github* dédié au lieu d'un même et seul dépôt regroupant les deux.

Cette décision a été prise, car la séparation des deux entités faciliterait l'automatisation du déploiement au niveau de l'infonuagique. La très grande majorité des entreprises font la même division, ce qui fait en sorte que cette façon de faire est devenue un standard. L'équipe a décidé de suivre ce standard.

Les fichiers de configuration de l'infrastructure constituent un autre livrable essentiel qui permet la configuration de l'hébergement en nuage et la configuration de la base de données.

La gestion des livrables est faite selon le rôle de chacun: trois personnes pour le Front-End, une personne pour le Back-End et une personne pour les configurations de l'infrastructure.

La qualité du Front-End et du Back-End est garantie par des tests exécutés par les membres de l'équipe pour vérifier si toutes les exigences fonctionnelles sont satisfaites. Des tests de validation ont aussi été effectués séparément du côté Front-End pour vérifier la prise en charge de toutes les entrées possibles par l'utilisateur.

La revue de code du Front-End a été effectuée par Philippe, qui avait déjà une expertise avec Angular, la technologie utilisée par ce segment. Cette opération assure la qualité de l'implémentation, toujours du côté Front-End.

La qualité des fichiers de configuration de l'infrastructure est quant à elle constamment vérifiée par les divers tests de connexion au serveur.

Nom	Description
Rapport de projet	Ce document contient la description du projet, des tâches qui ont été réalisées et la présentation des technologies.
Code source Front-End	Le code source *Angular*.
Code source Back-End	Le code source *Spring Boot*.
Configurations de l'infrastructure	Les fichiers de configurations relatifs à l'installation de l'infrastructure *AWS*.

Tableau 03: Description des livrables du projet

PROJET - Analyse des risques

Tous les risques reliés au projet sont indiqués dans le tableau ci-dessous. Les divers risques mentionnés ont été déterminés en fonction des expériences passées des membres de l'équipe.

Plusieurs moyens de contrôle ont été identifiés et mis en place afin de minimiser l'impact qu'engendrerait la matérialisation de l'un des risques identifiés. L'utilisation de l'application de gestion de projet Trello pour l'identification, l'assignation et le suivi des livrables a permis de bien contrôler les ressources et le temps alloué à chacune des tâches, ce qui a grandement aidé à diminuer la probabilité des risques. L'utilisation de l'outil de messagerie instantanée Slack a grandement aidé dans les échanges d'informations et à rendre la communication beaucoup plus efficace. Slack a contribué beaucoup à la diminution du risque de mauvaise communication.

Les tâches les plus importantes sont proposées en priorité en considérant le niveau de difficulté ou la probabilité, les conséquences ou impacts sur les livrables et le niveau de risque sur le projet. L'emphase est mise sur l'importance d'avoir des tâches critiques terminées par rapport à celles qui ne le sont pas, car la conséquence de ne pas avoir une fonctionnalité importante est bien plus grave que la conséquence de ne pas avoir une tâche moins importante.

À titre d'exemple, une mauvaise gestion de communication peut entraîner une incompréhension dans la façon précise d'effectuer une tâche et génère des frustrations et des délais dans la livraison de la tâche. En autre, la tâche pourrait ne pas être effectuée de la manière prévue. Une des causes de cette mauvaise communication peut être attribuée au manque de précision des tâches sur Trello.

ID	Description	Prob.	Cons.	Risque
R01	Mauvaise communication au sein de l'équipe.	Probable	Sévère	Grave
R02	Absence de visibilité et/ou de décision inadaptée.	Possible	Sévère	Normal
R03	Perte de ressources disponibles.	Improbable	Sévère	Normal
R04	Délai insuffisant.	Probable	Sévère	Grave
R05	Spécifications trop ambitieuses.	Possible	Sévère	Normal
R06	Non-disponibilité des technologies.	Improbable	Irrécupérable	Normal
R07	Sous-estimation de la complexité.	Probable	Sévère	Grave
R08	Sous-estimation des ressources.	Possible	Sévère	Normal
R09	Changement de l'échéancier.	Improbable	Sévère	Normal

Tableau 04 : Analyse des risques

PROJET - Méthodologie de travail

La méthodologie de travail ayant été appliquée lors du développement de ce projet est la méthodologie *Agile* avec des sprints d'une durée d'une semaine. La méthodologie *Agile* découpe le cycle de développement en plusieurs processus. Ces processus consistent de : *Product Backlog*, *Sprint Planning*, *Sprint Backlog*, *Sprint Review*, *Sprint Retrospective* et *Increment*. L'objectif de cette méthodologie est d'avoir, à la fin de chaque *sprint*, une version incrémentale de l'application stable.

Processus

Le *Product Backlog* est un ensemble de toutes les tâches à accomplir durant le cycle de développement. Cet ensemble de tâches est dynamique et peut changer avec le temps en fonction des demandes du client.

Le *Sprint Planning*, soit la planification de *sprint*, est le processus où l'équipe de développement, aussi appelée *Scrum Team*, détermine les tâches qu'elle veut accomplir durant la période allouée au *sprint*. L'équipe de développement courante a opté pour des *sprints* d'une semaine.

Les tâches choisies pour une période de *sprint son*t ensuite mises dans le *Sprint Backlog*. Cet ensemble, en bonne pratique, ne devrait pas changer durant le *sprint*. S'il y a de nouveaux ajouts que l'on veut avoir dans le projet, ceux-ci sont déposés dans le *Product Backlog* s'ils ont été approuvés.

Le *Sprint Review* consiste en des démonstrations des tâches accomplies durant la période de *sprint* aux autres membres de l'équipe ainsi qu'au client pour montrer l'avancement du projet. L'accumulation des tâches accomplies durant le *sprint* est considérée comme étant une incrémentation de l'application.

Le *Sprint Retrospective* est une rencontre du *Scrum Team* afin de discuter des points forts et faibles du *sprint* terminé. Il y a aussi des discussions sur les tâches incomplètes et leurs raisons afin de trouver une solution le plus rapidement que possible. Suite au *Sprint Retrospective*, l'équipe de développement débute un nouveau *sprint* avec l'étape du *Sprint Planning*.

Dans le cadre de ce projet, l'équipe a décidé de négliger les rencontres quotidiennes pour des raisons de manque de disponibilité des membres de l'équipe. Elle a donc opté pour des rencontres hebdomadaires où il y aura seulement le *Sprint Planning* et le *Sprint Retrospective*.

Outils de gestion de projet

Slack

Slack est une plateforme de communication collaborative propriétaire (Saas) ainsi qu'un logiciel de gestion de projets créé par Stewart Butterfield en août 2013. Slack fonctionne à la manière d'un chat IRC organisé en canaux correspondant à autant de sujets de discussion. La plateforme permet également de conserver une trace de tous les échanges, permet le partage de fichiers au sein des conversations et intègre en leur sein des services externes comme GitHub.

Trello

Trello est un outil de gestion de projet en ligne, lancé en septembre 2011, et inspiré par la méthode Kanban de Toyota. Il est basé sur une organisation des projets en planches listant des cartes, chacune représentant des tâches. Les cartes sont assignables à des utilisateurs et sont mobiles d'une planche à l'autre, traduisant leur avancement.

Google Drive

Google Drive est un service de stockage et de partage de fichiers dans le nuage lancé par la société Google. Google Drive, qui regroupe Google Docs, Sheets et Slides, Drawings, est une suite bureautique permettant de modifier des documents, des feuilles de calcul, des présentations, des dessins, des formulaires, etc. Les utilisateurs peuvent rechercher les fichiers partagés publiquement sur Google Drive par l'entremise de moteurs de recherche Web.

GitHub

GitHub est un service Web d'hébergement et de gestion de développement de logiciels, utilisant le logiciel de gestion de versions Git. GitHub propose des comptes professionnels payants, ainsi que des comptes gratuits pour les projets de logiciels libres. Le site assure également un contrôle d'accès et des fonctionnalités destinées à la collaboration comme le suivi des bogues, les demandes de fonctionnalités, la gestion de tâches et un wiki pour chaque projet.

LucidChart

LucidChart est une application Web qui sert à produire des diagrammes. Cet outil fonctionne sur les navigateurs Web qui supportent HTML5. Ce logiciel permet à plusieurs personnes de travailler sur un diagramme en même temps. Cette application Web permet de dessiner divers types de diagrammes ayant un lien dans un domaine d’affaires spécifique tel que des diagrammes UML, diagramme de Gantt, diagrammes de ventes, etc.

PRODUIT - Description

Problématique

Une partie du métier d'avocat consiste à organiser et planifier les procès. Le protocole d'entente entre les avocats impose une procédure qui rend la planification préprocès et l'acceptation d'une date consentante extrêmement complexe. Les propositions de date, le remplissage de formulaire avec des formats de dates non standard, les échanges de courriels, l'accès à l'information à un endroit centralisé et la disponibilité des parties impliqués dans le procès ne sont pas toujours faciles à coordonner. Ces difficultés engendrent de longs délais dans une planification qui avait déjà des moments clés à respecter selon le protocole.

Le tableau ci-dessous présente de façon sommaire les éléments clés du processus de coordination des procès afin d'avoir une vue d'ensemble simple et claire sur la problématique existante.

Le problème de	*difficulté de coordination entre les parties prenantes lors de la planification des rencontres*
affecte	*les clients, les avocats et toutes les autres parties prenantes.*
Résultant en	*délais de planification, problèmes de communication et tâches requises non faites.*
Une solution serait	*un logiciel facilitant la communication et la planification des différentes parties prenantes.*

Tableau 05 : Analyse de la problématique

Sommaire des fonctionnalités

Afin de respecter le protocole d'entente entre les avocats, une liste de requis a été définie par le promoteur du projet. Le tableau ci-dessous présente plus en détail le rôle de chacune des fonctionnalités, leur degré d'importance et les efforts nécessaires pour leur implémentation. Sommairement, l'application doit obligatoirement offrir la possibilité de créer et modifier un événement, de grouper les tâches liées à un événement et d'afficher tous les événements disponibles via un calendrier. Avec une priorité moindre, mais essentielle, un système de vote sur les événements, la gestion des membres et l'authentification des parties prenantes doivent être aussi implémentés. Optionnellement, le filtrage et la recherche des événements ou des groupes de tâches à réaliser, ainsi que la suggestion des plages horaire pour chaque événement pourraient être implémentés.

Fonctionnalité	Description	Diff.	Priorité	Effort
Gestion des comptes	*Permet aux utilisateurs de s'inscrire ou changer toutes les informations relatives à leur compte.*	Basse	Moyen	50 heures
Authentification	*Permet aux utilisateurs de se connecter ou déconnecter du logiciel.*	Basse	Moyen	75 heures
Gestion des tâches	*Permet aux utilisateurs de gérer des tâches pour chaque événement.*	Basse	Moyen	50 heures
Gestion des événements	*Permet aux utilisateurs de gérer des événements.*	Moyen	Haute	50 heures
Calendrier	*Permet aux utilisateurs de visualiser les événements planifiés*	Élevé	Haute	100 heures
Système de votation	*Permet aux utilisateurs de voter sur les dates d'événements pour lesquels ils doivent assister*	Moyen	Moyen	75 heures

Tableau 06 : Analyse des fonctionnalités

Product Breakdown Structure (PBS)

L'application Protogest 2.0 est subdivisée en trois principaux services, soit le l'infrastructure, qui englobe les solutions d'hébergements et les méthodes de déploiement, le *Front-End,* qui représente la passerelle entre les utilisateurs vers le service de *Back-End,* qui lui encapsule les manipulations logiques de l'écosystème.

La fondation de l'écosystème technologique a été mise en place par l'équipe de la phase 1. En partant de la fondation mise en place, l'équipe 2 a poussé plus loin son infrastructure de manière à ce que l'application soit prête pour le déploiement ainsi que le développement continu avec des services infonuagiques. Les services utilisés doivent, entre autres, être en mesure d'héberger l'application en ligne dans son entièreté.

Le *Front-End* utilise la technologie *Angular* qui met l'accent sur la décomposition des pages Web en petits morceaux réutilisables. Pour ce faire, cette plateforme permet la division des pages en composants qui sont indépendants les uns des autres sauf si déclaré explicitement. Les fonctionnalités communes sont placées dans des fichiers *JavaScript* fortement typés, soit en *TypeScript*, et sont par la suite importées dans les composants qui en ont besoin. L'importation de ces services ou de librairies *JavaScript* est faite à travers les modules associés à chaque composant. De cette manière, l'importation des libraires se fait de manière dynamique. Chaque fichier est chargé et déchargé lorsque nécessaire seulement.

Le *Back-End* consiste d'un *API REST* découpé en microservices où chaque microservice est indépendant. Cette décision a été prise par les prédécesseurs du projet et demeure une bonne pratique au niveau applicatif. Les microservices sont modulaires, leur permettant d'être modifiés et déployés sans avoir un effet direct sur les autres microservices.

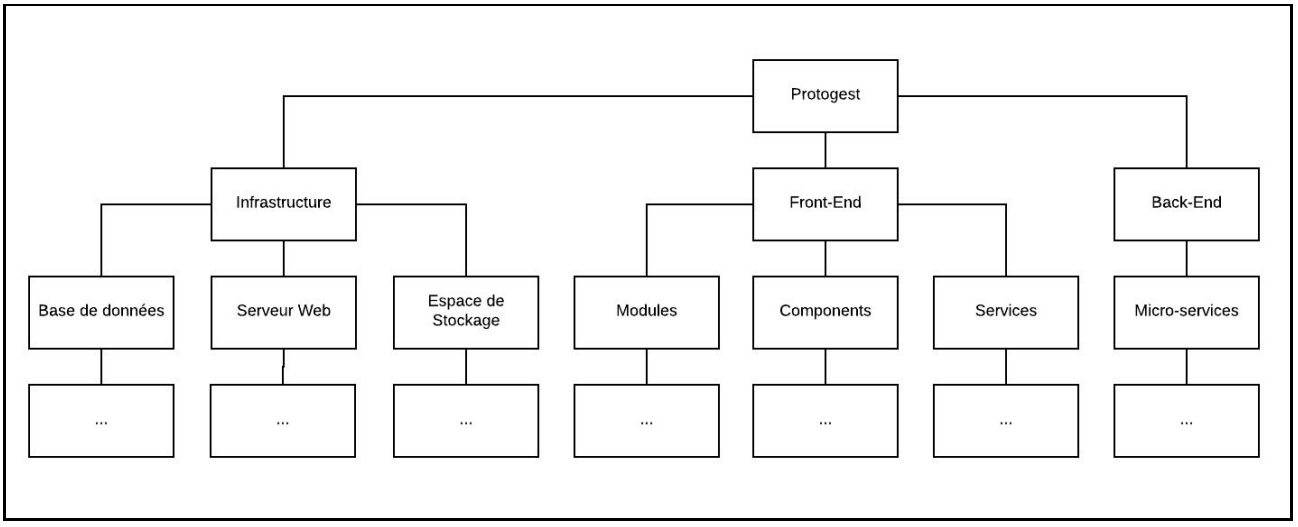


Figure 02 : *Product Breakdown Structure* (PBS)

PRODUIT - Technologies et Outils de développement

Motivations

Le projet, ayant pour objectif de migrer vers le *OpenSource*, utilisait des ressources propriétaires avec les technologies de *ReactJS/Redux*. Pour assurer une bonne conversion vers un projet non propriétaire, l'équipe a opté pour une technologie plus complète et *OpenSource* que *ReactJS*, soit *Angular*. L'avantage d'utiliser *ReactJS* comparé avec *Angular* est sa légèreté. Par contre, cela signifie aussi que *ReactJS* est limité en termes de fonctionnalités fournies sans l'aide de librairies externes.

De plus, l'hébergement et le déploiement continu se sont tournés vers AWS puisqu'il est un des grands *leaders* du marché qui propulse les solutions infonuagiques. De plus, il est facile de trouver la documentation nécessaire à sa compréhension.

L'équipe a choisi de rester avec la technologie *Spring Boot* pour le développement du *Back-End* pour faciliter le développement. De plus, ce choix offrira l'opportunité aux développeurs d'acquérir des connaissances sur cette technologie grandissante.

Spring Boot

Spring est un framework libre ayant pour objectifs : définir l'infrastructure d'une application *Java*, facilité le développement ainsi que simplifier l'implémentation de tests. *Spring* est fût créé en 2004, mais son utilisation fût popularisé en 2009 lors de la publication de *Spring 3.0*.

Spring s'appuie principalement sur l'intégration de trois concepts clés :

1. l'inversion de contrôle : la recherche de dépendances et l'injection de dépendances;
2. la programmation orientée aspect;
3. une couche d'abstraction.

L'utilisation de cette technologie au sein du projet se justifie par la facilité que procure le framework Spring à développer un ensemble de microservices.

Apache Maven

Couramment appelé *Maven*, *Apache Maven* est un outil de gestion et d'automatisation de production des projets logiciels *Java* en général. L'objectif recherché est de produire un logiciel à partir de ses sources, en optimisant les tâches réalisées à cette fin et en garantissant le bon ordre de fabrication. *Maven* utilise un paradigme connu sous le nom de *Project Object Model* (POM) afin de décrire un projet logiciel, ses dépendances avec des modules externes et l'ordre à suivre pour sa production. Il est livré avec un grand nombre de tâches prédéfinies, comme la compilation de code *Java* ou encore sa modularisation.

Angular

Angular est un *framework* libre pour les interfaces Web utilisant le *TypeScript*, soit du *JavaScript* fortement typé. Le *framework* suit une architecture basée sur les composantes où chaque composante est basée sur le patron de conception MVC.

Une application a toujours au moins un module racine qui active le démarrage et généralement beaucoup plus de modules de fonctionnalités. Les composants définissent des vues, qui sont des ensembles d'éléments d'écran parmi lesquels *Angular* peut choisir et modifier en fonction de la logique et des données de l'application. Les composants utilisent des services, qui fournissent des fonctionnalités spécifiques non directement liées aux vues. Les services peuvent être intégrés aux composants en tant que dépendances, ce qui rend le code modulaire, réutilisable et efficace. Les projets *Angular* sont généralement des applications à une page, aussi connues comme étant des *Single-Page Application*. Dans ces applications, on y retrouve une combinaison des différentes composantes créées par le développeur.

Pour optimiser la performance de ce *framework*, celui-ci affiche le contenu nécessaire dynamiquement. Comme mentionné ci-dessus, chaque composante a ses propres modèle, vue et contrôleur. Lorsque le modèle change dans une des composantes, la composante en question s'occupe à rafraichir sa vue sans avoir un impact sur les autres composantes.

Amazon Web Service (AWS)

Amazon Web Services (AWS) est une plateforme de services infonuagique proposée par *Amazon.com*. Elle est dédiée aux services à la demande pour les entreprises et particuliers. Les premières offres AWS sont nées en 2006, dont les plus connus le EC2 (*Elastic Compute Cloud*) et S3 (Simple Storage Service) qui permettait uniquement la location des instances en mode hébergé, c'est-à-dire des machines virtuelles. Actuellement, AWS dispose d'un catalogue de services Web de plus de 90 services. Ces services sont répartis en plusieurs catégories : calcul, stockage, base de données, outils pour développeurs, outils de gestion, services mobiles, services applicatifs, messagerie, productivité d'entreprise, internet des objets, centres d'appels et autres. La plupart d'entre eux ne sont pas directement exposés à l'utilisateur final, mais offrent des fonctionnalités que d'autres développeurs peuvent utiliser à travers des API.

MySQL

MySQL est une marque déposée de *MySQL AB*. Dans le jargon informatique, *MySQL* est un système de gestion de bases de données relationnelles. Il est distribué sous une double licence *GNU GPL* et propriétaire. Les utilisateurs peuvent choisir entre utiliser *MySQL* comme un Logiciel libre, sous les termes de la licence *GNU General Public License* ou bien, ils peuvent acheter une licence commerciale auprès de *MySQL AB*. Le nombre d'API dont il dispose *MySQL* le rend très intéressant aux développeurs d'application. En effet, il s'intègre très facilement dans des applications écrites en : *C*, *C++, Eiffel, Java, Perl, PHP, Python, Ruby* et *Tcl*.

Outils de développement

Sprint Tools Suite

Spring Tools Suite (STS) est l'outil officiel pour développer un projet Spring Boot. Cet outil consiste en un Integrated Development Environment (IDE) Eclipse avec une extension développé spécifiquement pour intégrer les opérations nécessaires au développement d'un projet *Spring*.

Visual Studio Code

Visual Studio Code est présenté comme un éditeur de code multiplateforme, logiciel libre et gratuit. C'est un éditeur de code source léger, mais puissant qui s'exécute sur le bureau et est disponible pour *Windows*, m*acOS* et *Linux*. Il est livré avec un support intégré pour *JavaScript*, *TypeScript* et *Node.js* et possède un écosystème riche en extensions pour d'autres langages (tels que *C++*, *C#*, *Java*, *Python*, *PHP*, *Go*) et les environnements d'exécution (tels que *.NET* et *Unity*). Le code source est fourni sous la licence libre MIT tandis que l'exécutable est offert via le site officiel de Microsoft.

Swagger 2.0

Swagger est un logiciel libre offrant un large écosystème d'outils aidant la conception, le développement, la documentation ainsi que la production de tests d'un *API Web RESTful*. *Swagger* offre un éventail de fonctionnalités tel que la génération de documentation, la génération de code ainsi que la génération de tests. En outre, il définit un standard qui permet aux développeurs d'application de découvrir et comprendre les méthodes disponibles dans une application ou un service sans avoir besoin d'accéder à son code ou consulter une documentation supplémentaire. Lorsqu'une *API* a été correctement définie avec *Swagger*, les développeurs peuvent comprendre et interagir avec le service avec beaucoup moins d'effort d'implémentation.

MySQL Workbench

MySQL Workbench est un logiciel de gestion et d'administration de bases de données *MySQL* créé en 2004. Via une interface graphique intuitive, il permet, entre autres, de créer, modifier ou supprimer des tables, des comptes utilisateurs, et d'effectuer toutes les opérations inhérentes à la gestion d'une base de données. Pour ce faire, il doit être connecté à un serveur *MySQL*.

Node Package Manager

Node Package Manager (NPM) un outil de gestion de librairie open source pour la langue de programmation *JavaScript*. Apparu avec *NodeJS* en 2009 l'utilisation de *NPM* dépasse aujourd'hui l'environnement serveur. Il est aussi le gestionnaire par défaut dans l'environnement d'exécution *NodeJS*. Il est de plus en plus utilisé dans le développement des interfaces utilisateurs. Il est simple à utiliser et permet d'accéder au plus gros dépôt de paquets tous langages confondus.

- *NPM* permet aux développeurs de télécharger depuis le serveur vers les packages tiers écrits par d'autres pour une utilisation locale.
- Il permet aux développeurs de télécharger et d'installer le programme de ligne de commande écrite par quelqu'un d'autre pour utiliser le serveur local à partir du *NPM*.
- Il permet aux développeurs d'écrire leur propre programme package ou ligne de commande téléchargée sur le serveur pour les autres à utiliser *NPM*.

Cet outil est responsable d'installer, de mettre à jour et d'enlever les librairies désirées par l'utilisateur.

Angular Command Line

Angular Command Line (CLI) est une librairie open source qui facilite la création de projets Web utilisant la technologie *Angular*. Il permet de générer différents fichiers contenant un minimum de code pour le fonctionnement dans l'environnement *Angular*.

Bootstrap

Bootstrap est une librairie open source qui permet de facilement concevoir une page Web avec du *HTML*, *CSS* et du *Javascript*. *Bootstrap* est exclusivement utilisé pour le *Front-End*. Cet outil sert à faciliter le design de l'interface utilisateur, le design d'un site Web adaptatif selon la taille de l'écran. Cette librairie aide le développeur à économiser du temps en réduisant le temps nécessaire pour éditer les fichiers *CSS*.

Lombok

Lombok est une librairie *Java* s'intégrant automatique à un environnement de développement et autres outils de développement afin d'améliorer l'expérience de développement du programmeur *Java*. Cet outil évite au programmeur de coder les différentes méthodes d'accès, de comparaison, d'écrire et autres méthodes triviales.

PRODUIT - Conception, Migrations et Intégrations

Back-End

Afin de poursuivre les efforts de l'équipe précédente au sein de ce projet, il a été décidé de conserver les technologies utilisées en *Back-End*. Par contre, afin que l'équipe de développement du *Back-End* ait la possibilité d'acquérir de nouvelles connaissances, une nouvelle implémentation des microservices fut élaborée. Résultat, une nouvelle structure de microservices fut développée, une communication avec *DTOs* pour chaque opération à l'*API* fût implémentée, l'outil de développement *Swagger* fut intégré, la librairie *Lombok* fut intégrée, ainsi qu'une nouvelle structure pour chaque microservice a été choisie.

Architecture microservices

Afin qu'un client Web, tel qu'un client *Angular* ou *React*, puisse interagir avec le *Back-End*, il a été décidé de créer un RESTful API. Ainsi, toutes opérations exécutées par le client auront un point d'accès à l'*API*.

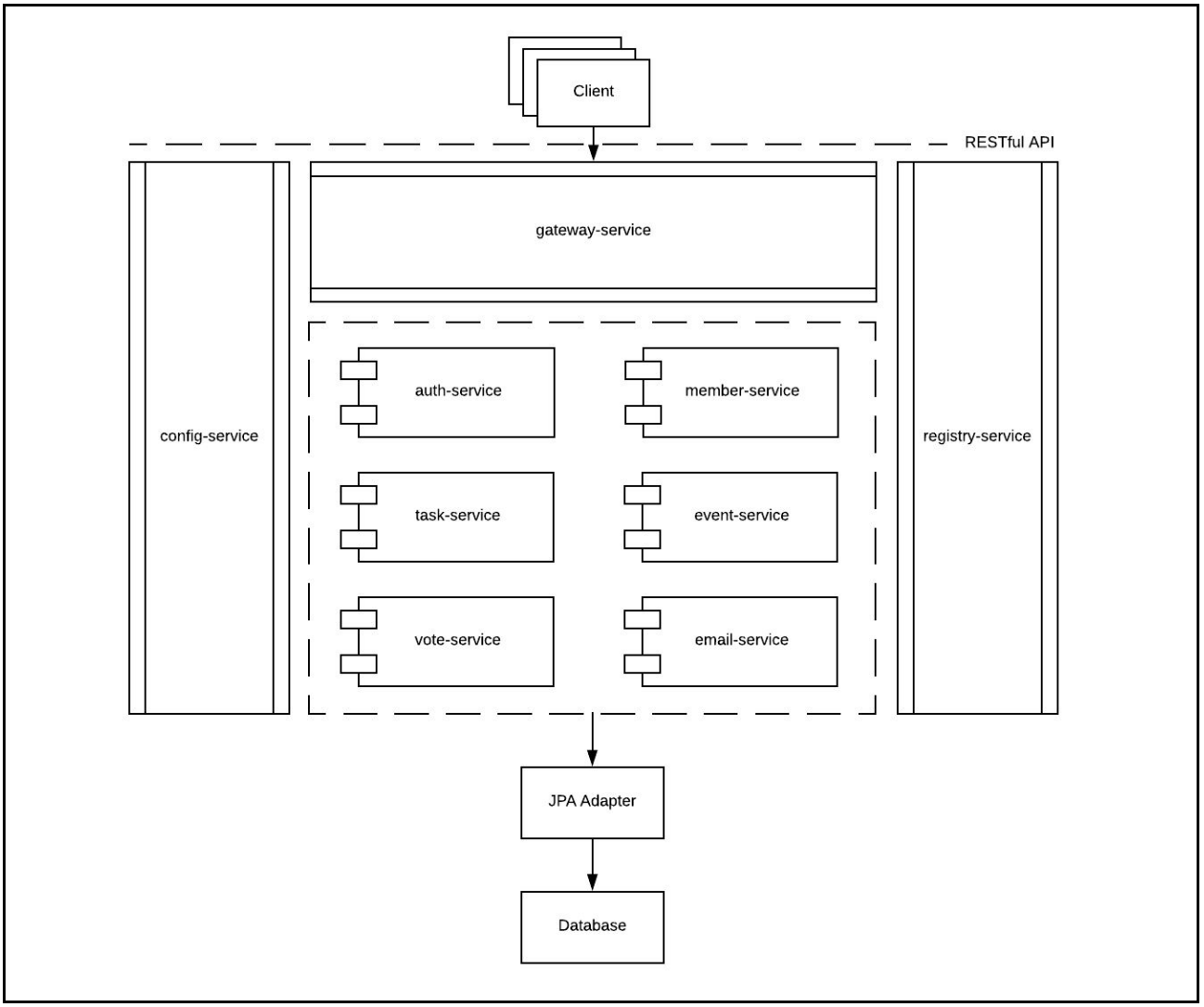


Figure 03 : Architecture du *Back-End*

Composant	Description
Client	*Une instance d'un client Web, tel qu'un client Angular ou React.*
RESTful API	*L'agglomération des APIs exposés par les différents microservices du système. Ceux-ci se doit d'exposer les méthodes standards : GET, POST, DELETE et PUT.*
config-service	*Microservice ayant la responsabilité de sauvegarder et partager les différentes configurations requises par les autres services, tel que les informations nécessaires pour accéder à la base de données.*
gateway-service	*Microservice ayant la responsabilité de filtrer et répartir les différents appels à l'API vers les autres microservice. Selon l'architecture, ce microservice peut avoir d'autres responsabilités telles que la sécurité des appels à l'API.*
registry-service	*Microservice ayant la responsabilité de gérer les instances des microservices du système. Chaque microservice s'enregistre auprès de ce microservice.*
auth-service	*Microservice exposant les opérations d'authentification et de sécurité au sein de l'API du système. Par exemple, l'inscription ou la connexion d'un utilisateur se fera via ce microservice.*
task-service	*Microservice exposant les opérations de gestion des tâches au sein de l'API du système. Ainsi, toutes créations, modifications ou suppressions de tâches ou groupe de tâches se feront via ce microservice.*
vote-service	*Microservice exposant les opérations relatives aux suggestions de vote de date d'un événement au sein de l'API du système. Ainsi, les suggestions de dates et les votes se feront via ce microservice.*
member-service	*Microservice exposant les opérations de gestion des membres au sein de l'API du système. Ainsi, toutes créations, modifications ou suppressions de membres se feront via ce microservice.*
event-service	*Microservice exposant les opérations de gestion des événements au sein de l'API du système. Ainsi, toutes créations, modifications ou suppressions d'événement ou groupe d'événements se feront via ce microservice.*
email-service	*Microservice exposant les opérations d'envoi de courriels spécifiques au sein de 'API du système. Ainsi, tout envoi de courriels se fera via ce microservice.*
JPA adapter	*Contexte offrant au programmeur la possibilité d'interagir avec la base de données via des entités plutôt que par des requêtes SQL.*
Database	*Base de données relationnelle du système.*

Tableau 07 : Description des composants du *Back-End*

Architecture d'un microservice

Afin de standardiser les microservices, une structure de base a été créée et appliquée à tous les microservice. Afin que chaque microservice puisse être déployé de façon indépendante, chaque microservice implémente les objets requis pour interagir avec la base de données : les entités et répertoires. De plus, afin de sécuriser la base de données et découpler l'utilisation des entités, des *DTOs* ont été créés.

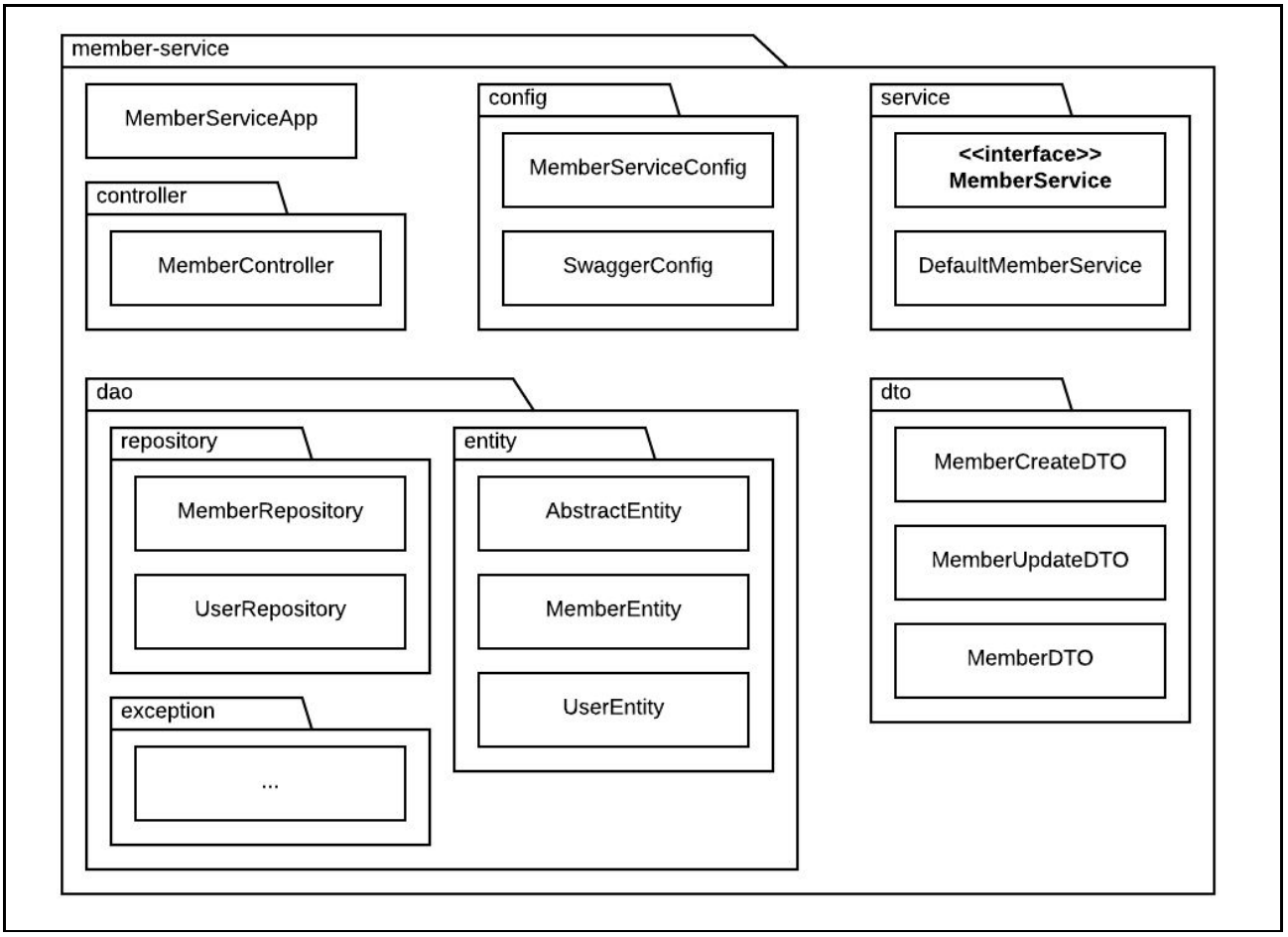


Figure 04 : Structure des classes d'un microservice

Package	Description
controller	*Package regroupant les différents controllers du service.*
config	*Package regroupant les différentes configurations du microservice. Entre autres, on y retrouve la configuration du microservice, la configuration de Swagger et la configuration de sécurité du microservice.*
service	*Package regroupant les différentes définitions des classes service du microservice.*
dao	*Package regroupant les différentes définitions des classes relatives à l'accès à la base de données. Entre autres, on peut y retrouver les packages repository, entity et exception.*

repository	*Package regroupant les différentes définitions des classes permettant d'interagir avec la base de données via le contexte JPA.*
entity	*Package regroupant les différentes définitions des classes représentant une instance de données au sein de la base de données.*
exception	*Package regroupant les différentes définitions des exceptions pouvant être rencontrées lors des interactions avec la base de données.*
dto	*Package regroupant les différentes définitions des DTOs pour chaque interaction entre clients et contrôleurs.*

Tableau 08 : Description des *packages* d'un microservice

Également, afin de découpler les responsabilités des objets au sein d'un microservice, un service (*Spring Service*) fut créé pour chaque microservice ayant comme responsabilité d'implémenter les différentes règles d'affaires (par exemple, les interactions avec la base de données) et un contrôleur (*Spring Controller*) ayant pour but de gérer toutes les interactions entre les clients Web et les microservices.

Pour maximiser l'optimisation, le mécanisme d'injection de *Spring Boot* fut utilisé pour injecter l'implémentation du service (*Spring Service*) au sein du contrôleur (*Spring Controller*).

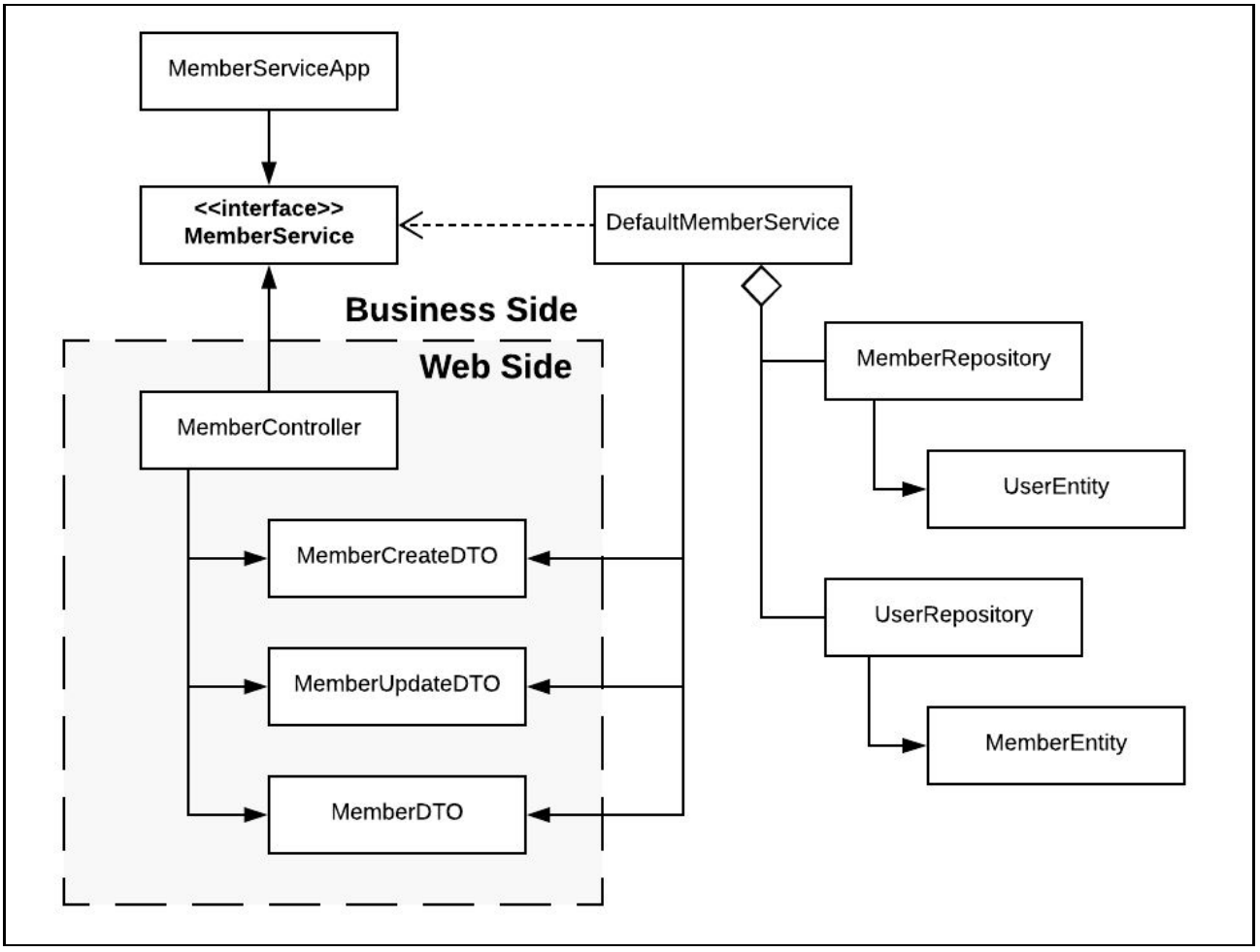


Figure 05 : Dépendances des classes d'un microservice

Classe	Description
***Service**	*Classes responsables d'implémenter les règles d'affaires du microservice (interactions avec la base de données). Cette classe est annotée comme Spring Service.*
***Controller**	*Classes responsables de gérer les interactions entre clients Web et microservice (par exemple, les requêtes Web). Cette classe est annotée comme Spring Controller.*
***DTO**	*Classes responsables de définir les informations requises pour les interactions entre clients Web et microservice.*
***Repository**	*Classes responsables de définir les opérations possibles auprès des entités de la base de données. Cette classe est annotée comme Spring Repository.*
***Entity**	*Classes responsables de définir les informations contenues au sein de la base de données.*
***ServiceApp**	*Classe principale pour chaque microservice. Cette classe est annotée comme Spring Application.*
***ServiceConfig**	*Classes définissant les configurations requises pour le microservice. Cette classe est annotée comment Spring Configuration.*

Tableau 10 : Description des classes d'un microservice

Afin de mieux comprendre le fonctionnement d'un microservice, voici la séquence d'événements qu'une opération provoquera au sein du microservice. Par exemple, voici la séquence d'événements pour l'opération de création d'un membre au sein du système.

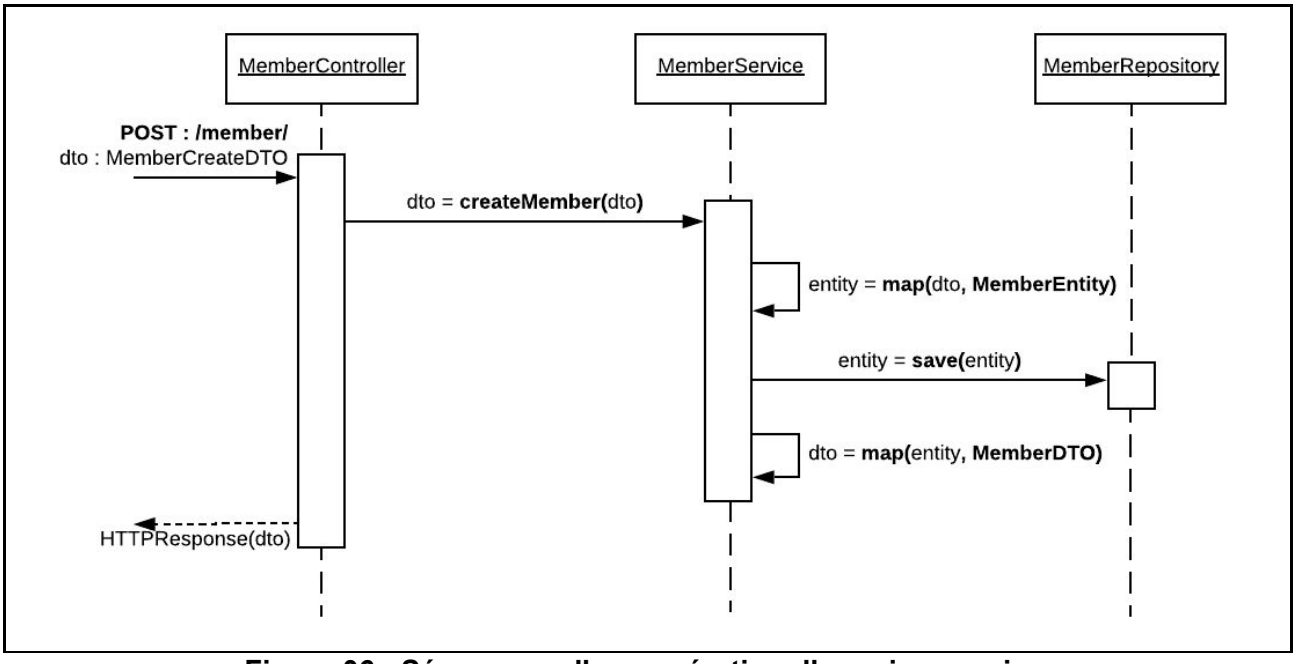


Figure 06 : Séquences d'une opération d'un microservice

Premièrement, le client Web exécute l'opération POST /member/. Le microservice Spring Boot détecte l'opération et passe l'événement au contrôleur du microservice. Celui-ci sauvegarde le DTO passé en paramètre, envoie la requête de création au service. Le service convertit le DTO en entité pour pouvoir le sauvegarder au sein de la base de données. Puis, pour compléter, celui-ci reconvertit l'entité en DTO afin de pouvoir le retourner au client Web via le contrôleur.

Front-End

Dans la liste des améliorations proposées sur le premier prototype de l'application, la migration du *Front-End* vers un cadriciel libre a été identifiée comme une priorité. Pour faciliter la migration vu les compétences disponibles pour la réalisation de l'interface utilisateur, le choix d'*Angular* et *Bootstrap* a fait consensus et a été validé par le tuteur du projet.

Un composant *Angular* est un constituant indépendant qui forme la base d'une application *Angular*. Plusieurs composantes permettent de former le *Front-End*. Chaque composante est constituée de *HTML*, de *CSS* et d'une classe en *Typescript*.

L'application Web utilisant *Angular* 6 se décompose en composant pour chacune des fonctionnalités, comme le démontre le schéma suivant.

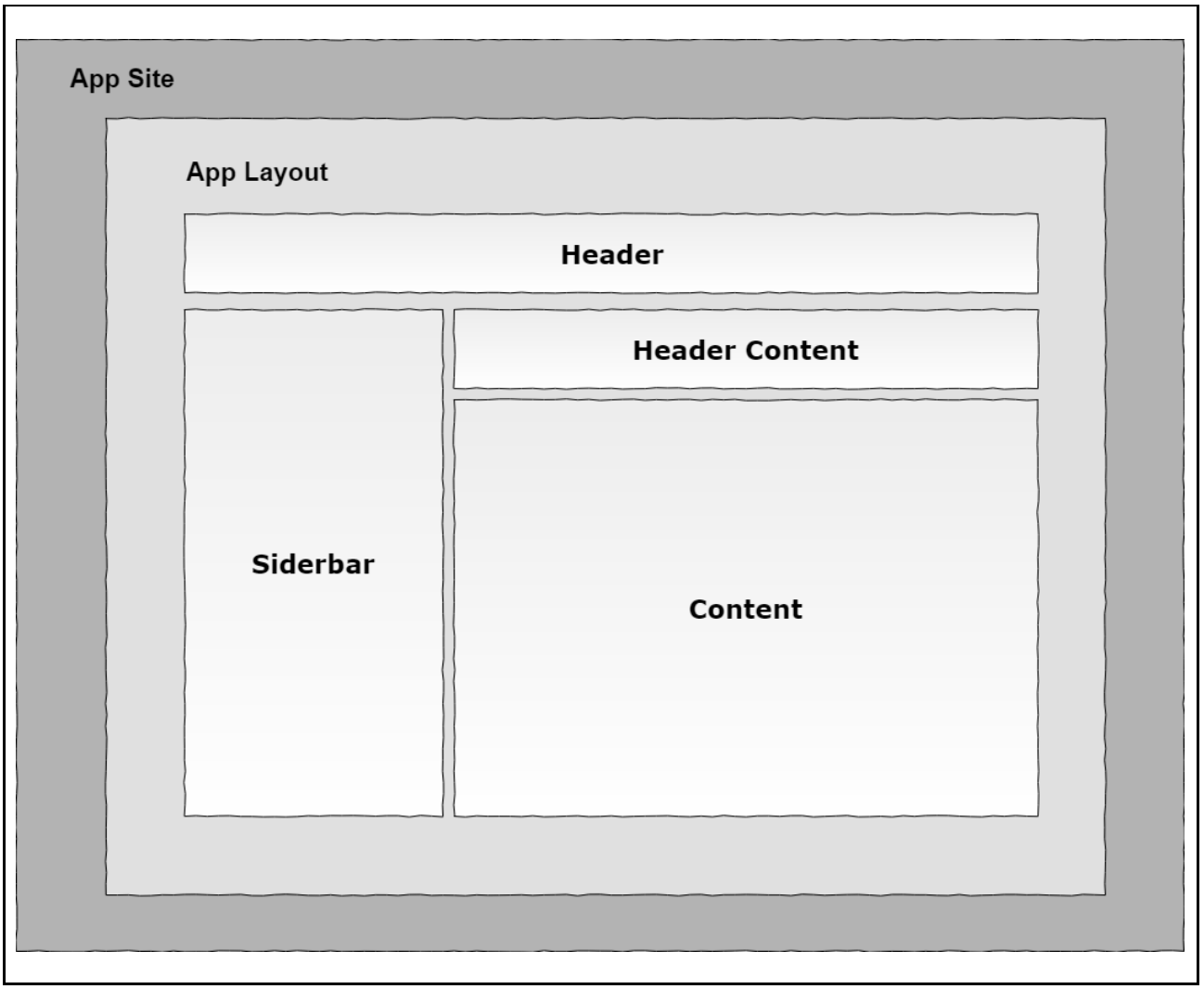


Figure 07 : Structure visuelle du *Front-End*

Le tableau ci-dessous présente une description à très haut niveau de la structure globale de l'application Web *Angular* Protogest.

Composant	Description
App Layout	*C'est la structure contenant l'ensemble des composants qui affichent une interface aux utilisateurs.*
Siderbar	*Contient le menu de navigation principale de l'application.*
Header	*Contient l'entête principal de l'application, soit l'accès aux notifications et au profil de l'utilisateur.*
Content & Header Content	*Contient l'ensemble des composantes formant les différentes pages de l'application, en excluant la page d'enregistrement et d'authentification. Cette composante est au même niveau que l'entête et le menu de navigation vertical.*

Tableau 11 : Description des composants visuels du *Front-End*

Le diagramme des flots des écrans ci-dessous présente les interconnexions ou les liens existants entre les affichages de l'application.

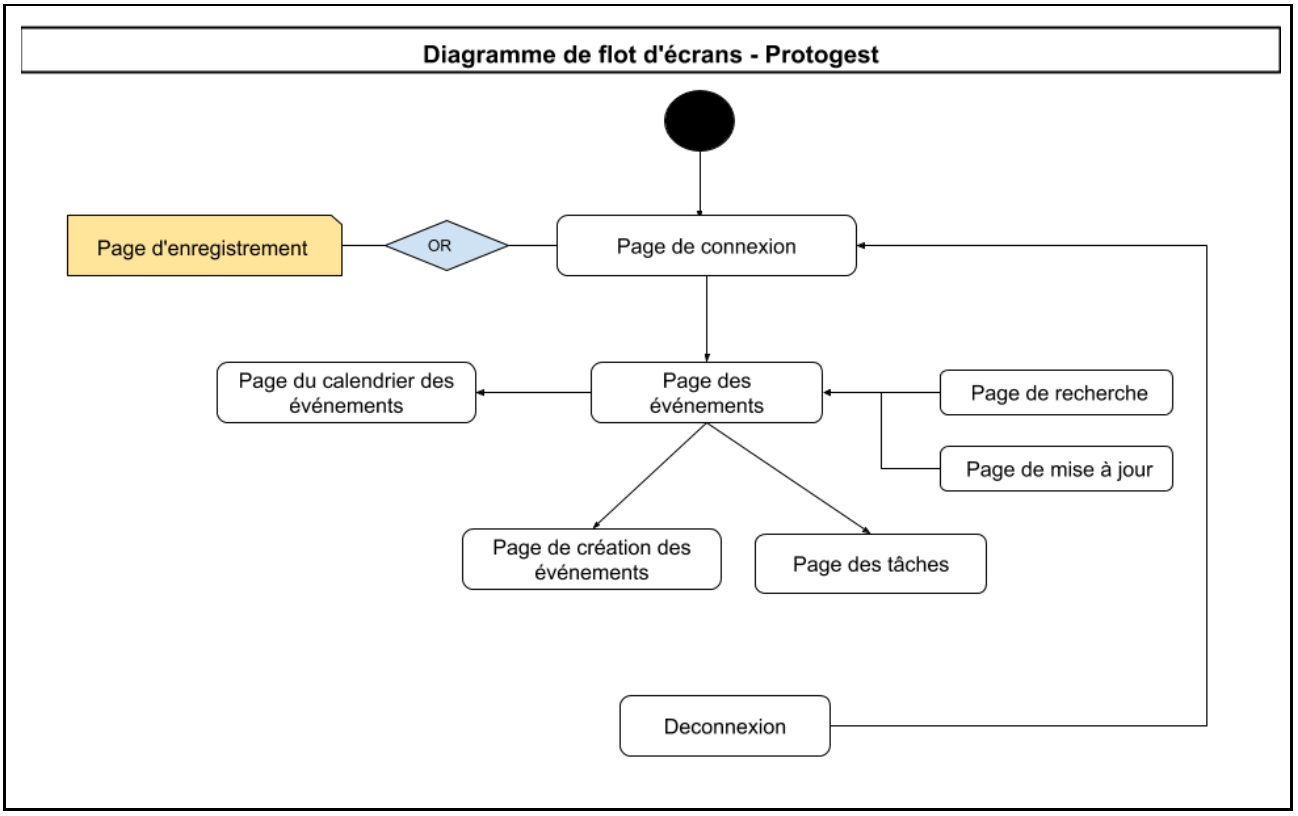


Figure 08 : Flots des écrans du *Front-End*

Architecture de communication des composants

Angular impose une approche structurée à base de composants et une façon claire d'échanger les données entre les composants. *Angular* encourage une implémentation par composant permettant

de réutiliser plus facilement du code *Angular* dans d'autres contextes. Un composant est un élément indépendant, réutilisable et responsable d'une seule action métier.

Routage

Le routage est un peu particulier, car il déroge un peu de la structure standard proposée par *Angular*. L'approche consiste à inclure un fichier de routage par composante de telle sorte que la navigation interne est distribuée et gérée localement par les composants. Ainsi, un fichier de routage racine est disponible et est associé à l'*app* qui est la composante racine de l'application.

Composants

Les composants correspondent aux différentes parties de l'application (page d'accueil, le calendrier, la liste des événements, etc.), il va comprendre un controller, une vue, un scss et une route. Comme dit plus haut, chaque composant est individuel, il ne dépend pas des autres et fonctionne comme une petite application *MVC*. Chaque composant définit une classe contenant des données d'application et une logique, associée à un modèle *HTML* définissant une vue à afficher.

Services

Les services servent à faciliter le partage entre les composants. Une définition de classe de service est immédiatement précédée d'un décorateur. Le décorateur fournit les métadonnées permettant à votre service d'être injecté dans les composants du client en tant que dépendance. Les services font la validation des entrées des membres et se connectent directement à la console. Un dossier de services partagés est aussi disponible dans l'application afin de permettre l'utilisation de certains services à plusieurs composants.

Modules

Un module déclare un contexte de compilation pour un ensemble de composants un flux de travail ou un ensemble de fonctionnalités étroitement liées. Les modules exposent à l'extérieur tout ce qui est nécessaire et qui peut être réutilisé dans d'autres applications ou d'autres composants. Un Module peut associer ses composants à un code associé, tel que des services, pour former des unités fonctionnelles. Protogest possède un module racine *AppModule* qui fournit le mécanisme d'amorçage pour son lancement et plusieurs modules fonctionnels.

Un module est aussi utilisé en tant que flux de traitement (*pipe*) dans l'application pour filtrer le contenu affiché dynamiquement. Ce type de module prend les entrées dynamiques et apporte les changements aux données avant qu'elles soient affichées sur les navigateurs.

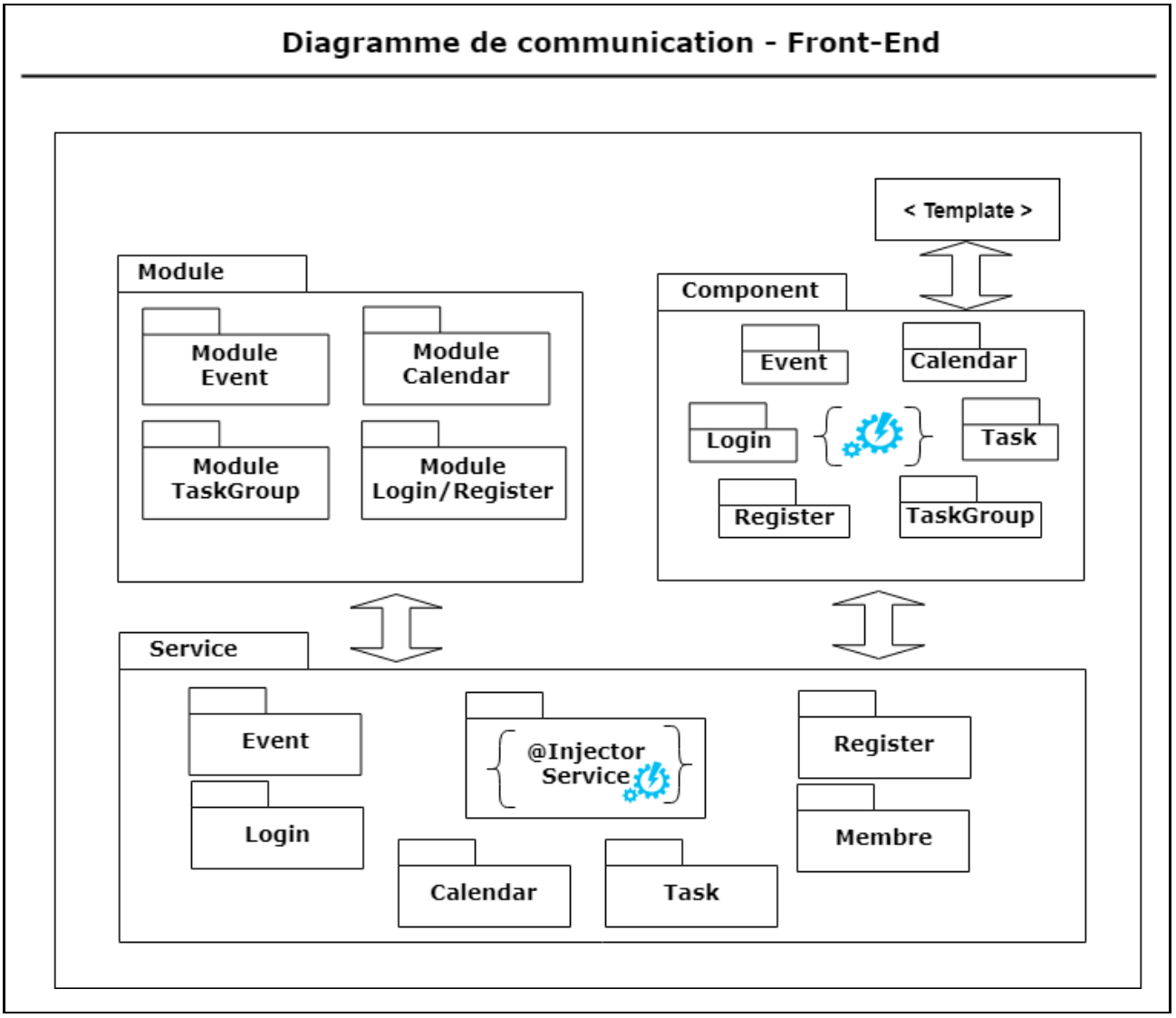


Figure 09 : Communications des composants *Front-End*

Composant	Description
connexion	*La composante Login est la composante qui va permettre à l'utilisateur de s'authentifier. Il y a 2 champs qui sont nécessaires pour s'authentifier: le nom de l'utilisateur et le mot de passe.* *Le service "user" va envoyer au Back-End les champs: nom de l'utilisateur et le mot de passe. La réponse obtenue du serveur Back-End est un objet contenant les informations de l'utilisateur, soit son identifiant et son nom d'utilisateur, qui est sauvegardé dans la mémoire locale(cache) du navigateur. Ceci permettra au serveur du Front-End de déterminer que l'utilisateur est déjà authentifié au système.*
register	*La composante Signup permet à l'utilisateur de se créer un compte dans l'application. Cette composante contient un formulaire que l'utilisateur aura à remplir pour créer son compte. Le formulaire contient les champs: nom de l'utilisateur, le mot de passe, la confirmation du mot de passe, le prénom de l'utilisateur, le nom de famille de l'utilisateur et le courriel de l'utilisateur.*

	Tout d'abord, les champs nom de l'utilisateur et le mot de passe vont être envoyé au service "user". Cette requête POST va permettre la création d'un utilisateur, mais ce n'est pas suffisant. *Il faut également créer un membre qui est associé à cet utilisateur. L'utilisateur concerne seulement l'aspect authentification. Le membre concerné l'aspect de l'action de l'utilisateur dans l'application, par exemple: créer un événement.* *La réponse du serveur par rapport à la requête pour créer l'utilisateur va permettre de récupérer l'ID de l'utilisateur. Ce même ID sera utilisé pour créer un membre.* *Une requête POST va être effectuée pour créer un membre. Les champs suivants vont être envoyés au Back-End: le prénom de l'utilisateur, le nom de famille de l'utilisateur, le courriel de l'utilisateur et ID de l'utilisateur*.
Create event	*La composante "Create-event" sert à créer un événement dans l'application. Cette composante va contenir un formulaire que l'utilisateur aura à remplir pour créer un événement. Ce formulaire contient les champs suivants: nom de l'événement, la description de l'événement et la date de l'événement.* *Le service de Create-event qui est "event" va envoyer au Back-End tous les champs du formulaire en plus de l'ID du membre qui crée l'événement et l'ID du group event qui est associé à ce membre.*
Event	*La composante "Event" est une vue servant à lister, filtrer, rechercher et éditer les événements d'un membre en particulier ou de tous les membres.*
Task	*La composante Task est une vue présentant tous les groupes de tâches associées à un événement spécifique. Ces groupes de tâches sont affichés dans une liste et permettent d'accéder à la vue contenant toutes les tâches associées à un groupe sélectionné.* *En accédant à cette composante, celle-ci récupère l'identifiant de l'événement dans les paramètres du URI qui est nécessaire pour la requête HTTP GET vers le microservice Task Service pour récupérer les TaskGroups associé à cet événement.*
Calendar	*La composante Calendar, ou bien scheduler, permet aux utilisateurs de visualiser les événements qui sont planifiés sur une base horaire. Présentement, cette composante récupère tous les événements qui sont sauvegardés dans la base de données. Elle utilise aussi une librairie JavaScript nommée DayPilot qui permet l'affichage d'un calendrier incluant diverses options configurables. Pour ajouter des événements au calendrier, ce dernier prendra en paramètre une liste d'objets qui contient le nom de l'événement, la date et heure de début et de fin ainsi que la rangée dans*

	laquelle elle sera affichée.

Tableau 12 : Descriptions des composants du *Front-End*

Service	Description
Connexion	*Ce service sert à envoyer les requêtes HTTP vers le microservice auth-service du Back-End afin de faciliter l'authentification des membre déjà enregistrés*
register	*Ce service sert à envoyer les requêtes HTTP vers le microservice auth-service du Back-End afin de faciliter l'inscription d'un nouveau membre.*
member	*Le service Member est le fichier service qui est responsable d'effectuer les requêtes HTTP vers le microservice member-service du Back-End. Ce fichier contient chacune des diverses méthodes HTTP pour la communication avec le Back-End.*
Event	*Ce service sert à envoyer les requêtes HTTP nécessaires au microservice event-service du Back-End afin de faciliter les opérations de création, modifications ou suppressions d'événement ou groupe d'événements. Ce service facilite en outre l'affichage des événements dans le calendrier.*

Tableau 13 : Descriptions des services du *Front-End*

Infrastructure

Lors de ce projet, il a été décidé qu'une exploration des technologies d'hébergement devait être effectuée afin de trouver le meilleur moyen d'héberger l'application. Lors de la planification initiale, le promoteur de projet a suggéré l'utilisateur de la plateforme nuagique Amazon Web Service (AWS) à cause de son grand potentiel et sa popularité. Avec ce choix en tête, il a été nécessaire d'effectuer des recherches poussées afin de trouver par quel moyen transférer la base de données et héberger les microservices.

Base de données

La première étape fut de trouver par quel moyen il serait possible de transposer la base de données de la plateforme *H2* à la plateforme *MySQL*. Cette recommandation avait été faite par l'équipe de développement précédente. Afin d'accommoder le choix de langage pour la partie *Back-End*, il a été nécessaire d'effectuer des recherches sur le moyen d'implémenter ce type de base de données avec une application *Spring Boot*. De plus, puisque la technologie AWS a été choisie pour l'hébergement, il a été également nécessaire de rechercher par quel moyen héberger une base de données sur ce service. Le service Relational Database Service (RDS) a été choisi pour héberger la base de données *MySQL* sur AWS. Ce service était adapté pour faire la gestion de base de données relationnelle et l'utilisation est simple. Une instance *MySQL* fut alors créée sur le service afin de pouvoir prendre la place de la base de données *H2*. En utilisant cette instance, le service database-service a été modifié afin d'utiliser l'instance et ainsi créer la structure de la base de données. Cette structure est demeurée la même que la version précédente du logiciel. Ce

n'était pas l'objectif de l'équipe de modifier la structure de la base de données, mais plutôt d'assurer sa conversion et son hébergement.

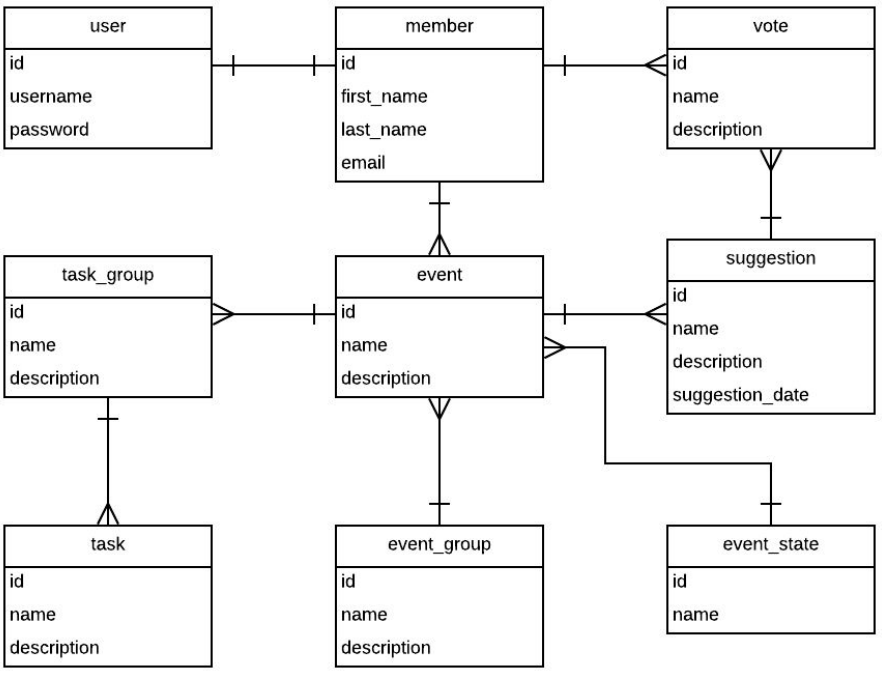


Figure 10 : Relations des tables de la base de données

Table	Description
Member	*Contient les informations relatives à un utilisateur de l'application. Contient en autre le courriel de l'utilisateur pour l'envoi des notifications par le email-service.*
User	*Contient les informations de connexion d'un membre. Est la table qui sera interrogée lors du processus de connexion.*
Event	*Contient la liste des événements qui seront affichés dans le module de calendrier. Contient l'information principale de l'application.*
Event_group	*Contient les groupes servant à rassembler les différents types d'événements. Est utilisé pour le filtrage des événements.*
Event_state	*Contient les différents états que peut avoir un événement. Est utilisé pour filtrer et désactiver des événements.*
Task_group	*Contient des groupes de tâches associés à un événement. Sert de table intermédiaire entre la table des événements et la table des tâches.*
Task	*Contient les informations de base pour une tâche qui est associée à un groupe.*
Suggestion	*Contient les informations sur les plages horaires suggérées pour chaque événement.*

Vote	*Contient les informations de vote de chaque utilisateur sur les différentes plages horaires suggérées.*

Tableau 14 : Descriptions des tables de la base de données

En choisissant de mettre la base de données en ligne en début de projet, il a été facile de coordonner les tests et le développement parmi tous les membres de l'équipe. Tous les membres de l'équipe avaient une source unique de données et il y a avait une diminution des requis lors de la préparation de l'environnement de développement pour chacun des membres de l'équipe.

Hébergement

Au début de la phase de développement, le promoteur du projet à suggérer l'utilisation du service Lambda de la plateforme AWS pour héberger le logiciel. Ce service qui a gagné en popularité durant les dernières années à cause de son concept. Celui-ci permet l'exécution de logiciels dans un environnement *serverless* c'est-à-dire sans serveurs. Cette méthode d'hébergement permet une certaine flexibilité sur les coûts d'opération d'un logiciel. Les fonctionnalités *Back-End* du logiciel sont séparées en fonction et sont exécutées uniquement lorsqu'elles sont appelées. Le restant du temps, ces fonctions restent en attente et n'utilisent pas de ressources. Le format Lambda permet de charger les clients uniquement lorsque des ressources sont utilisées. La division en microservice du logiciel est un avantage qui devait être exploité et qui était idéal pour l'utilisation du service *Lambda*.

L'objectif principal du développeur d'infrastructure était alors de trouver un moyen d'adapter le logiciel pour pouvoir être hébergé sur *Lambda*. Afin d'accomplir cela, plusieurs éléments devaient être considérés. Tout d'abord, une recherche a dû être faite afin de déterminer la bonne manière d'héberger une application *Spring Boot* sur *Lambda*. Deux approches ont été trouvées en ligne, la première nécessitant des modifications drastiques au code de chaque microservice et la seconde faisant plutôt l'usage d'une classe intermédiaire. La seconde approche a été choisie afin de ne pas affecter les microservices et d'assurer des modifications transparentes. Voici l'architecture visée pour l'application.

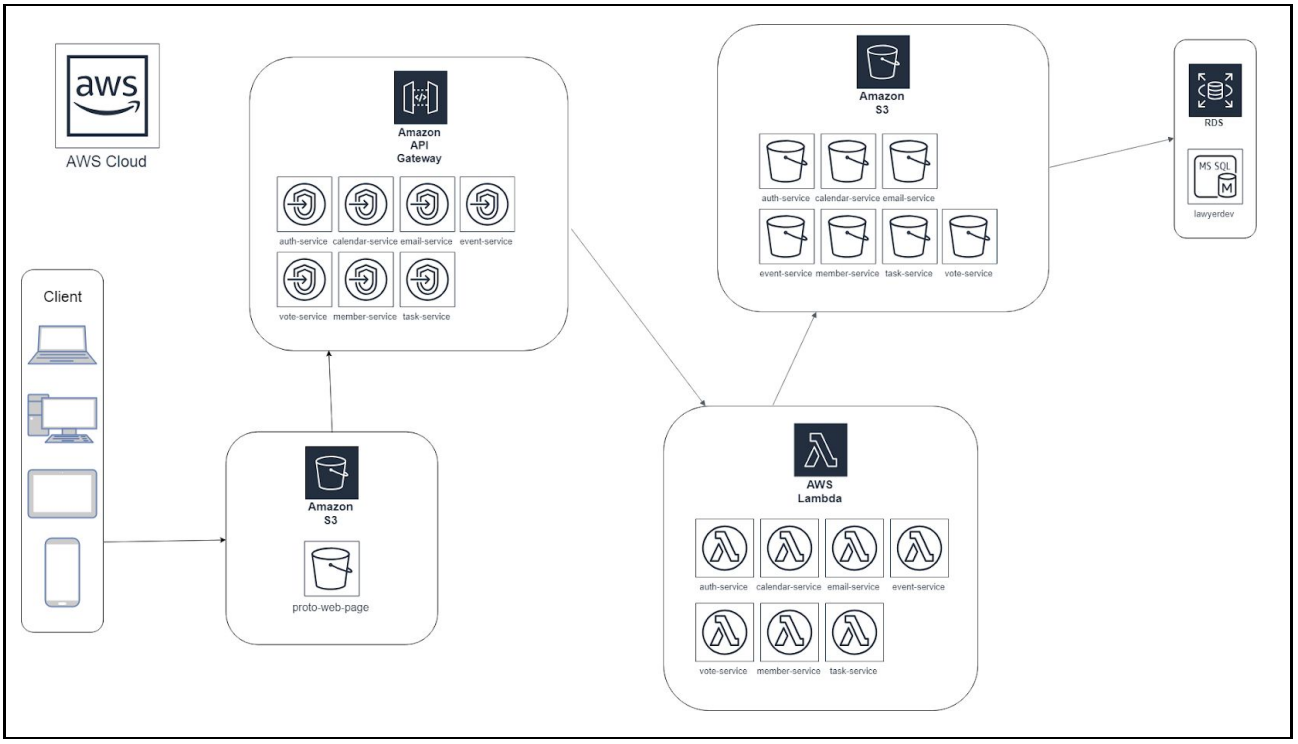


Figure 11 : Architecture d'hébergement visée

Tout d'abord, une classe intermédiaire doit être créée dans chaque microservice. Cette classe fait usage des librairies de container pour le service AWS. Cette classe va s'occuper de transmettre l'information entre le service *Lambda* et le microservice. Dans cette classe, une déclaration est faite pour pointer sur la classe principale du microservice qui sert à démarrer celui-ci. En plus d'implémenter cette classe, un fichier de configuration *serverless.yml* doit être créé afin d'établir les propriétés de la fonction *Lambda* ainsi que de l'API Gateway qui s'occupera de recevoir les requêtes du *Front-End* et de lui retourner l'information.

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: AWS Serverless member service
Globals:
  Api:
    EndpointConfiguration: REGIONAL

Resources:
  MemberServiceFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: com.pfe.ldb.member.StreamLambdaHandler::handleRequest
      Runtime: java8
      CodeUri: target/member-service-0.0.1-SNAPSHOT.jar
      MemorySize: 1536
      Policies: AWSLambdaBasicExecutionRole
      Timeout: 60
      Events:
        GetResource:
          Type: Api
          Properties:
            Path: /{proxy+}
            Method: any

Outputs:
  MemberServiceApi:
    Description: URL for application
    Value: !Sub 'https://${ServerlessRestApi}.execute-api.${AWS::Region}.amazonaws.com/Prod/'
    Export:
      Name: MemberServiceApi
```

Figure 12 : Configuration d'un microservice

Lors du déploiement du microservice sur *Lambda*, ce fichier sera utilisé afin de générer tous les artefacts nécessaires à l'exécution du microservice. L'*API* sera créé et l'utilisation du bloc *proxy* assure une détection automatique des différentes méthodes REST déclarées dans la classe contrôleur du microservice. Ce fichier permet également de définir le nom de la fonction *Lambda*, la méthode qui s'occupera du transfert des données entre l'architecture Lambda et le code *Spring Boot* ainsi que l'adresse du *gateway* qui sera défini dans les routes du *Front-End*. En général, le point d'entrée du logiciel sera à travers les *API Gateway*. Il est donc essentiel de définir cet élément pour chacun des microservices.

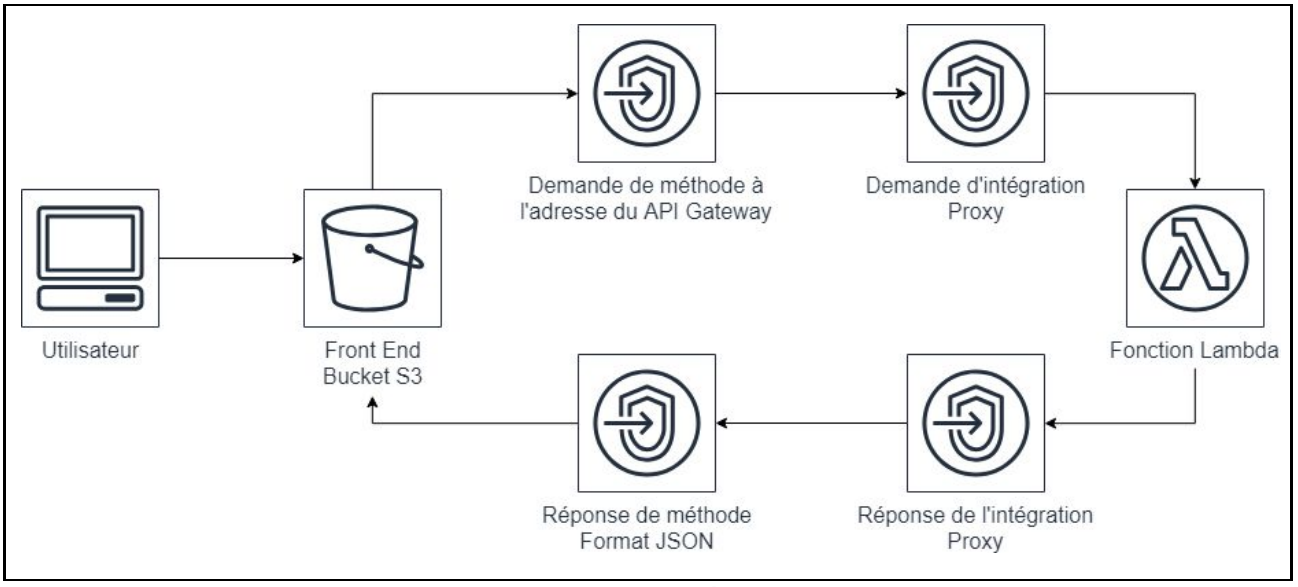


Figure 13 : Séquence d'une requête avec *Lambda* (AWS)

Alternativement, il est possible d'héberger l'application en utilisant un autre service sur AWS. En cas de problématique avec l'implémentation sur *Lambda*, le logiciel sera hébergé sur le service *Elastic Beanstalk*. Ce service se rapproche plus des méthodes traditionnelles d'hébergement d'application Web. Chaque service aura une instance associée qui s'exécute continuellement. Afin d'héberger l'application de cette manière, chaque microservice sera mis dans un dossier compressé généré avec les dépendances et téléversé sur son instance. Chaque instance a une adresse *HTTP* qui devra être mise dans les routes du *Front-End* à la place de l'adresse des *API Gateway*. Voici à quoi ressemblerait l'architecture si cette solution est utilisée.

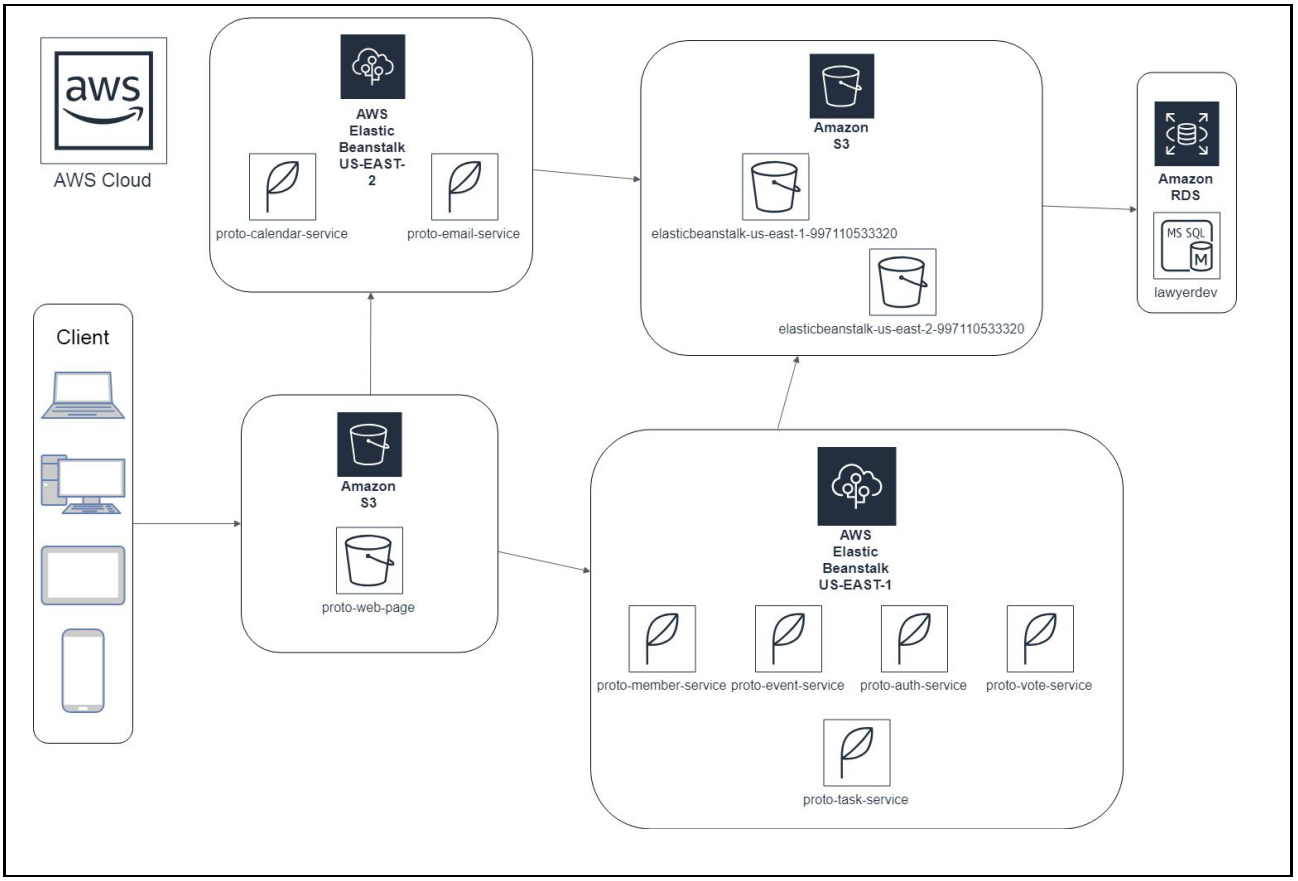


Figure 14 : Architecture d'hébergement alternative

DISCUSSION

Back-End

Architecture en microservices

La principale difficulté rencontrée lors du développement *Back-End* fut de se familiariser avec la structure d'un projet microservices *Spring Boot*. Il a été nécessaire d'apprendre les bonnes pratiques relatives à cette implémentation dont l'usage d'un microservice pour gérer les configurations (*config-service*), d'un microservice pour gérer les instances de microservices actifs (*registry-service*) et d'un microservice agissant en tant que passerelle pour les autres microservices (*gateway-service*).

De plus, l'implémentation de microservice pouvant être déployé indépendant fut plus compliquée que prévu. Afin de faciliter le déploiement du projet sur AWS, chaque microservice doit être capable de communiquer avec la base de données, et ce, sans l'usage d'un *package* ou service intermédiaire.

Architecture sans session

Une autre difficulté rencontrée fut celle de développer un *Back-End* sans l'usage de session. Ceci est nécessaire afin que le développement de plusieurs microservices simultanément soit possible, et ce, sans avoir besoin d'un package ou service intermédiaire.

Ainsi, contrairement à la version précédente du projet qui tentait tout de même de conserver des informations relatives à l'usager connecté, Protogest 2.0 ne conserve aucune information en cache et requiert toutes les informations nécessaires lors des opérations à l'API.

Sécurité des microservices

Lorsqu'il a été le temps de sécuriser les opérations et les points d'accès aux APIs des microservices, l'équipe de développement du *Back-End* a frappé un mur. Était-il nécessaire d'implémenter une sécurité pour chaque microservice ou suffit-il de développer une sécurité qu'au sein du microservice passerelle? De plus, quelle stratégie de sécurité est-il nécessaire d'être implémenté? Finalement, dans le cadre de ce projet, est-il nécessaire d'implémenter cette sécurité? La réponse, non. Ainsi, afin de se concentrer sur d'autres objectifs, il a été choisi de ne pas sécuriser les points d'accès de l'API des microservices.

Communication avec *DTOs*

Une des limitations présente dans la version originale de Protogest était l'exposition des entités auprès du client. C'est-à-dire, le client Web devait communiquer à l'API à l'aide des entités directement contenues dans la base de données. Ceci crée une violation à la sécurité de l'application et ajoute une complexité aux opérations, car elles nécessitent l'ajout de paramètres supplémentaires. Pour contrer cette limite, l'implémentation de Data Transfer Object (DTO) fût créée.

Ainsi, pour chaque opération de l'API fut créée un *DTO* ayant les champs requis pour compléter l'opération. La difficulté fut de convertir ces *DTOs* en entités pertinentes selon le contexte. De plus, cette stratégie permet de découpler l'implémentation de la base de données avec l'implémentation des opérations de l'API. Donc, si la structure de la base de données doit être modifiée, l'implémentation ne nécessitera aucun changement. De plus, cette stratégie facilite les modifications aux différentes opérations de l'API puisque chaque opération (ou presque) possède son *DTO*.

Suppression de microservices

Après analyse, le microservice calendar-service créer au sein de la version précédente du *Back-End* était fortement couplé avec le *Front-End*. Afin de découpler son utilisation, celui-ci fût supprimé, car toutes les informations nécessaires à l'affichage des événements au sein du calendrier peuvent être accédées avec le microservice event-service.

Front-End

Routage pour chaque composante

La structure de routage décrite dans l'architecture du *Front-End* a ses avantages, cependant cela complexifie beaucoup la mise en place de la navigation vers les liens. À la place de regarder dans un seul fichier, il faut se déplacer d'un fichier de routage vers un autre fichier de routage pour analyser la navigation de l'application. Lorsqu'un changement doit être effectué, plus d'un fichier de routage doit être modifié pour avoir le résultat escompté. Ce qui amène une complexité additionnelle à gérer dans la structure évolutive de l'application.

Module pour chaque composante

Chaque composante a son propre module. Le module sert généralement à faire déclarer la liste d'*imports* de modules, d'*exports* de modules et des différentes composantes. Normalement, chaque application doit avoir un seul fichier module. L'application ne suit pas cette approche.

Il est plus simple et pratique d'avoir un fichier module pour toute l'application, car il est plus facile de savoir où regarder exactement. Cependant, il faut vérifier l'*import*, l'*export* et la déclaration des composantes pour chaque composante.

Conversion du template

La conversion du *template* est une des causes d'une autre difficulté rencontrée. La copie des fichiers *HTML* et *CSS* a permis de faire la conversion du template plus rapidement. Cependant, le code source n'a pas été conçu par l'équipe. Cela fait en sorte qu'il y a plus de difficultés à modifier du code qui n'est pas fait par soi-même. Certains éléments de l'interface utilisateur étaient difficiles à positionner à cause de certaines propriétés *CSS* qui ont été appliquées de façon globale à l'application. Il y avait une difficulté d'appliquer ces changements à un élément graphique précis sans modifier les propriétés *CSS* globales.

Mécanisme de sécurité *Cross-Origin Resource Sharing*

Lorsque des requêtes sont faites vers le *Back-End* à partir du *Front-End*, il y avait des erreurs de *Cross-Origin Resource Sharing* qui empêchait la requête d'être exécutée. *Cross-Origin Resource Sharing* est un mécanisme de sécurité intégré au navigateur Web Chrome et Firefox. Pour remédier à ce problème de façon temporaire, il était nécessaire de désactiver ce mécanisme de sécurité pour pouvoir correctement tester les requêtes *HTTP*. Il n'était pas facile de désactiver ce mécanisme avec les configurations de base. Des modifications plus avancées étaient nécessaires, ce qui constitue une difficulté supplémentaire.

Intégration de la nouvelle interface Web gratuite

Une version payante de l'interface Web *React* a été utilisée initialement. Il a été demandé de faire une migration vers une solution gratuite. Il fallait au minimum reproduire toutes les fonctionnalités de l'ancienne interface Web en prenant en considération le niveau de connaissance des développeurs *Front-End* de la technologie qui sera utilisée. Des semaines de lecture et d'analyse ont conduit vers *Angular 6* comme technologie de développement, ainsi un prototype *Bootstrap 4* a été choisi du même coup pour la poursuite des travaux. Une application de base *Angular* a été mise en place pour faciliter l'intégration de la première version du prototype qui a été choisi. Les difficultés rencontrées lors de la mise en place des liens entre les différents composants lors des tests d'intégration avec la structure en place dans le dépôt *GitHub* ont forcé le choix d'une autre version plus complète.

Configuration d'un environnement de développement fonctionnel

Lors d'une séance commune, après le démarrage du projet, l'un des membres de l'équipe qui avait implémenté l'application *Spring Boot* (*Back-End*) a aidé tous les membres de la nouvelle équipe à installer et configurer les outils nécessaires afin de bien comprendre le fonctionnement de l'application sur laquelle les améliorations doivent être faites toute la session. Afin de faciliter la documentation des changements effectués au niveau de l'API, l'outil *Swagger* a été intégré dans l'application. Depuis, certains environnements de développement ont commencé à mal fonctionner ou ne fonctionne tout simplement pas. Ce qui a considérablement affecté le travail de quelques-uns et entravé la livraison de livrables testés et fonctionnels à cent pour cent. Une des solutions de contournement utilisées était de demander de l'aide à quelqu'un d'autre à chaque fois que des tests étaient nécessaires. Ceci n'a pas toujours fonctionné malheureusement.

Tests de la version précédente

Durant le démarrage du projet, l'équipe de développement devait faire une analyse sur le fonctionnement de l'ancienne application Web afin de mieux comprendre les différentes composantes provenant de la solution payante. La technologie *Redux* ne notifie pas les changements d'état au fureteur Chrome et nécessite l'ajout de l'extension *Redux Devtools* pour pouvoir faire du débogage.

Implémentation de composant

Il fallait trouver une alternative à l'utilisation du Google Calendar qui a été intégré dans la version de l'interface Web *React* pour l'affichage des événements d'un membre. Des solutions disponibles en version d'essai ont été testées, mais n'offrent pas toutes la latitude et le résultat escorté.

Trouver la bonne version à utiliser pour le projet dans la liste des utilitaires (*FullCalendar, DayPilot* et *DHTMLX Scheduler*) testés en dehors de l'application reste un vrai défi.

Infrastructure

Migration *H2* à *MySQL* sur *RDS*

Lors de la période de développement, il a été nécessaire de rechercher le meilleur moyen pour transférer l'application d'une base de données dynamique à une base de données statique hébergée en ligne. Après la création de l'instance sur RDS, la modification des informations dans le *database-service* a été suffisante afin de créer la structure des tables et les relations dans le nouvel environnement. Par contre, des modifications supplémentaires étaient nécessaires afin de permettre aux différents services d'accéder à cette base de données. Avec l'usage de fonctionnalités *Spring Cloud Function*, chaque service contient les informations relatives à leur source de données dans leur fichier *bootstrap*. En utilisant l'adresse de l'instance RDS et les informations de connexions du compte administrateur, tous les fichiers furent modifiés afin de permettre à chaque service de fonctionner de manière autonome sans l'usage du *database-service*. C'est d'ailleurs pour cette raison que ce service fut retiré de la version finale du produit.

Connexions refusées à la base de données

Lors de la phase initiale de la conversion, il été remarqué que ce n'était pas tous les membres de l'équipe qui étaient en mesure de se connecter à la base de données. Seul l'administrateur de la base de données pouvait se connecter. Après de la recherche, il a été trouvé qu'il y avait des règles de sécurité qui avaient été créées lors de la création de l'instance RDS. L'instance n'autorisait que les connexions provenant de l'adresse de l'administrateur de la base de données. Comme mesure temporaire, cette règle a été modifiée pour permettre les connexions sur le port *MySQL* 3306 provenant de n'importe quelle adresse IP. Afin d'assurer une plus grande sécurité, il serait possible de mettre en place des règles de sécurités afin de permettre des connexions provenant de zones et d'adresses spécifiques.

Moteur des tables

Lors de la période de développement, des modifications sur la structure de la base de données ont voulu être faites. Cependant, ces modifications n'ont pas été faciles à faire à cause du moteur des tables dans la base de données. Lors de la création des tables, celles-ci furent initialisées en format *MyISAM*. À cause de ce format, l'usage de certaines fonctionnalités relationnelles était impossible tel que l'utilisation de clés étrangères pour faire des liens en les tables. Afin de ne pas introduire d'erreurs supplémentaires, des changements à la structure des tables ont été faits le moins possible. Comme recommandation future, le moteur des tables devrait être changé pour le format *InnoDB* afin de permettre l'usage des fonctionnalités manquantes.

Multiples connexions à la base de données

Une difficulté rencontrée à la fin du projet fut la limitation des connexions à la base de données. Utilisant la valeur par défaut, une connexion à la base de données n'est limitée qu'à 30 minutes. N'ayant accès qu'à un compte AWS gratuit, l'instance de la base de données est limitée à 65 connexions. Normalement, une équipe ayant un compte AWS payant ne serait pas confrontée à

cette difficulté, car les connexions se fermeraient avant d'atteindre le nombre maximal. Autrement, après avoir utilisé l'application pendant quelques minutes, le nombre de connexions maximales était atteint. Pour résoudre ce problème, une limite de temps fut définie au sein des propriétés de la base de données.

Méthodes de déploiement pour *Spring Boot* sur *Lambda*

Durant la phase de développement du projet, un problème est survenu quand à la manière d'héberger une application *Spring Boot* sur *Lambda*. En effectuant des recherches sur le Web, plusieurs manières ont été trouvées sur le moyen de convertir une application *Spring Boot*. Cependant, il y avait plusieurs différences entre ces méthodes et cela a causé une grande perte de temps lors de la conversion du logiciel. L'équipe devait travailler beaucoup en essai et erreurs puisqu'il n'y avait pas moyen de savoir si les méthodes seraient fonctionnelles avec l'architecture particulière du projet. Puisque l'application utilise des fonctionnalités *Spring Boot* et *Spring Cloud*, plusieurs moyens différents existaient pour mettre chaque architecture sur *Lambda*. La méthode d'utilisation d'une classe *proxy* semblait le moyen le plus prometteur, mais celui-ci n'a pas résulté en succès.

Problème avec méthode *proxy*

Pour permettre l'hébergement de l'application sur *Lambda*, la méthode de conversion par *proxy* a été choisie. Cependant, plusieurs problèmes ont été rencontrés lors de la conversion des services. Tout d'abord, le format initial du fichier de configuration serverless n'était pas le bon pour fonctionner avec la classe *proxy*. Lorsque la fonction *Lambda* était exécutée, celle-ci n'arrivait pas à localiser la fonction s'occupant des requêtes au sein de la classe de proxy. Après la modification du fichier au format suggéré par la librairie pour AWS dans *Java*, la fonction Lambda était en mesure de passer à la prochaine étape. Cependant, une autre erreur a empêché le fonctionnement des microservices sur *Lambda*. Les librairies AWS importées dans les microservices entraient en conflits. À cause des différentes méthodes de conversion essayées, plusieurs librairies différentes étaient importées. Par contre, ces librairies importaient deux versions différentes de plusieurs fonctions. En retirant les librairies inutiles et en incluant les répertoires de configuration et d'entités dans les microservices, l'application a été en mesure de compiler correctement et d'être exportée sur la plateforme *Lambda*. Finalement, malgré le débogage, les microservices n'étaient toujours pas en mesure de fonctionner sur *Lambda*. Un débogage plus profond ainsi qu'une révision de l'architecture des microservices sont nécessaires afin de permettre l'hébergement des microservices sur *Lambda*.

Solution alternative d'hébergement sur *Beanstalk*

Puisqu'il n'a pas été possible d'héberger l'application sur *Lambda*, une solution alternative avait été préparée pour l'hébergement. Le service *Elastic Beanstalk* d'AWS a été choisi pour héberger les microservices. Ce service se rapproche des moyens d'hébergement classiques tel qu'utiliser un serveur dédié qui s'exécute sans arrêt et qui attend les requêtes. Par contre, des problèmes ont été rencontrés lors de l'hébergement sur cette plateforme. Tout d'abord, lors de la compilation et l'exportation sur l'instance, des messages d'erreurs se sont présentés lors de l'envoi de la première requête. La modification du port d'écoute de l'application et l'ajout de règles de sécurité sur l'instance n'ont pas permis de régler cette erreur. Après plusieurs recherches, la cause du problème a été ciblée. Le fichier de compilation de chaque microservice avait été modifié pour ne pas inclure la librairie *Tomcat* telle que suggérée pour l'hébergement sur *Lambda*. Cependant, ce

service était essentiel pour l'hébergement sur *Beanstalk*. Après l'avoir réintégré dans la compilation, les microservices étaient fonctionnels tels que planifiés. Cependant, les microservices ont dû être séparés sur deux régions à cause de la limite du nombre d'adresses IP permis par AWS. En rétrospective, les architectures d'hébergement ne sont pas tant différentes. À cause de l'impossibilité d'utiliser le service *Lambda*, les microservices ont été mis sur *Elastic Beanstalk*. Au lieu d'avoir un *API Gateway* et une fonction *Lambda* par microservice, ceux-ci sont assignés à une instance *Beanstalk* chaque. Il serait plus avantageux de mettre les microservices sur *Lambda* puisque chaque instance Beanstalk est facturée à l'heure. Celles-ci sont continuellement en marche et il y a sept instances pour desservir tous les microservices. Les coûts d'opération de l'application sont nettement supérieurs avec cette solution comparée au coût inférieur de *Lambda*.

Changement de la structure de la base de données

Certains changements ont été faits sur la base de données afin de permettre un accès plus facile et logique aux données. Tout d'abord, les tables de tâches ont été simplifiées. Un événement est associé à un groupe de tâches au lieu de faire l'intermédiaire entre les tâches pour se rendre aux groupes. L'objectif était de permettre aux événements de contenir plusieurs ensembles de tâches et de pouvoir effectuer un filtrage sur ces groupes. Ensuite, la table de suggestion a été jointe à la table des événements. Les suggestions affectent directement les événements et la table devrait être directement jointe sur celle des événements afin de permettre un suivi et une gestion plus adéquate. La table des votes a été associée à la table des membres afin de pouvoir faire un suivi plus facile des votes de chaque utilisateur sur les suggestions de plage horaire. De plus, la table d'état des votes a été remplacée par un champ d'état dans la table de votes. Cela permet de diminuer le nombre de tables nécessaire lors d'une requête sur un vote. Finalement, la table de jointure entre la table d'état des événements et celle des événements a été retirée. Cette table n'avait pas d'avantages particuliers et était rarement utilisée. Afin de faciliter la gestion de la base de données, celle-ci a été retirée.

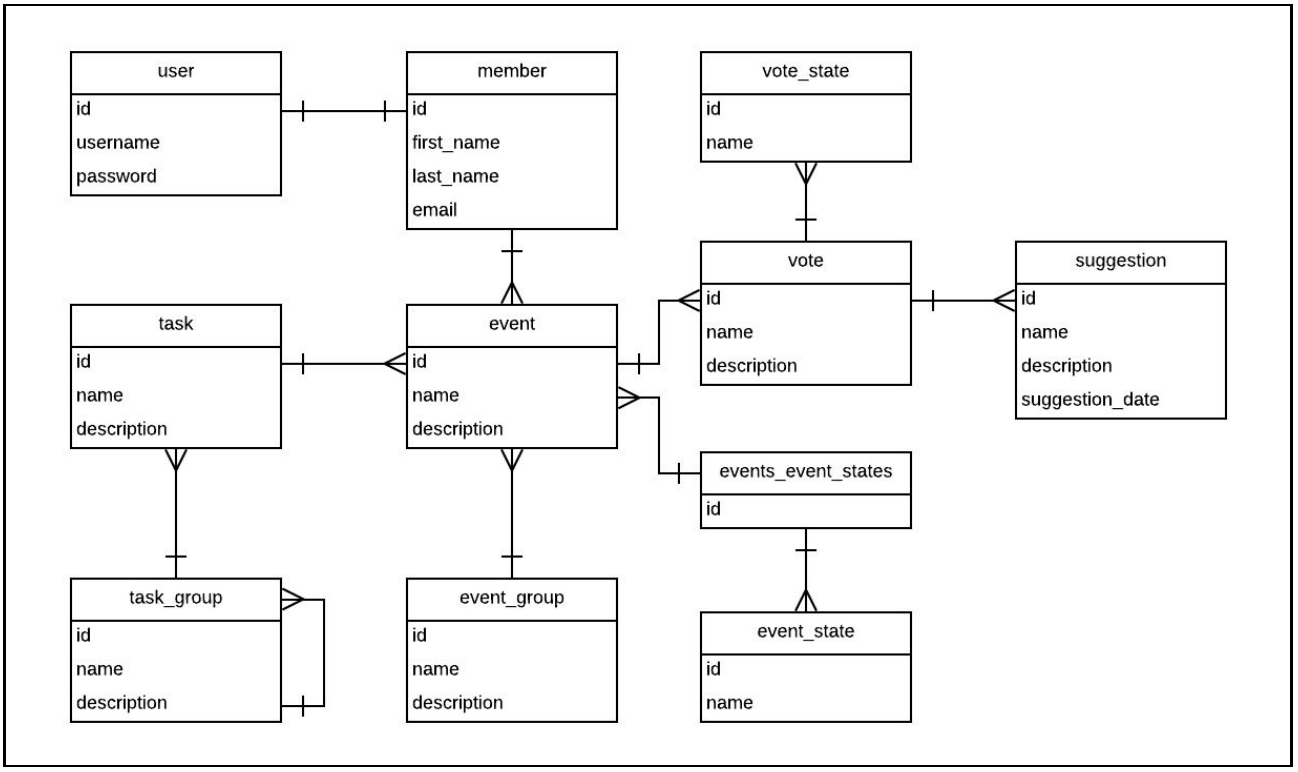


Figure 15 : Relations des tables de la base de données originale

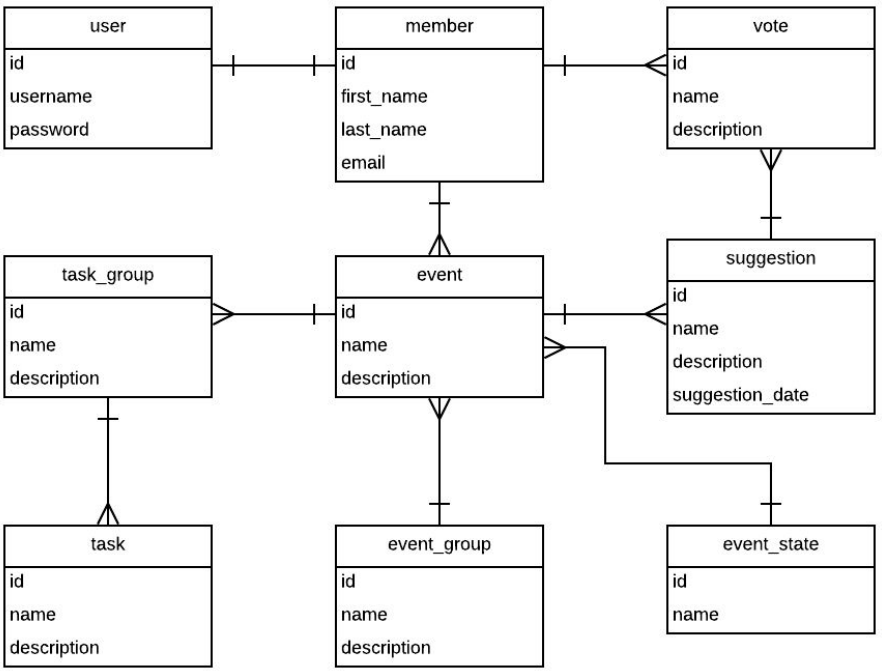


Figure 10 : Relations des tables de la base de données

PROCHAINE PHASE

Développement à continuer

ID	Description	Motivation
DC01	Changement du moteur des tables.	*Le moteur présent ne permet pas de créer des relations traditionnelles entre les tables.*
DC02	Implémentation du microservice *vote-service*.	*L'implémentation des opérations à base de données ainsi que des règles d'affaires relatives aux votes seront nécessaires à la complétion du projet.*
DC03	Modification de la structure de la base de données.	*Modification de la structure et des relations entre les tables pour accommoder les ajouts de fonctionnalités.*

Tableau 15 : Développement à continuer

Changements technologiques suggérés

ID	Description	Motivation
CT01	Migration vers *Gradle*	*Gradle offre une plus grande flexibilité, de meilleures performances et une utilisation plus facile.*
CT02	Hébergement *Lambda* (AWS)	*La technologie Lambda permet une plus grande flexibilité de gestion et est plus adaptée pour les applications basées sur des microservices.*
CT03	Plateforme *CloudFront* (AWS)	*Utilisation d'une plateforme d'hébergement de domaine afin d'offrir un service simple pour les utilisateurs.*
CT04	Gestion de sessions par *DynamoDB* (AWS)	*Utilisation d'une base de données DynamoDB à table unique pour enregistrer les informations de session des utilisateurs.*

Tableau 16 : Changements technologiques suggérés

Fonctionnalités suggérées

ID	Description	Motivation
FS01	Création de tâches	*La création de tâches est nécessaire afin de les attribuer à des événements.*
FS02	Suppression de tâches	*La suppression de tâches est nécessaire lorsqu'une tâche est inutile.*
FS03	Impression de calendrier	*L'utilisateur sera en mesure d'avoir une copie papier du calendrier.*
FS04	Envoi de courriel	*Cette fonctionnalité va permettre aux utilisateurs de ne pas avoir à utiliser un logiciel externe pour la communication.*
FS05	Notifications	*Cette fonctionnalité va permettre à l'utilisateur de recevoir des notifications lorsqu'une action effectuée requiert son attention.*
FS06	Internationalisation	*Cette fonctionnalité va permettre à l'utilisateur de choisir l'anglais ou le français selon ses préférences.*
FS07	Impression des tâches	*L'utilisateur sera en mesure d'imprimer la liste des tâches nécessaire à un événement.*

Tableau 17 : Fonctionnalités suggérées

CONCLUSION

L'objectif du projet qui consistait à poursuivre le cycle de développement de la première phase de l'application de gestion et planification des procès pour les avocats, connus sous le nom de Protogest, a été entièrement atteint.

L'analyse des pistes d'amélioration laissées par la première équipe qui a développé la solution a grandement orienté les décisions de démarrage de cette nouvelle phase ou tout simplement le nouveau produit qu'est Protogest 2.0. Une analyse de risque a été réalisée afin de mettre en perspective les menaces potentielles au projet ainsi que les moyens de contournements nécessaires menant aux succès. Pour ce faire, une méthodologie de travail a été mise en place incluant des processus de gestion des communications, gestions documentaires ou de fichiers et gestion des versions de codes accompagnés de *Scrum Meeting* hebdomadaires dans le but de produire des livrables de qualité dans les délais impartis.

Somme toute, nombreux ont été les défis à relever tout au long du développement. Compte tenu des difficultés rencontrées au niveau de la mise en place des infrastructures AWS, la migration du prototype vers *Bootstrap* du côté *Front-End*, la migration de la base de données et toutes les transformations dans la structure du *Back-End*, les participants se sont montrés productifs et le produit final est à la hauteur des attentes.

Protogest 2.0 s'inscrit dans le cadre d'une version améliorée de la version antérieure, mais non complète. Ceci laisse place à des améliorations au niveau des fonctionnalités existantes, l'ajout de nouvelles fonctionnalités ou des changements de technologies. Des suggestions d'amélioration sont documentées dans le rapport ainsi que leurs motivations afin de fournir une bonne base de démarrage utile aux équipes qui poursuivront le développement de ce produit.

RÉFÉRENCES

[1] wikipedia.org. (2018). *Slack*.
[En ligne] https://en.wikipedia.org/wiki/Slack_(software) [Consulté le 30 Nov. 2018].

[2] wikipedia.org. (2018). *Trello*.
[En ligne] https://en.wikipedia.org/wiki/Trello [Consulté le 30 Nov. 2018].

[3] wikipedia.org. (2018). *Google Drive*.
[En ligne] https://en.wikipedia.org/wiki/Google_Drive [Consulté le 30 Nov. 2018].

[4] wikipedia.org. (2018). *GitHub*.
[En ligne] https://en.wikipedia.org/wiki/GitHub [Consulté le 30 Nov. 2018].

[5] wikipedia.org. (2018). *Spring Boot*.
[En ligne] https://en.wikipedia.org/wiki/Spring_Framework [Consulté le 30 Nov. 2018].

[6] wikipedia.org. (2018). *Angular.*
[En ligne] https://en.wikipedia.org/wiki/Angular_(application_platform) [Consulté le 30 Nov. 2018].

[7] wikipedia.org. (2018). *Amazon Web Service*.
[En ligne] https://en.wikipedia.org/wiki/Amazon_Web_Services [Consulté le 30 Nov. 2018].

[8] wikipedia.org. (2018). *MySQL*.
[En ligne] https://en.wikipedia.org/wiki/MySQL [Consulté le 30 Nov. 2018].

[9] spring.io. (2018). *STS.*
[En ligne] https://spring.io/tools [Consulté le 30 Nov. 2018].

[10] wikipedia.org. (2018). *MySQL Workbench.*
[En ligne] https://fr.wikipedia.org/wiki/MySQL_Workbench [Consulté le 30 Nov. 2018].

[11] wikipedia.org. (2018). *Node Package Manager.*
[En ligne] https://en.wikipedia.org/wiki/Npm_(software) [Consulté le 30 Nov. 2018].

[12] wikipedia.org. (2018). *Visual Studio Code.*
[En ligne] https://en.wikipedia.org/wiki/Visual_Studio_Code [Consulté le 30 Nov. 2018].

[13] wikipedia.org. (2018). *Node Package Manager.*
[En ligne] https://en.wikipedia.org/wiki/Npm_(software) [Consulté le 30 Nov. 2018].

[14] wikipedia.org. (2018). *Swagger*.
[En ligne] https://en.wikipedia.org/wiki/Swagger_(software) [Consulté le 30 Nov. 2018].

[15] wikipedia.org (2018). *Gestion des risques.*
[En ligne] https://fr.wikipedia.org/wiki/Gestion_des_risques [Consulté le 07 déc. 2018]

[16] wikipedia.org (2018). *Évaluation des risques.*
[En ligne] https://fr.wikipedia.org/wiki/Évaluation_des_risques [Consulté le 07 déc. 2018]

[17] lescahiersdelinnovation.com (2016). *Les risques types d'un projet.*

[En ligne]
https://www.lescahiersdelinnovation.com/2016/01/les-risques-type-dun-projet-analyse-des-risques/ [Consulté le 07 déc. 2018].

[18] Projectlombok.org (2018). *Project Lombok.*
[En ligne] https://projectlombok.org [Consulté le 07 déc. 2018]

[20] wikipedia.org (2018). *Lucidchart.*
[En ligne] https://en.wikipedia.org/wiki/Lucidchart [Consulté le 12 déc. 2018].

[21] wikipedia.org (2018). *Apache Maven..*
[En ligne] https://en.wikipedia.org/wiki/Apache_Maven [Consulté le 13 déc. 2018].

[22] Amazon Web Services, Inc. (2018). *Create and Connect to a MySQL Database.*
[En ligne] https://aws.amazon.com/getting-started/tutorials/create-mysql-db/

[23] Ryan Zhou (2018). *Using Amazon Web Services Relational Database Service for your Spring Boot App.* [En ligne]
https://medium.com/@ryanzhou7/using-aws-rds-for-your-spring-boot-app-ca8f4b09c9b8

[24] StackOverflow (2017). *Changing embedded database in Spring Boot from H2 to MySQL.*
[En ligne]
https://stackoverflow.com/questions/45488742/changing-embedded-database-in-spring-boot-from-h2-to-mysql

[25] Spring.io. (2018). *Accessing data with MySQL.*
[En ligne] https://spring.io/guides/gs/accessing-data-mysql/#initial

[26] Serverless, Inc. (2018). *Serverless.yml Reference*
[En ligne] https://serverless.com/framework/docs/providers/aws/guide/serverless.yml

[27] Stefano Buliani. (2018). *Running APIs Written in Java on AWS Lambda*
[En ligne] https://aws.amazon.com/blogs/opensource/java-apis-aws-lambda

[28] Amazon Web Services, Inc. (2018). *Creating a .jar Deployment Package Using Maven without any IDE (Java).* [En ligne]
https://docs.aws.amazon.com/lambda/latest/dg/java-create-jar-pkg-maven-no-ide.html

[29] Apache Maven Project. (2012). *Skipping Tests.*
[En ligne]
http://maven.apache.org/plugins-archives/maven-surefire-plugin-2.12.4/examples/skipping-test.html

[30] Greg Emerick. (2018). *AWS Lambda with Spring Boot.*
[En ligne] https://keyholesoftware.com/2018/04/26/aws-lambda-with-spring-boot/

[31] Pascal Alma. (2016). *Making Spring Boot application run serverless with AWS*. [En ligne]
https://pragmaticintegrator.wordpress.com/2016/12/01/making-spring-boot-application-run-serverless-with-aws/

[32] Rowell Belen. (2017). *Serverless Microservices with Spring Boot and Spring Data.*

[En ligne]
https://www.rowellbelen.com/serverless-microservices-with-spring-boot-and-spring-data/

[33] Sergey Kryvets. (2018). *A complete guide for running Spring Boot MVC in the cloud using AWS Lambda.*
[En ligne] https://skryvets.com/blog/2018/06/02/spring-boot-aws-lambda-guide

[34] Sapessi*. (2018). *Quick start Spring Boot*. [En ligne]
https://github.com/awslabs/aws-serverless-java-container/wiki/Quick-start---Spring-Boot

[35] Ryan Zhou. (2018). *Running Spring Boot on Amazon Web Services for Free.* [En ligne]
https://medium.com/@ryanzhou7/running-spring-boot-on-amazon-web-services-for-free-f3b0aeec809

[36] Juan Villa. (2016). *Deploying a Spring Boot Application on AWS Using AWS Elastic Beanstalk.* [En ligne]
https://aws.amazon.com/blogs/devops/deploying-a-spring-boot-application-on-aws-using-aws-elastic-beanstalk/

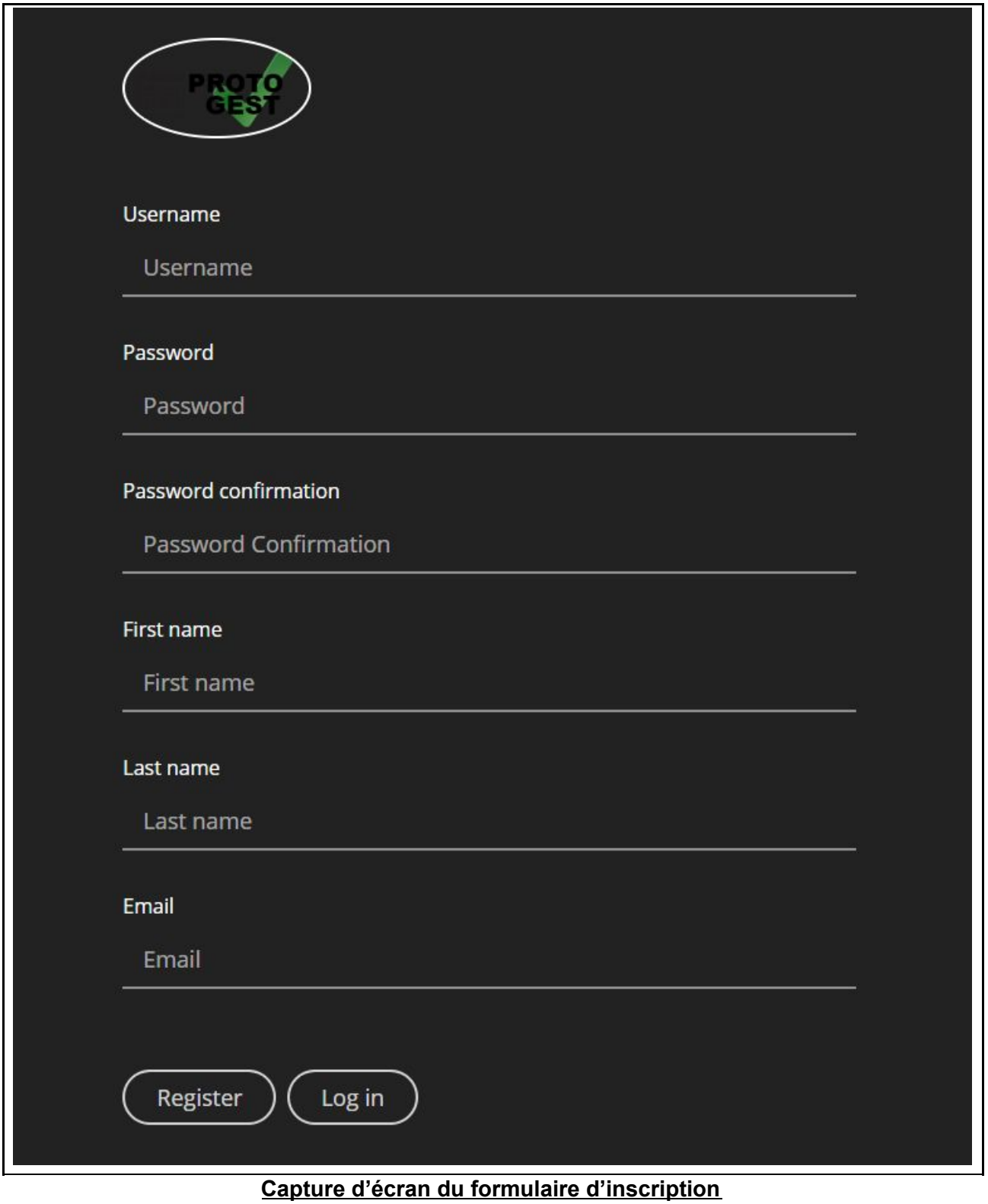
[37] Kittiphat Srilomsak. (2018). *Deploy an Angular with S3 and CloudFront*. [En ligne]
https://medium.com/@peatiscoding/here-is-how-easy-it-is-to-deploy-an-angular-spa-single-page-app-as-a-static-website-using-s3-and-6aa446db38ef

[38] Angular official site. (2018). Getting Started. [En ligne]
https://angular.io/guide/quickstart

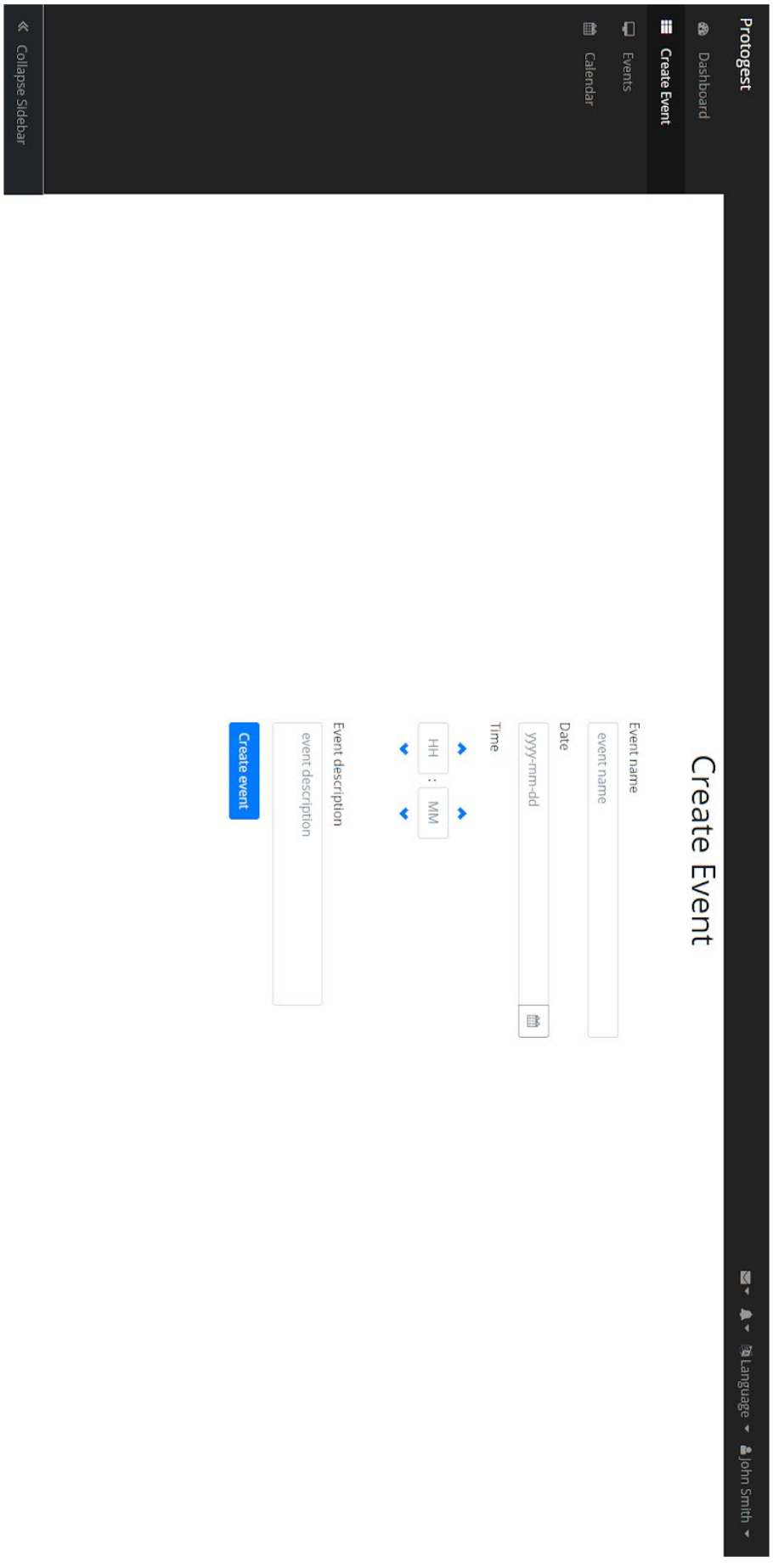
ANNEXES

ANNEXE 01 - Captures d’écran du *Front-End*

Capture d’écran du formulaire d’authentification

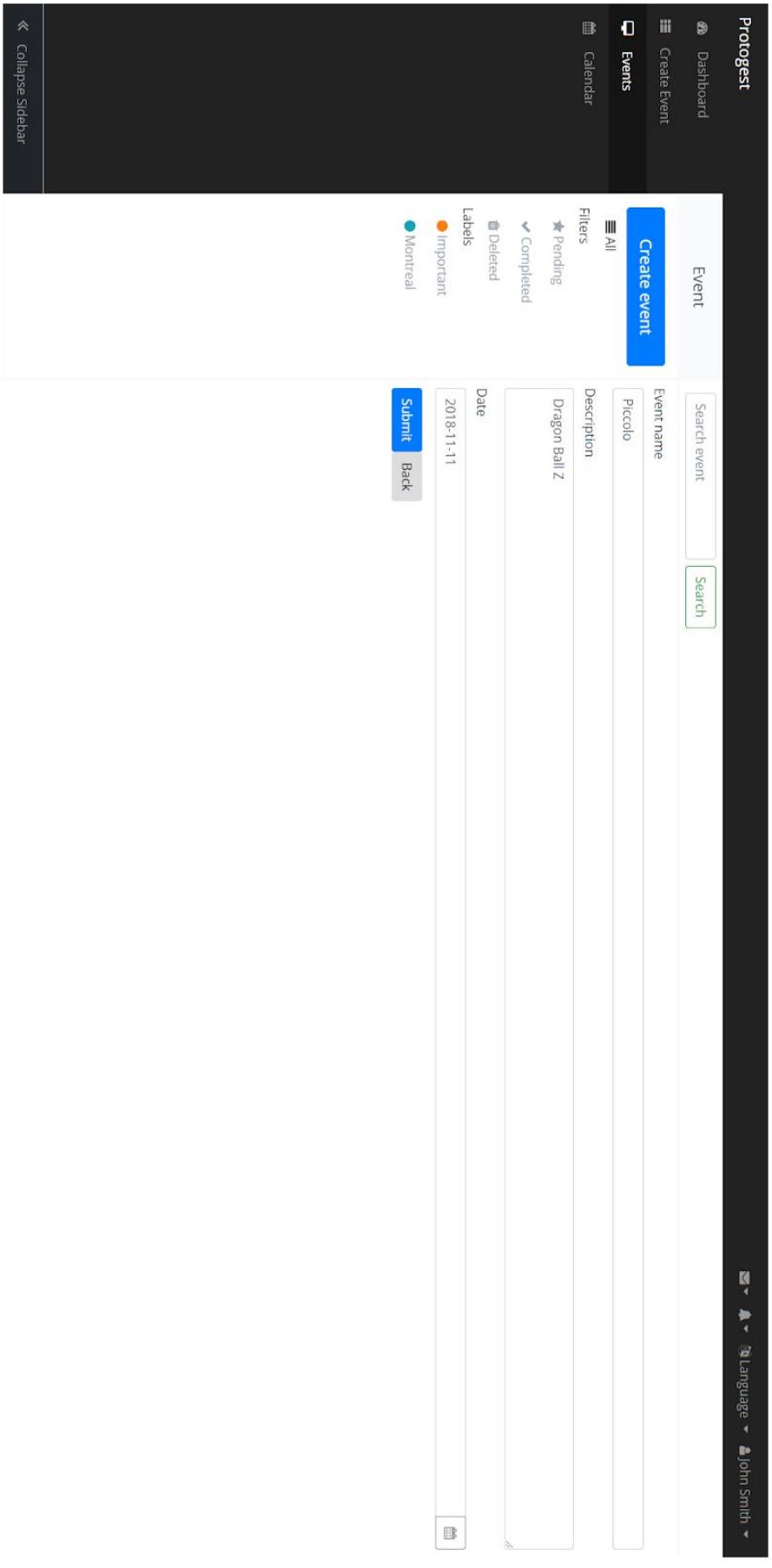


Capture d’écran du formulaire d’inscription

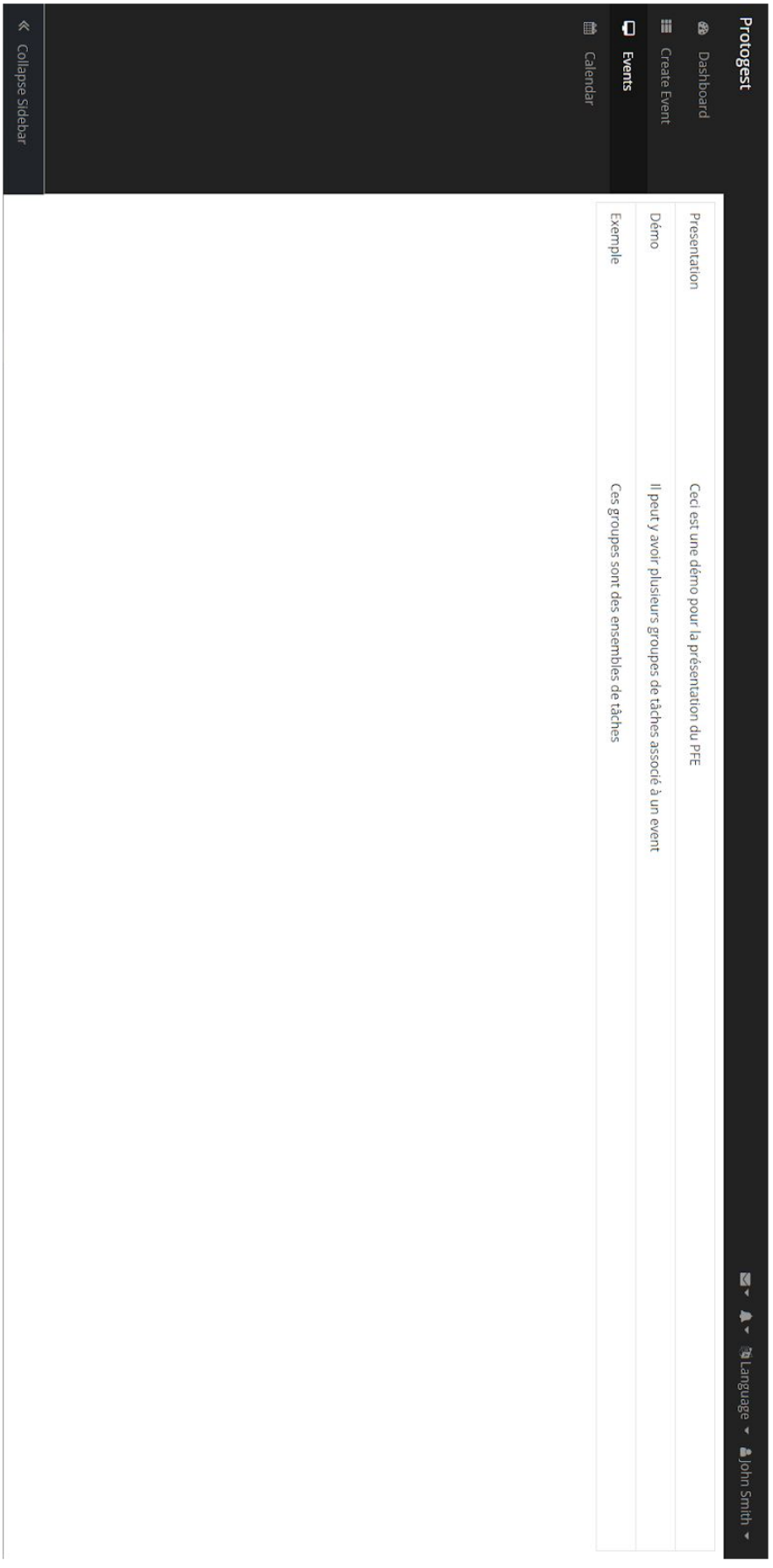


Capture d'écran du formulaire de création d'un événement

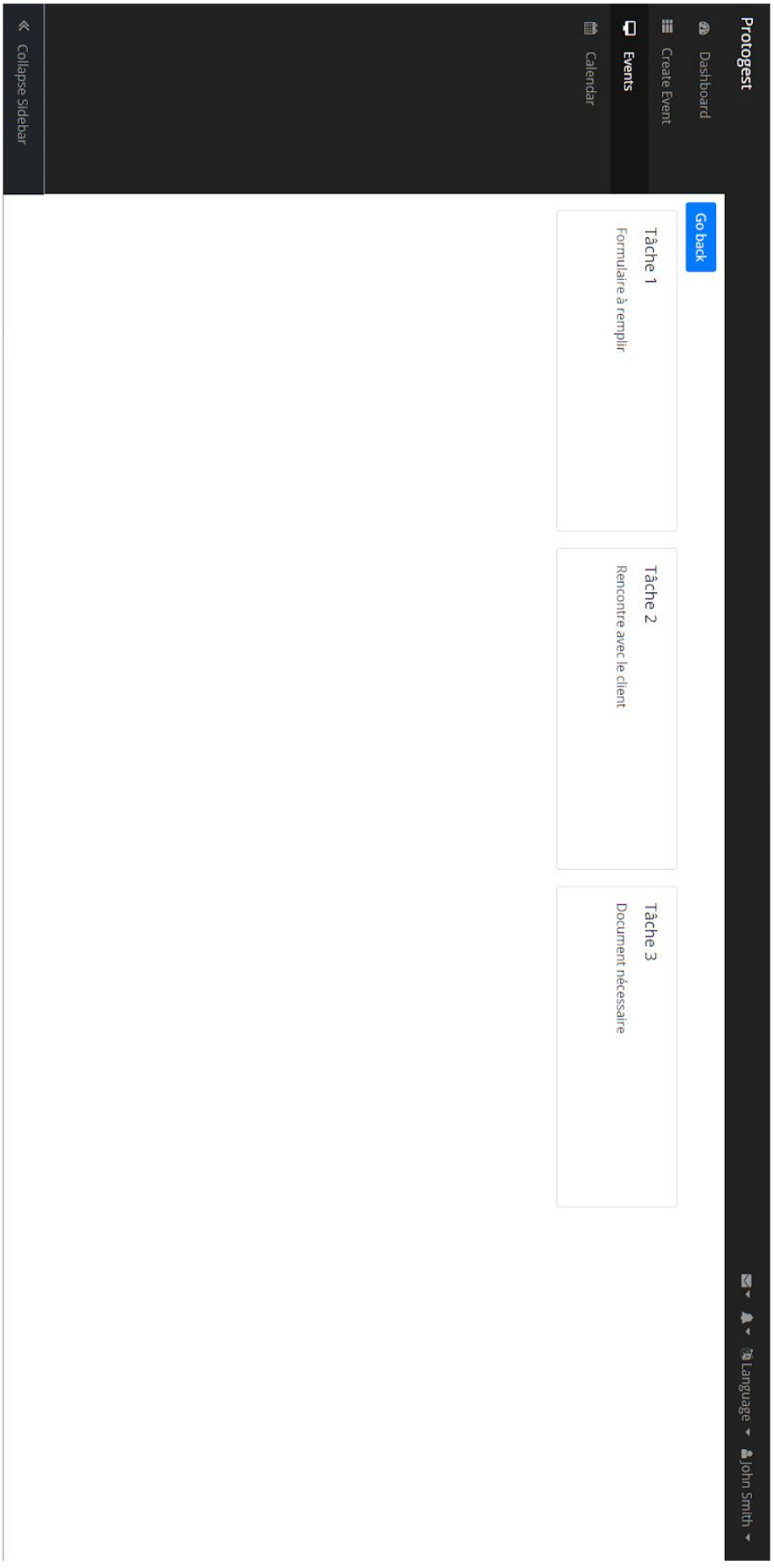
Capture d'écran de la page d'affichage d'événements



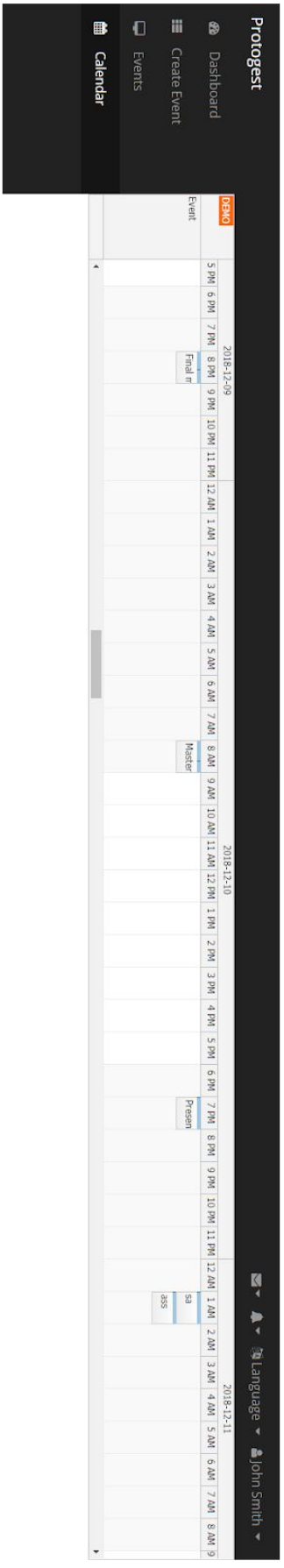
Capture d'écran du formulaire de mise à jour d'un événement



Capture d’écran de la page d’affichage des regroupements de tâches



Capture d'écran de la page d'affichage de tâches appartenant à un regroupement



Capture d'écran du calendrier d'événements

ANNEXE 02 - Métriques de l'analyse des risques

Voici la description des différents degrés des métriques nécessaires à l'analyse des risques du projet.

Degré	**1**	**2**	**3**	**4**	**5**
Probabilité	Impossible	Improbable	Possible	Probable	Certitude

Degré	**1**	**2**	**3**	**4**	**5**
Conséquence	Aucune	Faible	Moyenne	Sévère	Irrécupérable

Degré	**(0, 2]**	**(2, 6]**	**(6, 12]**	**(12, 20]**	**(20, 25]**
Risque	Acceptable	Bénin	Normal	Grave	Inacceptable

ANNEXE 03 - Métriques de l'analyse des efforts

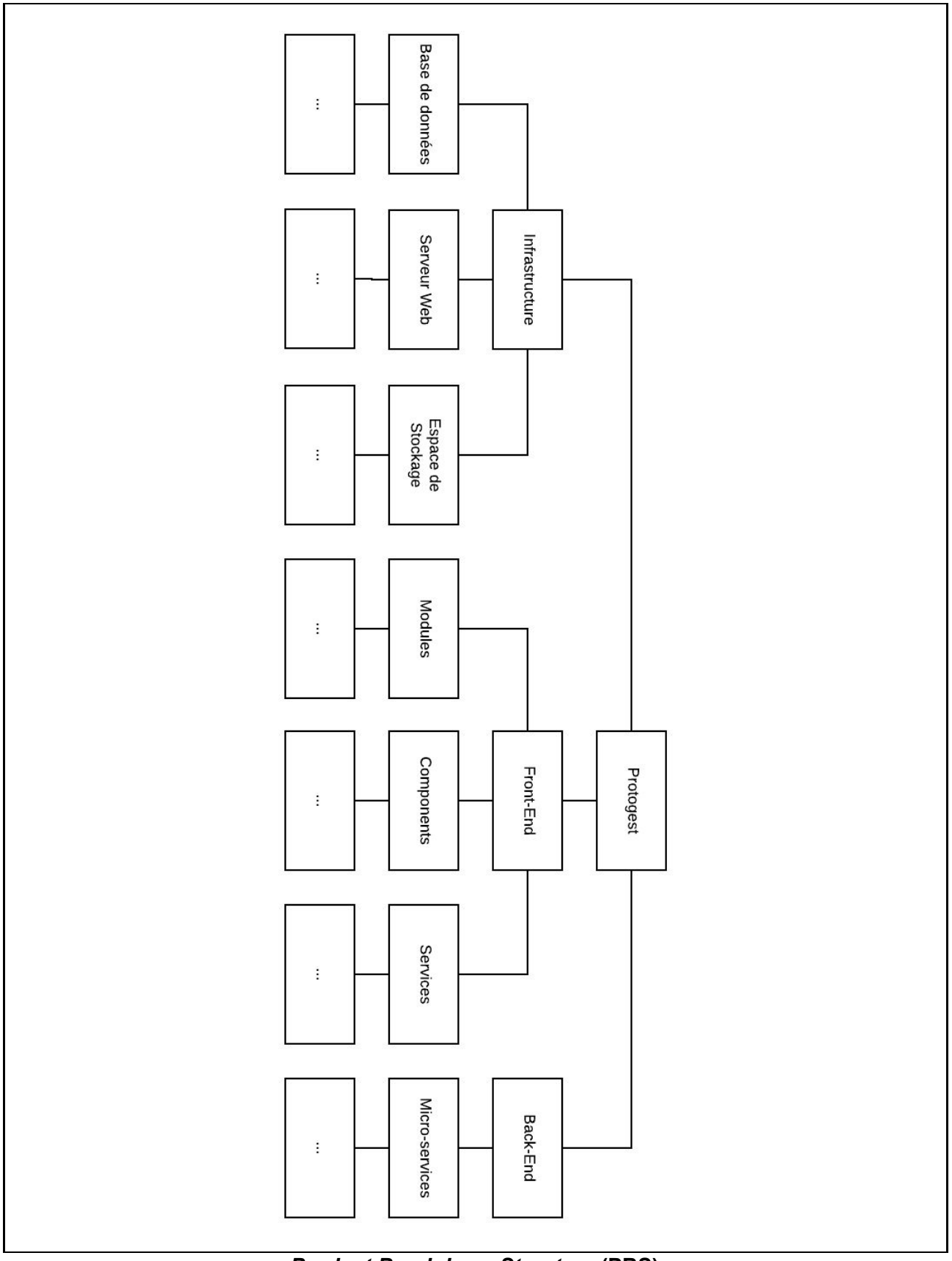
Voici la description des différents degrés des métriques nécessaires à l'analyse des efforts du projet.

Degré	1	2	3	4	5
Difficulté	Aucune	Basse	Moyenne	Haute	Extrême

Degré	1	2	3	4	5
Priorité	Aucune	Basse	Moyenne	Haute	Extrême

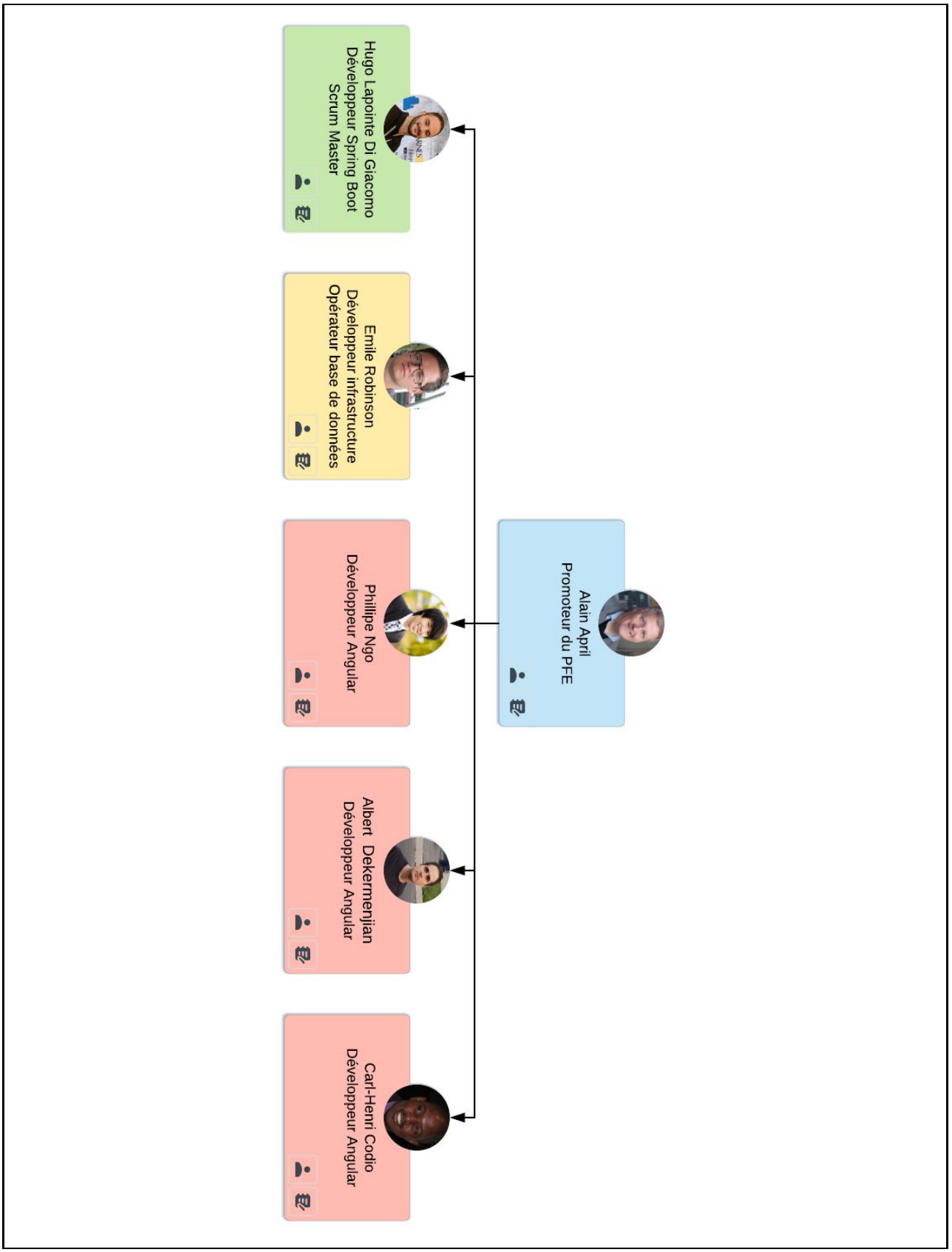
Degré	1	2	3	4	5
Effort	0 heure	25 heures	50 heures	75 heures	100 heures

ANNEXE 04 - *Product Breakdown Structure* (PBS)



***Product Breakdown Structure* (PBS)**

ANNEXE 05 - *Organization Breakdown Structure* (OBS)



***Organization Breakdown Structure* (OBS)**

ANNEXE 06 - *Work Breakdown Structure* (WBS)

Voici une liste exhaustive des tâches du projet. Cette liste fut générée suite à l'exportation des cartes Trello. À noter que les autres éléments associés aux cartes, tels que les commentaires, les membres, la date de création, ont été retirés afin de simplifier l'affichage de la liste.

Catégorie	Titre
Front-End, Back-End	*Maîtriser l'application fonctionnelle en se référant au document de vision.*
Front-End	*Trouver un gabarit pour le Front-End.*
Front-End, Back-End	*Installer et tester la version actuelle de Protogest.*
Infrastructure	*Identifier les prochaines étapes pour l'élaboration de l'infrastructure.*
Front-End, Back-End	*Tous les membres devraient être capable d'accéder aux répertoires GitHub du Front-End et du Back-End.*
Gestion de projet	*Tous les membres devraient être en mesure d'accéder à Trello et avoir les permissions suffisantes.*
Back-End	*Mettre à jour la documentation du fichier README du répertoire GitHub du Back-End.*
Gestion de projet	*Communiquer avec Walid pour planifier une rencontre.*
Gestion de projet	*Créer une feuille Excel pour prendre en note les heures au projet.*
Front-End	*Se familiariser avec Angular.*
Infrastructure	*Chercher des tutoriels sur AWS Lambda et son intégration avec React*
Back-End	*Ajouter les dépendances du Back-End au fichier de configuration yarn.*
Infrastructure	*Se familiariser avec Lambda de AWS.*
Infrastructure, Back-End	*Remplacer le database-Service H2 par MySQL.*
Front-End	*Script NPM pour le Front-End*
Front-End	*Installation de l'environnement de développement pour le Front-End*
Back-End	*Intégration des DTO dans Task-Service et restructuration du service.*
Back-End	*Intégration de Lombok et transformation des classes.*
Front-End	*Découper et préparer la conversion de REACT/REDUX vers Angular.*

Gestion de projet	*Mise en place de la structure du projet dans l'environnement de développement*
Back-End	*Intégration de Swagger 2.0 au sein des microservices afin de faciliter le développement entre Back-End et Front-End.*
Base de données	*Convertir la base de données H2 à MySQL.*
Back-End	*Compléter le microservice Task-Service afin qu'il implémente toutestes les opérations CRUD.*
Front-End	*Coder le app component*
Front-End	*Configurer les routes*
Back-End	*Intégration des DTO dans Event-Service et restructuration du service.*
Front-End	*Coder le event component*
Back-End	*Restructuration du microservice auth-service.*
Gestion de projet	*Création du rapport de stage.*
Back-End	*Implémenter la méthode getAllEventFromCurrentUser() au sein du microservice event-service..*
Infrastructure, Back-End	*Tester et partager la nouvelle base de données dans le Back-End.*
Gestion de projet	*Trouver une solution pour notre prochaine rencontre du 02 novembre 2018.*
Back-End	*Corriger les problèmes de dépendances du projet Back-End.*
Back-End	*Extraire le microservice auth-service du service member-service.*
Gestion de projet	*Obtenir des informations concernant le rapport à remettre et débuter sa rédaction.*
Back-End	*Restructuration du microservice member-service.*
Front-End	*Implémenter le EventsService*
Front-End	*Conversion du template dans le projet Angular*
Infrastructure	*Familiariser avec Docker avant rencontre avec Mathieu*
Infrastructure	*Discuter avec Mathieu concernant le démarrage de container.*
Gestion de projet	*Créer un diaporama pour la présentation du PFE.*
Front-End	*Coder le calendar component*

Front-End	*Coder le login component*
Gestion de projet	*Rapport - Diagramme (Back-End).*
Front-End	*Coder le create-event component*
Back-End	*[BOGUE] EventStateId ne devrait pas être requis pour créer un event.*
Front-End	*Coder le signup component*
Back-End	*[BOGUE] Un event n'est pas supprimé par l'opération DELETE : /event/{id}.*
Infrastructure	*Transposer Spring Boot dans AWS.*
Gestion de projet	*Rapport - Présentation des technologies (Spring Boot).*
Gestion de projet	*Rapport - Présentation des technologies (Angular).*
Front-End	*Coder le task component.*
Back-End	*[BOGUE] Corriger l'inscription d'un User.*
Base de données	*[IMPROVE] Change la relation Suggestion/Vote/Member/Event.*
Back-End	*Mise à jour des contrôleurs afin de refléter la mise à jour de la BD.*
Infrastructure, Back-End	*[IMPROVE] Retirer le projet entity-management et distribuer son contenu au sein des autres services.*
Base de données	*Remplir la base de données d'exemples de données afin de faciliter les tests.*
Back-End	*[BOGUE] Ajouter de la validation lors de l'ajouter ou modification d'éléments.*
Back-End	*[BOGUE] Impossible de supprimer un User.*
Base de données	*[IMPROVE] Inverser le lien entre User et Member.*
Back-End	*[IMPROVE] Ajouter la validation des clés étrangères lors de l'ajouter d'un entité.*
Base de données	*[IMPROVE] Changer le lien entre Event, TaskGroup et Task.*
Gestion de projet, Infrastructure	*Poser la question à Alain : si on ne réussit pas à implémenter Lambda, devrons-nous héberger le projet normalement?*
Gestion de projet	*Rapport - Diagramme (BD).*
Back-End	*Implémenter la sécurité des points de connexion de chaque microservice.*
Back-End	*Implémenter des tests unitaires pour chaque contrôleur des microservices.*

Front-End	*Planifier une rencontre avec David Lozon pour en apprendre davantage sur REACT.*
Front-End	*Coder le reset-password component*
Front-End	*Coder le task component avec add et delete function.*
Gestion de projet	*Rapport - Conclusion.*
Gestion de projet	*Rapport - Résumé*
Back-End	*Implémentation du microservice vote-Service.*
Infrastructure	*Analyser la virtualisation de du projet.*
Back-End	*Intégration des DTO dans Member-Service et restructuration du service.*
Gestion de projet	*Rapport - Analyse du projet précédent.*
Back-End	*Intégrer ModelMapper pour mapper les DTO avec les entités.*
Gestion de projet	*Rapport - Conceptions, Corrections et Intégrations.*
Gestion de projet	*Rapport - Objectifs*
Gestion de projet	*Rapport - Diagramme (Infrastructure).*
Gestion de projet	*Rapport - Présentation des technologies (AWS).*
Gestion de projet	*Rapport - Difficultés rencontrées.*
Gestion de projet	*Rapport - Annexe (PBS/WBS/OBS).*
Gestion de projet	*Rapport - Références*
Front-End	*Coder le requests component*
Gestion de projet	*Rapport - Sitemap.*
Gestion de projet	*Rapport - Remerciements*
Gestion de projet	*Rapport - Diagramme (Front-End).*
Gestion de projet	*Rapport - Méthodologie de travail*

ANNEXE 07 - Heures de travail

Hugo Lapointe Di Giacomo

#	D	F	DESCRIPTION	HRS	NB	CUM
1	07/09	14/09	- Lecture du document de vision. - Gestion de projet.	1.5 0.5	2	2
2	14/09	21/09	- Installation et configuration de l'environnement de développement. - Gestion de projet.	2.0 1.0	2	3
3	21/09	28/09	- Installation et configuration de l'environnement de développement. - Gestion de projet.	1.0 0.5	2	1.5
4	28/09	05/10	- Configuration de Yarn pour l'installation du Front-End. - Analyse de la documentation. - Gestion de projet.	1.0 2.0 0.5	3	3.5
5	05/10	12/10	- Familiarisation avec l'implémentation de ModelMapper. - Débute l'implémentation de ModelMapper. - Gestion de projet.	3.0 2.3 0.5	3	5.8
6	12/10	19/10	- Création des DTO, Facade et Service pour Task-Service. - Restructuration des différents services : Task-Service et Event-Service. - Intégration de lombok et restructuration des entités. - Lecture sur OAuth 2.0 et la sécurité d'une application Spring avec microservices et sur l'utilisation de DTO au sein des différents contrôleurs. - Gestion de projet.	3.8 3.0 1.0 2.0 0.5	5	10.3
7	19/10	26/10	- Complétion des opérations du microservice task-service. - Intégration de Swagger auprès des micros-services event-service et task-service. - Restructuration du microservice event-service. - Gestion de projet.	3.0 2.5 3.5 0.5	4	9.5
8	26/10	02/11	- Extraction du microservice auth-service du microservice member-service. - Restructuration du microservice member-service. - Gestion de projet.	4.0 4.3 0.5	3	8.8
9	02/11	09/11	- Création du rapport d'achèvement. - Implémentation d'opérations au sein du microservice event-service. - Gestion de projet.	1.0 1.5 0.3	3	2.8
10	09/11	16/11	- Restructuration du microservice auth-service. - Test et intégration de la base de données MySQL. - Gestion de projet.	5.3 4.3 1.0	3	10.5
11	16/11	23/11	- Corrections de bogues. - Rédaction du rapport. - Gestion de projet.	3.5 2.0 1.0	3	6.5

12	23/11	30/11	- Corrigé auth-service. Mise à jour de la BD. - Correction de task-service, event-service et member-service. - Retirer le projet "entity-management" pour permettre le déploiement des autres services de façon indépendante. - Corrections de bogues et refactoring. - Rédaction du rapport. - Gestion de projet.	5.8 5.5 3.0 5.5 2.3 1.0	6	23
13	30/11	07/12	- Implémentation du vote-service. - Corrections aux différents services. - Création du diaporama pour la présentation. - Préparation de la présentation - Rédaction du rapport. - Gestion de projet	0.0 0.3 1.0 4.0 9.0 6.0	6	20.3
14	07/12	14/12	- Rédaction du rapport. - Révision du rapport. - Gestion de projet	4.0 4.0 2.0	3	10
15	14/12	21/12	- Rédaction du rapport. - Révision du rapport. - Gestion de projet	6.0 12.0 1.0	3	19
					51.0	**136.3**

Philippe Ngo

#	D	F	DESCRIPTION	HRS	NB	CUM
1	07/09	14/09	- Lecture du document de vision. - Scrum meeting	1.0 1.0	3	2.0
2	14/09	21/09	- Installation et configuration de l'environnement de développement. - Scrum meeting	2.0 1.0	3	3.0
3	21/09	28/09	- Installation et configuration de l'environnement de développement. - Scrum meeting	2.0 1.0	3	3.0
4	28/09	05/10	- Maîtriser l'application fonctionnel en se référant au document de vision. - Analyser les choix technologiques (Angular, React Native, ReactJS) - Scrum meeting	2.0 6.0 1.0	3	9.0
5	05/10	12/10	- Découper et préparer la conversion de REACT/REDUX vers Angular. - Scrum meeting	5.0 1.0	3	6.0
6	12/10	19/10	- Mise en place de l'environnement de développement en Angular 6 - Mise en place de la structure du projet dans l'environnement de développement - Scrum meeting - Codage du routing de base	2.0 3.0 1.0 2.0	5	10.0

			- Découper et préparer la conversion de REACT/REDUX vers Angular.	2.0		
7	19/10	26/10	- Coder le app component - Configurer les routes - Scrum meeting	5.0 0.5 1.0	3	6.5
8	26/10	02/11	- Code review - Event Service - Scrum meeting	1.0 5.0 1.0	3	7.0
9	02/11	09/11	- Event component - Config du backend avec lombok - Scrum meeting	8.0 2.0 1.0	3	11.0
10	09/11	16/11	- Event component - Scrum meeting	9.0 1.0	3	10.0
11	16/11	23/11	- CORS error in localhost - Event component (bug fixes and connection to the api) - Scrum meeting - Rédaction du rapport (Technologie Angular)	1.0 3.0 1.0 1.0	4	6.0
12	23/11	30/11	- Task component - Task service - Rapport méthodologie de travail - Scrum meeting - Refactoring	4.0 2.0 2.0 1.0 1.0	5	10.0
13	30/11	07/12	- Code review - Rapport - Présentation/Démo - Scrum meeting - Connexion debugging - Calendar component - Code merging - Create Task component	9.0 2.0 8.0 1.0 1.0 1.0 2.0 1.0	8	25.0
14	12/07	12/14	-	0.0	0	0.0
15	12/14	12/21	- Rédaction rapport	7	1	7
					50	115.5

Carl-Henri Codio

#	D	F	DESCRIPTION	HRS	NB	CUM
1	07/09	14/09	- Prendre connaissance du document de vision - Scrum meeting	1.0 1.0	2	2.0
2	14/09	21/09	- Installation et configuration de l'environnement de développement de l'ancienne version de l'application - Scrum meeting	2.0 1.0	2	3.0

3	21/09	28/09	- Installation et configuration de l'environnement de développement de l'ancienne version de l'application - Analyse du Back-End et Front-End de l'ancienne application - Scrum meeting	4.0 4.0 1.0	3	9.0
4	28/09	05/10	- Analyse d'impact de la migration du Front-End - Recherche sur les possibilités de migration de la BD H2 - Scrum meeting	3.0 2.0 1.0	3	6.0
5	05/10	12/10	- Mise en place de l'environnement de développement Angular 6 pour le développement du Front-End - Scrum meeting	2.0 1.0	2	3.0
6	12/10	19/10	- Choisir un UI pour le Front-End - Valider l'intégration du UI avec l'application - Scrum meeting	1.0 3.0 1.0	3	5.0
7	19/10	26/10	- Valider l'integration du UI (Front-End) avec la nouvelle structure de l'application - Scrum meeting	3.0 1.0	2	4.0
8	26/10	02/11	- Conversion template Front-End - Scrum meeting	6.0 1.0	2	7.0
9	02/11	09/11			0	0.0
10	09/11	16/11	- Calendar component (Front-End) - Test d'integration calendar FullCalendar - Scrum meeting	8.0 4.0 1.0	3	13.0
11	16/11	23/11	- Configurer la nouvelle version du Back-End - Redaction rapport (Outil Visual Code) - Redaction rapport (Ajout NPM et Swagger) - Redaction rapport (Ajout - Technologie Angular) - Redaction rapport (Technologie AWS et MySQL)	3.0 0.5 0.5 0.5 0.5	5	5.0
12	23/11	30/11	- Connexion component - Connexion service - Rapport diagramme du Front-End - Validation formulaire connexion - Test d'integration calendar DayPilot - scrum meeting	5.0 3.0 0.5 4.0 4.0 1.0	6	17.5
13	30/11	07/12	- Correction de la méthode POST du login - Test d'integration calendar (DHTMLX Scheduler) - Correction calendar component et service (DayPilot) - Rapport/Présentation - scrum meeting	1.0 8.0 6.0 8.0 1.0	5	24.0
14	07/12	14/12				
15	14/12	21/12	- Redaction du rapport - Revision du rapport	8 4	2	

					40	**110.5**

Albert Dekermenjian

#	D	F	DESCRIPTION	HRS	NB	CUM
1	07/09	14/09	- Lecture du document de vision - scrum meeting	1.0 1.0	2	2.0
2	14/09	21/09	- Installation et configuration de l'environnement de développement - scrum meeting	2.0 1.0	2	3.0
3	21/09	28/09	- Installation et configuration de l'environnement de développement - Apprentissage React - scrum meeting	2.0 2.0 1.0	3	5.0
4	28/09	05/10	- Mises à jour de la documentation - Apprentissage React - scrum meeting - Analyse du frontend de l'ancienne application	2.0 2.0 1.0 1.0	4	6.0
5	05/10	12/10	- Apprentissage Angular - scrum meeting	2.0 1.0	2	3.0
6	12/10	19/10	- Apprentissage Angular - Préparation environnement de développement - Scrum meeting	5.0 1.0 1.0	3	7.0
7	19/10	26/10	- Import de la branche de développement - Exploration du code - Scrum meeting - Essai de l'intégration des dropdown	1.0 1.0 1.0 1.0	4	4.0
8	26/10	02/11	- Design du template - Scrum meeting	4.0 1.0	2	5.0
9	02/11	09/11	- Coder create-event component - Scrum meeting	3.0 1.0	2	4.0
10	09/11	16/11	- Reset-Password UI - Ajout bootstrap Create event - scrum meeting - Recherche des données à envoyer	1.0 1.0 1.0 2.0	4	5.0
11	16/11	23/11	- Coder signup - Coder create-event - scrum meeting	6.0 3.0 1.0	3	10.0
12	23/11	30/11	- Coder create-event - scrum meeting	9.0 1.0	2	10.0
13	30/11	09/12	- Coder signup - Rédiger rapport - Préparer présentation	8.0 3.0 3.5	5	26.5

			- Test Login	1.0		
			- Coder create-event	11.0		
14	09/12	19/12	- Rédiger rapport	7	1	7
					38	**97.5**

Emile Robinson

#	D	F	DESCRIPTION	HRS	NB	CUM
1	07/09	14/09	- Meeting scrum	1.0	1	1.0
2	14/09	21/09	- Meeting scrum	1.0	1	1.0
3	21/09	28/09	- Installation de l'application pour test - Lecture et compréhension du document de vision - Meeting scrum	2.0 1.0 1.0	3	4.0
4	28/09	05/10	- Recherche et exécution de tutoriels sur AWS - Meeting scrum	3.0 1.0	2	4.0
5	05/10	12/10	- Recherche et exécution de tutoriels sur AWS - Lecture et analyse de la conversion H2 à MySQL - Création de base de données MySQL dans AWS	2.0 1.0 1.0	3	4.0
6	12/10	19/10	- Tutoriels et documentation AWS - Conversion BD de H2 à MySQL - scrum meeting	5.0 3.0 1.0	3	9.0
7	19/10	26/10	- Mise en place environnement git et upload de bd service - Test de fonctionnalité avec nouvelle BD - scrum meeting	2.0 1.0 1.0	3	4.0
8	26/10	02/11	- Test d'intégration nouvelle BD et débug - scrum meeting	2.0 1.0	2	3.0
9	02/11	09/11	- Rencontre avec Mathieu pour apprentissage AWS - Modification pour accès BD et debug 2.0 - scrum meeting	1.5 4.0 1.0	3	6.5
10	09/11	16/11	- Debug 2.0 pour accès bd pour Hugo - Debug auth service pour utilisation de 2.0	3.0 2.0	2	5.0
11	16/11	23/11	- Recherche et lecture des articles sur Aws et spring Test de conversion des microservices et troubleshooting	1.0 5.0	2	6.0
12	23/11	30/11	- Conversion des microservices et troubleshooting - Rédaction rapport (diag infra)	11.0 1.0	2	12.0
13	30/11	07/12	- Conversion des microservices et troubleshooting - Préparation de la présentation et pratique - Rédaction de rapport	20.0 4.5 8.0	3	32.5
					30	**92.0**