Faculté des sciences et technologies, Nancy
Université de Lorraine
Master informatique - spécialité Ingénierie des Logiciels

Promo 2024 - 2025

Gestion de projet
EduPLanner
Rapport partie technique

Rapport réalisé par
Madison NOYER, Théo FAEDO, Romain POLKOWSKI, Naoufal EL-AMRY, Mohammad Wael SABBAGH, Ismail TAGMOUTI, Ghalia LAHLOU, Aboubacar HASSANE CHEKOU KORE

11 mars 2025

1. Avant propos

Ce document a pour but de rapporter la description des différentes phases du projet EduPlanner, une application ayant pour but la gestion des affectations des enseignants. Nous montrerons les différentes phases du projet en exposant les choix techniques du projet, ainsi que les défis rencontrés.

2. Plan du rapport

3. Technologies / Outils utilisées

Pour pouvoir produire une application, le choix des technologies et des outils est crucial, puisque cela agit directement sur l'efficacité du travail et donc sur le produit final. Dans cette partie nous allons lister nos différents choix.

a. Technologies

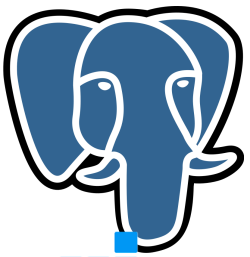
Angular

Pour réaliser la partie front-end de notre application, nous avons choisi d'utiliser le framework Angular. Nous avons choisi ce framework car celui-ci est l'un des plus utilisés aujourd'hui ce qui montre déjà un gage de qualité, mais également parce que c'est celui-ci qui nous a été enseigné durant un UE de cette année. Il semblait donc judicieux de l'utiliser car on savait que tout le monde le maîtrisait un minimum.

Spring-boot

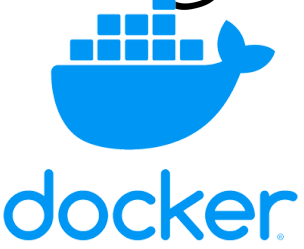
La création de la partie back-end de notre application a été permise par l'utilisation du framework Spring Boot. Nous avons choisi ce framework pour les mêmes raisons que le précédent, c'est-à-dire, car il est très utilisé et que toute l'équipe en connaît un minimum sur celui-ci. Une autre raison est aussi le fait que notre formation est énormément basée sur l'apprentissage du Java et spring-boot étant un framework Java, on avait sûrement plus de maîtrise dans celui-ci.

PostgreSQL



Nous avons choisi PostgreSQL comme système de gestion de base de données, puisqu'il a un fonctionnement ressemblant énormément à Oracle qui est un des systèmes de base de données pour lequel nous avons été le plus formés.

Docker



Lorsqu'on réalise une application, il peut être utile de pouvoir la déployer sur un maximum de configurations de machines différentes (assurer la compatibilité). Mais également d'assurer une chaîne de déploiement. C'est pour cette raison que nous avons choisi de conteneuriser notre application à l'aide de Docker qui est l'un des plus utilisés et des plus documentés.

b. Outils

Planner

La méthodologie SCRUM demande une organisation spécifique utilisant des cartes qu'on doit déplacer sur divers plans. Pour appliquer cette méthode, il existe de nombreux outils. Nous avons choisi d'utiliser Planner car certains savaient déjà s'en servir, on allait donc pouvoir l'utiliser de la meilleure des façons.

Discord

Durant les différents sprints de ce projet, nous n'étions pas dans un environnement traditionnel en entreprise, de ce fait, nous avons dû utiliser un outil pour pouvoir communiquer à distance. Nous avons choisi sans aucune hésitation Discord puisque nous avions tous déjà un compte sur ce réseau et qu'il permet d'organiser facilement les discussions via son système de channels.

GitHub

Lorsque l'on développe une application en équipe, l'utilisation d'un gestionnaire de version est obligatoire. Nous avons décidé d'utiliser Git puisque tout le monde le maîtrisait dans l'équipe. Et nous avons choisi comme plateforme GitHub pour les mêmes raisons.

4. Méthodologie de développement

Pour le développement de l'application, nous avons adopté le cadre SCRUM qui est un type de méthode Agile. L'organisation dans ce cadre se fait en sprint d'une durée de 1 mois en général. Durant le développement, nous avons appris au fur et à mesure à travailler dans ce cadre-là.

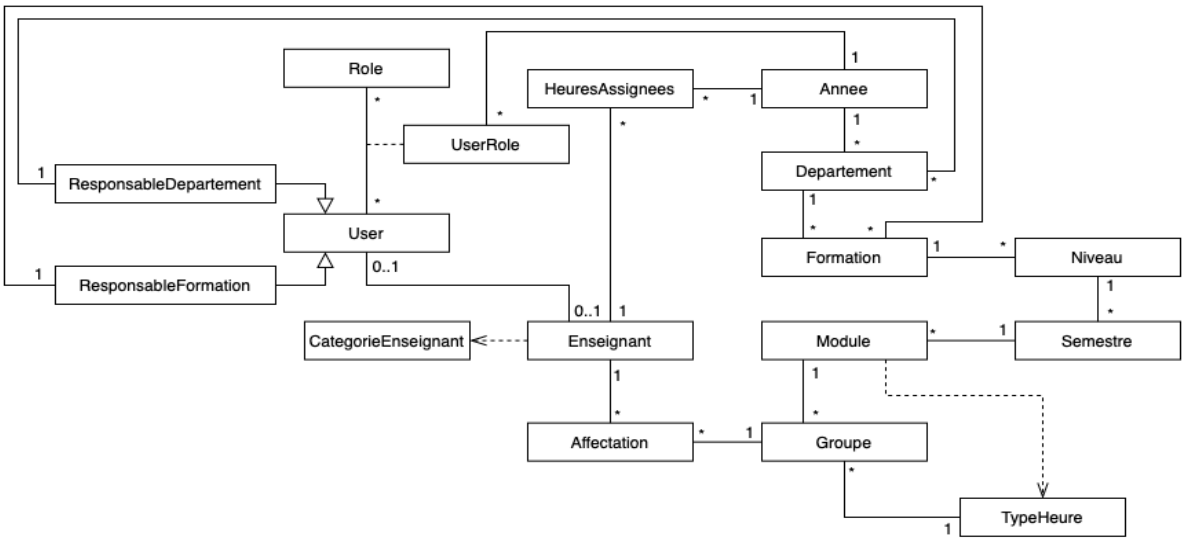
Nous avons pu adopter de plus en plus de principes et de méthodes et avons remarqué une amélioration de l'efficacité. Par exemple, nous n'avions pas de réunions régulières lors des premiers sprints et avons commencé à en faire au quatrième sprint. Nous avons alors remarqué une grande augmentation de la performance puisque nous avions réalisé autant de fonctionnalités dans ce sprint que lors des 3 derniers.

Au final, nous sommes donc arrivés à une organisation en minimum 1 réunion par semaine en plus des meetings organisés pendant les heures de cours. Dans ceux-ci, nous expliquions chacun ce qu'on a fait et ce qu'on va faire pour la prochaine fois. Pour chaque fonctionnalité nous avions un *item* constitué lui-même de plusieurs *task* à réaliser par un développeur volontaire. Dans la liste des fonctionnalités, nous ajoutions également des *improvements* qui sont des petites phrases décrivant des comportements à adopter pour améliorer l'efficacité, par exemple "être plus communicatif", pour bien se rappeler de l'être à chaque fois que nous consultions les tâches que nous avions à faire.

5. Modèle de données

a. Modèle général

Pour décrire le fonctionnement de l'application, nous avons réalisé un diagramme UML simplifié présentant les relations entre les différents modèles de l'application.



On peut observer sur ce diagramme la relation entre l'Utilisateur (User) et l'Enseignant. En effet, dans cette application il y a deux types d'utilisateurs : les Utilisateurs, ceux qui ont un compte et qui peuvent se connecter, et les Enseignants, ceux qui peuvent être affectés à des modules et dont on peut donc gérer les affectations. Mais, un Utilisateur peut aussi être un Enseignant et un Enseignant peut aussi être un Utilisateur. On a choisi de gérer cette propriété par la relation 0..1 entre ces deux entités. Si un Enseignant a un compte alors il connaît un Utilisateur et l'Utilisateur le connaît également, et inversement.

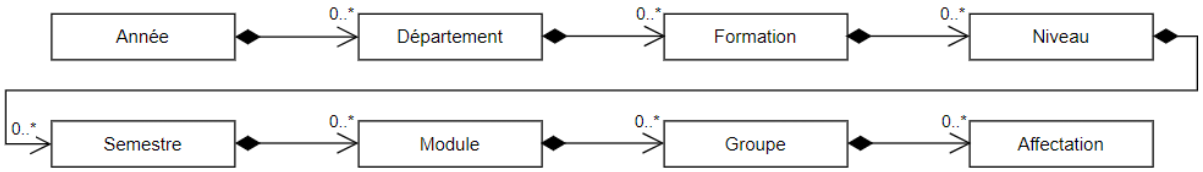
b. Initialisation des Données

Pour gérer la création des tables de la base de données, nous utilisons JPA qui offre notamment la possibilité de créer automatiquement les tables dans la base de données selon différents paramètres (voir l'option *spring.jpa.hibernate.ddl.auto*), c'est ce que nous utilisons pour la création des tables.

Pour s'assurer d'avoir à disposition des données pour les tests rapides en développement, nous avons créé un service dans le backend nommé *InitDevServiceDefault*. C'est donc dans ce service que toutes ces données sont créées.

c. Arborescence des Modules

Pour décrire l'arborescence des modules, concentrons nous sur la partie arborescence du diagramme précédent :



Nous avons défini cette arborescence de manière à ce qu'il soit possible de la parcourir dans les 2 sens : des Années vers les Affectations et des Affectations vers les Années. Nous avons fait ce choix car il était nécessaire de pouvoir accéder aux éléments "voisins" de chaque élément dans l'application, que ce soit simplement pour l'affichage des données mais aussi pour de nombreuses autres fonctionnalités. Chaque élément contient une liste d'éléments enfants et un élément parent. La structure est donc sous forme d'arbre, le nombre d'éléments augmentant en fonction de la progression dans l'arbre.

Nous allons maintenant présenter les champs contenus dans chaque élément de l'arborescence. Chaque élément possède un identifiant permettant son stockage dans la base de données et son accès depuis l'application. Nous allons donc omettre ce champ dans la présentation des éléments car commun à tous et peu intéressant.

- Année :
  - debut : année de début de l'année administrative (2025 pour l'année 2025-2026)
  - departements : liste de départements contenus dans cette année.
  - heuresAssignées : liste des heures assignées à cette année.
- Département :
  - nom : nom du département
  - formations : liste des formations contenues dans ce département
  - responsableDépartement : personne responsable de ce département
  - annee : annee dans laquelle ce département est contenu

- Formation :
  - nom : nom de la formation
  - totalHeures : nombre total d'heures que cette formation doit couvrir
  - responsableFormation : personne responsable de cette formation
  - niveaux : liste des niveaux contenus dans cette formation
  - departement : departement dans lequel cette formation est contenue
- Niveau :
  - nom : nom de cette formation
  - formation : formation dans laquelle ce niveau est contenu
  - semestres : liste de semestres contenus dans ce niveau

- Semestre :
    - nom : nom de ce semestre
    - niveau : niveau dans lequel ce semestre est contenu
    - modules : liste de modules contenus dans ce semestre

- Module :
    - nom : nom de ce module
    - heuresParType : nombre d'heures associé à chaque type d'heure de ce module (exemple : CM → 20H, TD → 10H…). Ces données servent à définir les heures assignées aux groupes de ce module.
    - semestre : semestre dans lequel ce module est contenu
    - groupes : liste de groupes contenus dans ce module

- Groupe :
    - nom : nom de ce groupe
    - totalHeuresDuGroupe : nombre total d'heures qui doivent être affectées pour ce groupe. Cette valeur doit correspondre à la définition du type d'heure associé dans le module parent.
    - heuresAffectées : nombre d'heures déjà affectées de ce groupe. Cette valeur sera mise à jour en fonction des affectations de ce groupe.
    - date : date à laquelle ce groupe a été créé
    - type : type d'heure associé à ce groupe
    - affectations : liste des affectations contenues dans ce groupe
    - module : module dans lequel ce groupe est contenu

- Affectation :
    - heuresAssignées : nombre d'heures du groupe parent qui ont été affectées avec cette affectation
    - dateAffectation : date à laquelle cette affectation a été créé
    - commentaire : commentaire optionnel permettant de préciser des informations relatives à cette affectation
    - enseignant : enseignant affecté au groupe parent grâce à cette affectation.
    - groupe : groupe dans lequel cette affectation est contenue.

6. Difficultés rencontrées et améliorations possibles

a. Gestions des années

L'application que nous avons réalisée a pour but la gestion des horaires des enseignants durant une année scolaire. Du fait qu'il faut gérer cela en fonction de l'année, il faut qu'une grande partie des modèles de l'application soit liée au système d'années. Le problème étant que nous avons appris ce fonctionnement après avoir déjà réalisé une grande partie du modèle de l'application. Il a donc fallu réfléchir à un moyen de gérer les années. Nous avons

essayé de le faire, mais on se rend compte aujourd'hui que certaines parties ne sont pas encore réglées.

C'est principalement le système de permissions qui a besoin de changements. En effet, actuellement, les permissions d'un utilisateur sont liées à son rôle, mais le rôle d'un utilisateur pour l'application est celui qu'il possède pour l'année enregistrée, étant la plus récente. Si on change d'année via le sélecteur, les permissions de l'utilisateur connecté ne changeront donc pas.

Il faut par conséquent revoir le système de permissions de l'application, pour que celui-ci soit lié au système d'années.

b. Mise en production

Pour pouvoir être utilisée par les utilisateurs, une application doit être mise en production et celle-ci ne déroge pas à la règle.

Nous avons rencontré des difficultés concernant le déploiement. Tout d'abord parce que notre *docker compose* était à la base adapté que pour le développement, il a donc fallu en créer un nouveau, ce qui a été fait dans le fichier *prod.compose.yaml*. Pour démontrer les difficultés que nous avons rencontrées, expliquons le contenu de ce fichier.

Ce *docker compose* gère 4 conteneurs différents : le serveur backend (Spring boot), le serveur frontend (Angular), le système de bases de données (PostgreSQL) et le reverse proxy (Nginx).

Habituellement dans un docker compose destiné à la prod nous renseignant uniquement des images déjà préconstruites provenant d'une *registry* mais ça n'a pas été le cas ici. En effet nous n'avons pas beaucoup de droits concernant la registry de l'université et n'avons pas voulu publier l'image en public non plus via Docker Hub, par exemple. Donc ici, les images de backend et de frontend sont fabriquées lors du lancement du *docker compose*. À noter que si vous souhaitez publier ces images par la suite, on peut tout à fait les préconstruire pour les publier et ce, assez rapidement.

Nous n'avions pas de machines à disposition pour déployer notre application, nous ne l'avons donc finalement pas déployée. Cependant, tout est prêt au niveau du *docker compose* pour la configuration en production puisque nous avons un reverse proxy *Nginx* configurable pour le déploiement de l'application. Plus d'instructions sont données dans le fichier *README.md* sur le dépôt git.

c. Organisation générale

Comme nous l'avons dit précédemment, nous avons adopté un cadre de travail Scrum pour ce projet. Mais nous l'avons dit également, ce cadre n'était pas respecté à sa pleine efficacité particulièrement lors des premiers sprints. Une piste d'amélioration serait donc d'assurer de meilleures méthodes de travail pour améliorer l'efficacité.

Également, comme nous n’avions pas démarré le projet de la façon la plus organisée, il y a beaucoup d’incohérences dans les noms de fichiers et de requêtes puisque chaque développeur a développé à sa manière. Il faudrait donc, par la suite, harmoniser l'arborescence des fichiers et les différentes requêtes.