

---

**Rapport de projet collaboratif**  
**Création d'un logiciel de gestion d'emploi du temps**

---



Rédigé par Jason LAFFAILLE, Théo HAFSAOUI & Tom DOMENGE  
Supervisé par Laurent LOISEAU & Jérémy MALLOFRE

Master d'informatique, développement et ingénierie des données (DID), 1<sup>re</sup> année.  
Université de Toulon, Version du 9 janvier 2023.



# Table des matières

<b>Table des matières</b>	<b>I</b>
<b>Introduction</b>	<b>III</b>
Avant-propos . . . . .	III
Rencontre des acteurs . . . . .	III
<b>Chapitre I. Exigences</b>	<b>1</b>
I.1. Utilisateur . . . . .	1
I.2. Administrateur . . . . .	1
I.3. Professeur . . . . .	2
I.4. Système . . . . .	2
<b>Chapitre II. Axe fonctionnel</b>	<b>3</b>
II.1. Diagramme de contexte statique . . . . .	3
II.2. Diagramme de cas d'utilisation . . . . .	3
II.3. Paquetage . . . . .	4
<b>Chapitre III. Axe statique</b>	<b>9</b>
III.1. Entités et objets métier . . . . .	9
<b>Chapitre IV. Axe dynamique</b>	<b>13</b>
IV.1. M.V.C . . . . .	13
IV.2. Entité . . . . .	13
<b>Chapitre V. Implantation</b>	<b>17</b>
V.1. Entité . . . . .	17
V.2. Facade . . . . .	17
V.3. Interface Homme Machine . . . . .	18
V.4. Test Unitaire . . . . .	22
V.5. Gestion de la persistance : Base de données . . . . .	22
V.6. Intégration continue . . . . .	24
<b>Chapitre VI. Gestion de projet</b>	<b>25</b>
VI.1. Chronologie . . . . .	25
VI.2. GitHub flow . . . . .	27
VI.3. Kanban . . . . .	30
VI.4. Analyse & conclusion . . . . .	31

<b>Chapitre VII. Conclusion analyse &amp; ouverture</b>	<b>33</b>
<b>Annexes</b>	<b>35</b>
<b>Table des figures</b>	<b>v</b>

# Introduction

## Avant-propos

Ce rapport est le résultat du travail réalisé sur le projet du premier semestre du M1 master info d'ingénierie logicielle. ce projet consiste en la réalisation sur 11 semaines d'un logiciel d'emploi du temps avec un fort accent sur la gestion de projet agile avec le soutien de professionnels (techlead/buisness analyst) de Sopra. Plus particulièrement, un effort important a été réalisé afin d'utiliser des méthodes de gestion moderne avec, github, pull request, kanban, intégration continue, le tout en parfaite synchronisation avec les possibilité offerte par Github.

Pour résumer, ce projet consiste en la réalisation d'une application similaire à hyperPlanning avec une gestion de base de donne et d'utilisateur sur java avec le SGBD de notre choix.

## Rencontre des acteurs

Dans le cadre de l'appropriation du sujet, nous avons décidé, dès que possible, d'interroger les acteurs primaires d'Hyperplanning dans notre université et à l'étranger. Pour ce faire, nous avons utilisé plusieurs mediums, D'abord bien sur des discussions physiques, nous remercions d'ailleurs M.VIA qui a pris le temps de nous parler deux fois, ensuite avec des formulaires en ligne. Ces retours furent riches d'information ; les utilisateurs d'Hyperplanning avait une tendance naturelle à se concentrer sur leur frustration et peu sur les composantes essentielle et utile pour eux, un travail de synthèse fut donc nécessaire pour la création des UC (cf.??).

**Formulaire - Hyperplanning**

Dans le cadre d'un projet de génie logiciel, nous aimerions vous poser quelques questions relatives au logiciel de gestion d'emploi du temps de l'université (hyperplanning).

Merci d'avance pour votre contribution.

1. En ce qui concerne l'emploi du temps vous considérez-vous plutôt comme ?

- ☐ Gestionnaire/Apprenteur
- ☒ Etudiant
- ☐ Enseignant
- ☐ Extérieur/Invité

5. Comment jugez-vous les fonctionnalités suivantes dans hyperplanning ?

	Inutile	Rarement utile	Souvent utile	Essentielle
Consulter l'emploi du temps (journalier)	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Consulter l'emploi du temps (semestre)	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
Consulter l'emploi du temps (mois)	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Consulter l'emploi du temps (semestre)	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Consulter l'emploi du temps (année)	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Consulter vos absences et retards	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Consulter l'état d'avancement de vos cours	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

7. Quelle est votre principale source de frustration avec hyperplanning ?

rien

Ci-dessous les retours synthétisés des différents acteurs principaux.

## Enseignants

D'après les retours et les avis différents reçu par les Acteurs Enseignants d'Hyperplanning, on peut en extraire de multiples composantes essentielles quant à l'utilisation du logiciel, tel que la consultation d'emplois du temps (par jour / semaine / mois), en revanche l'avis de consulter son emploi du temps au semestre ou à l'année était en faveur de l'indifférence, il en est de même pour consulter la progression d'un cours, ainsi que communiquer avec les étudiants directement depuis Hyperplanning. Contre toutes attentes, la réservation d'une salle est également sujet d'indifférence, puisque chaque enseignant doit d'abord notifier le Gestionnaire de la promotion pour y communiquer les changements au Gestionnaire du Planning. Pour conclure, nous avons demandé aux Acteurs, quelles sont vos principales frustrations quant à l'utilisation d'Hyperplanning, leur réponse été davantage tournée côté gestion qui traîne parfois à les notifier. Une idée a aussi été évoquée d'initier un système de notification automatique pour les étudiants et les enseignants en cas de changement soudain ou report de cours. On nous a aussi informés que si un Enseignant Chercheur souhaitait réserver une salle pour une réunion, hors du cadre d'une quelconque formation, elle ne pouvait être renseignée sur Hyperplanning sans créer une autre cohorte hors de la formation, donc pourquoi pas intégrer une possibilité de renseigner les réunions de différents intervenants. Finalement, un dernier point sur la facilité d'utilisation nous a été reporté, tel que le choix de filtrer les différentes informations de l'emploi du temps, filtrer par type d'enseignement (CM, TD, TP), ou simplement le numéro d'un groupe, des filtres qui ne sont actuellement pas possibles avec Hyperplanning. Les réponses obtenues sont riches en informations sous leur contexte d'utilisation respectif.

## Étudiant

Une attention particulière fut apportée sur le profile des étudiants sondés, avec une diversité de statue professionnelle, d'U.F.R, mais aussi géographique avec des étudiants étrangers. Les résultats sont similaires à ceux des étudiants avec une nette préférence pour l'affichage de l'emploi du temps de la journée et de la semaine. le reste des fonctionnalités est jugée *"inutile"* ce qui peut sembler intuitive concernant un emploi du temps, a noté que beaucoup de fonctionnalité sont inconnues des étudiants, comme l'e.d.t des salles ou l'accès a e.d.t d'autre promotion.

## Gestionnaire

Le gestionnaire, lui, nous a fait part de son souhait d'une interface plus sobre, tout en conservant la possibilité de visualiser en direct les emplois du temps de tous les groupes. Il nous a indiqué que la gestion des disponibilités des enseignants était un processus transverse, de sorte que nous n'avons pas retenu cette fonctionnalité dans notre logiciel.

# Chapitre I

## Exigences

### I.1 Utilisateur

Un utilisateur doit pouvoir voir son emploi du temps

---

Un utilisateur doit pouvoir se connecter de manière sécuriser à son emploi du temps

---

Un utilisateur doit pouvoir voir l'emploi du temps de la semaine prochaine

---

Un utilisateur doit pouvoir voir l'emploi du temps de la semaine précédente

---

Un utilisateur doit pouvoir voir l'avancement d'un cours.

---

### I.2 Administrateur

L'administrateur doit pouvoir se connecter de manière sécuriser à son emploi du temps

---

L'administrateur doit pouvoir voir l'emploi du temps d'un utilisateur.

---

L'administrateur doit pouvoir voir l'emploi du temps d'une salle.

---

L'administrateur doit pouvoir voir l'emploi du temps d'un enseignant.

---

L'administrateur doit pouvoir supprimer un créneau

---

L'administrateur doit pouvoir modifier un créneau

---

---

L'administrateur doit pouvoir ajouter un créneau

---

L'administrateur doit pouvoir lire un créneau

---

### **I.3 Professeur**

Un professeur doit pouvoir voir son emploi du temps

---

Un cours doit pouvoir avoir plusieurs enseignants

---

Un professeur doit pouvoir se connecter de manière sécurisée à son emploi du temps

---

Un professeur doit pouvoir voir l'emploi du temps de la semaine prochaine

---

Un professeur doit pouvoir voir l'emploi du temps de la semaine précédente

---

Un professeur doit pouvoir voir l'avancer d'un de ces cours

---

Un professeur doit pouvoir renseigner une indisponibilité

---

### **I.4 Système**

Le système doit se charger en moins de 3 seconde

---

Le système doit assurer la cohérence de l'emploi du temps avec :

1. Un enseignant ne peut être sur deux créneaux au même instant
  2. Une promotion ne peut être sur deux créneaux au même instant
  3. Assurer l'unicité des matières
  4. Assurer l'unicité des enseignants
  5. Assurer l'unicité des promotions.
-



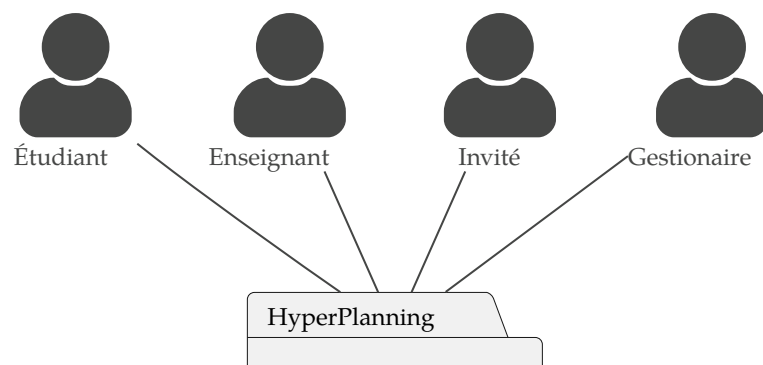
## Chapitre II

# Axe fonctionnel

L'axe fonctionnel se concentre sur les fonctionnalités qu'offrent notre logiciel, plus spécifiquement le but de cette partie est de dégager les *acteurs* du système, les *service* que le système doit rendre aux acteurs et les interactions entre les deux.

### II.1 Diagramme de contexte statique

face à cet objectif, nous devons commencer par un diagramme de contexte statique, ce diagramme isole le système et met en lumière les acteurs qui interagisse avec lui.



Ce diagramme a été réalisé grâce aux attendus du projet ainsi que les interviews réalisées au début de ce rapport. Deux choses sont à noter de ce diagramme, premièrement les cardinalités sont absentes, car très probablement identiques. Deuxièmement, les acteurs secondaires sont aussi absents, en effet il nous apparaît précipité de définir nos choix d'implantations et de subir involontairement une dette technique, dans une étape aussi précoce de la modélisation.

### II.2 Diagramme de cas d'utilisation

Maintenant que les acteurs sont définis, il est nécessaire de mettre en évidence les services rendus par notre système, afin d'éviter encore une fois un choix précoce nous allons commencer ce premier Cas d'usage par une approximation de la réalité que nous raffinerons plus tard.

Bien sûr, il ne s'agit que d'une approximation, mais deux choses importantes sont à noter :

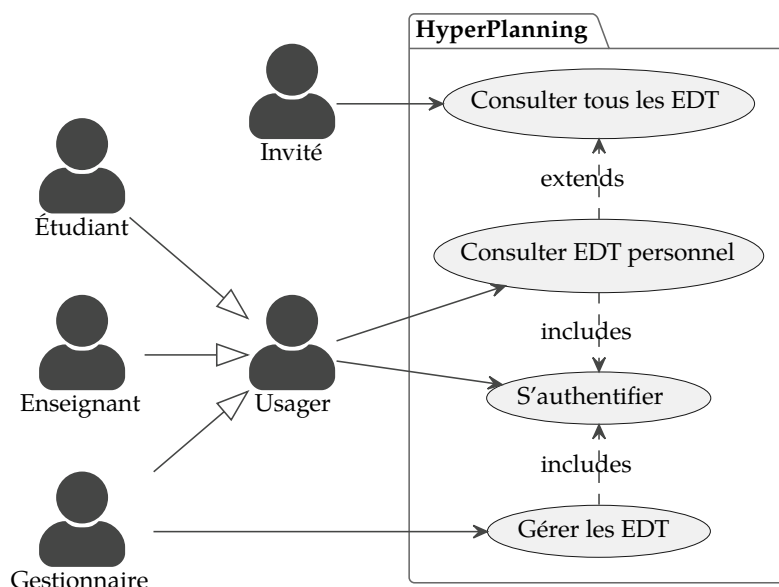


FIGURE II.1 – Diagramme de cas d'usage

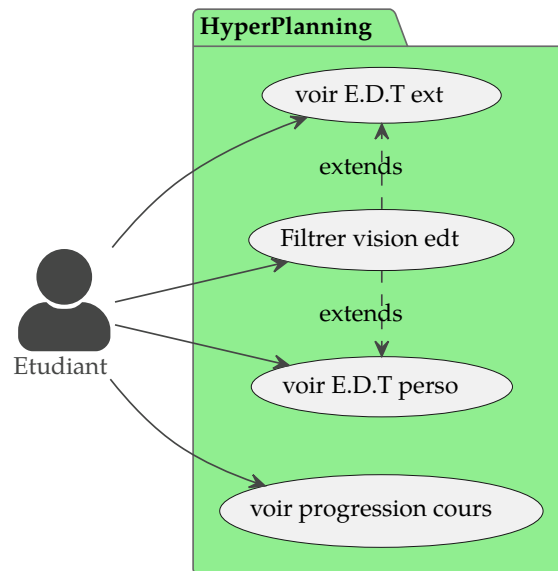
1. L'héritage entre acteurs. ici nous ne parlons pas forcément d'implantation mais bien d'un héritage contraint par les fonctionnalités, il s'agit d'un héritage artificielle car bien sur un Enseignant n'est pas *exactement* un Étudiant. Subtilité que nous étudierons dans les diagrammes suivant.
2. Le faible nombres UC. Ce faible nombre est un choix de granularité grossier pour mettre en évidence les lien entre acteurs. Même si au final cela représente tout nos objectifs pour ce projet.

Bien sur l'étape suivante, est l'écriture des scénario nominaux mais c'est le domaine de l'axe statique. Néanmoins si nous soulignons l'importance de cette étapes c'est parce que c'est grâce a elle que nous avons trouver les prochaine UC as un niveaux de granularité qui nous satisfaisait. Aussi si la structure du rapport peut sembler chronologique il n'en n'est rien la conception as était un meli-melo avec chaque étapes qui soulevait des ambiguïté et qui mettait en lumière des erreur faite dans les étapes précédente, ceci sans inclure l'implantation qui bien sur nous forcis a faire de nombreux changement.

### II.3 Paquetage

Cette Partie est celle ou nous obtenons la version final de nos UC. L'objectif ici est diamétralement différent du premier UC, en effet si précédemment nous souhaitions unir est trouver les fonctionnalité en commun, ici nous supposons chaque acteur unique. C'est a dire que le système a pour unique objectif de le servir lui, cette approche a pour desseins de faire ressortir les différence, et surtout de s'assurer de la cohérence de notre première UC avec la réalité des besoins de *chaque* acteurs.

### II.3.1 Étudiant

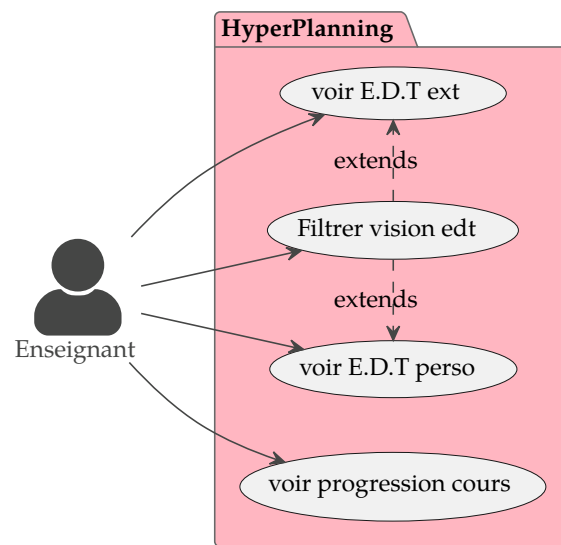


Cette UC est sans doute la plus connue des membres de l'équipe. En effet étant *Etudiants* il s'agissait sans doute du plus facile à cerner. Cette UC n'as pas toujours était aussi simple en effet lors des première itération nous avions une version beaucoup plus précise, mais inutilement précis, particulièrement en ce que nous avons appeler les *Filtre* qui ne sont au final que différent vue d'un même EDT préciser tout ces différent filtre ne polluer pas seulement la lisibilité du diagramme, elle rendait le développement imminent plus rigide après tout devons nous vraiment avoir une vision journalière? La réponse est non, en tout cas c'est le choix de l'équipe après avoir décider d'abandonner ces UC trop complexe.

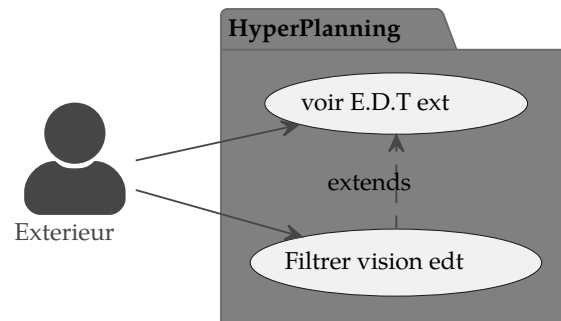


### II.3.2 Enseignant

L'enseignant est au final beaucoup plus similaire à l'étudiant du point vue fonctionnelle que l'on aurait pus se l'imaginer. En effet même si en terme d'implantation étudiant et professeur sont deux objet très différent, car bien sur un professeur est intrinsèquement lier a la notions de cours est donc d'emploi du temps au contraire d'un étudiant lui qui n'est lier que par ça promotion. Cependant malgré cette différence de nature les deux possède deux UC identique. Une nuance est néanmoins as apporter a cause des interview, puisque les deux n'apporte pas la même importance au fonctionnalité avec ici une nette augmentation de l'intérêt pour la progression du cours.

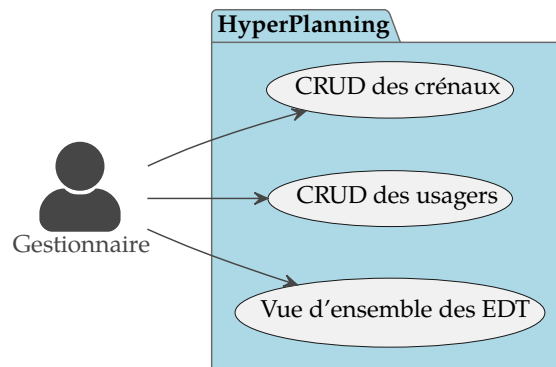


### II.3.3 Extérieur



l'extérieur est aussi un acteur intéressant, tant son existence n'est pas nécessaire. c'est fonctionnalité elle-même aussi similaire aux autres utilisateurs avec évidemment la suppression des UC EDT PERSO et les UC de progression de cours qui n'ont aucune raison d'être pour une personne qui n'a a priori aucun cours. Je l'ai donc supprimé car son existence était remise en question, mais il a été gardé tant qu'il est au final une étape afin de réaliser toutes les fonctionnalités du Gestionnaire.

### II.3.4 Gestionnaire



Le gestionnaire comme dit précédemment n'est en réalité qu'un hybride entre les utilisateur et les extérieur, avec la possibilité de voir l'emploi du temps d'un étudiant, d'un enseignant ou d'une promotion, cette diversité de vision est nous l'avons compris au fil de nos discussion un élément crucial dans le travail de M.VIA. aussi nous avons compris qu'il était donc nécessaire d'avoir des UC de lecture EDT solide, cependant pendant le developement le choix as était fait d'abandonner la vision par étudiants et par enseignants, nous croyons en effet que le rôle du gestionnaire n'est pas la gestion des individus mais la gestion de promotion, il s'agit aussi pour nous d'un raccourcis car cela nous permet d'éviter la création de feature inutile, qui surchargerait inutilement l'interface.

Enfin bien sur l'une des feature essentielle pour le gestionnaire et la modification, suppression, et l'ajout de cours. il s'agit la d'une partie nous le savons complexe est qui pourrait recevoir une myriade d'option pour simplifier la vie du gestionnaire, cependant cette richesse d'option seras réserver pour la roadmap et les amélioration potentielle puisque nous pensons d'abord essentielle d'avoir le coeur de la gestion avant leur amélioration potentielle, d'où la simple UC CRUD qui est nous le savons bien plus complexe que sont court nom pourrait laisser entendre.



## Chapitre III

# Axe statique

To build a software that your users understand, capture the language of th[ose] users  
in a class diagram.

— MICHAEL JESSE CHONOLIS, in *UML 2 for Dummies*

L'axe dynamique décrit l'aspect structurel de notre système, les classes qui le composent et leur relation. Une fois les exigences définies, une grande partie du temps avec les intervenants a consisté à dégager les éléments de la statique de notre système. Ainsi, le modèle change selon que l'on restreint un cours à un ou plusieurs enseignants : Toute la modélisation de la base de données en dépend également. Les retours du gestionnaire en la matière se sont montrés particulièrement intéressants, et nous ont permis de dégager la hiérarchie de classe finale de notre projet.

### III.1 Entités et objets métier

Comme nous aurons l'occasion de revenir là-dessus, l'une des clés de voûte de notre conception et la phrase ci-dessous. c'est un élément important, il synthétise toutes les informations nécessaires pour avoir un *Cours*.

Le vendredi 25 novembre 2021 *de neuf heures à midi*, les étudiants du *Master 1 Informatique* assistent à une séance de *travaux dirigés*, dispensée par MM. *MALLOFRE et LOISEAU* dans le cadre du module *projet collaboratif (I 143)*, en salle *U.001*.

Les informations en *italique* sont propres à chaque cours, mais l'on peut déjà en dégager des informations comme des contraintes

1. spatio-temporelles : dans une certaine salle, à une heure donnée
2. de multiplicité, pour les enseignants
3. de dépendance à d'autres objets, les noms d'enseignants et de promotions étant vraisemblablement présents sur plusieurs cours

Forts de ce constat, nous et après de multiple modification, nous avons obtenue ce diagramme :

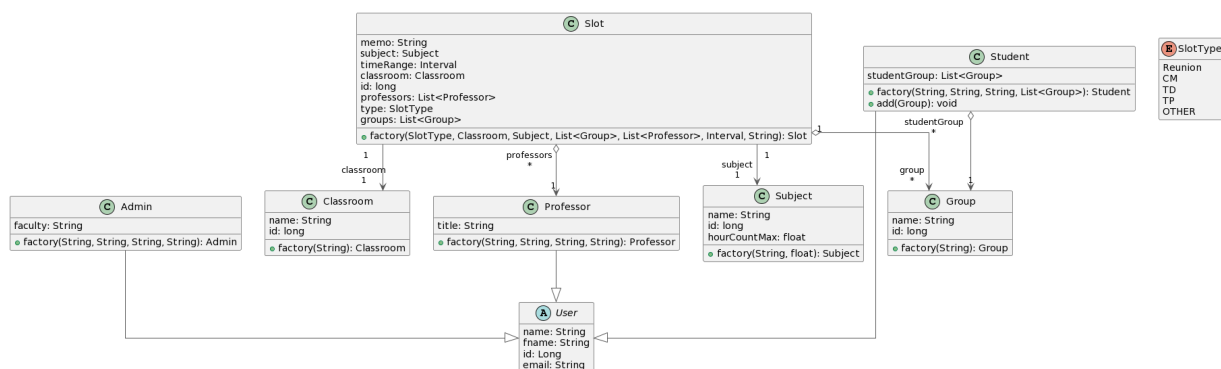
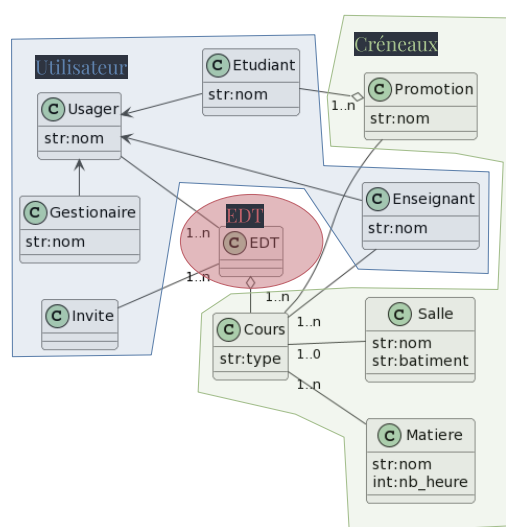


FIGURE III.1 – Diagramme de classes des entités

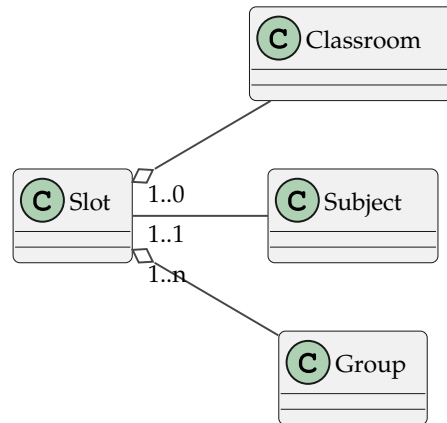
Bien sûr, il s'agit d'un diagramme plutôt vaste puisqu'il regroupe tout notre système, aussi l'une des étapes réalisées afin de vraiment comprendre ce diagramme est la scission en *paquets*. Ce paquetage est réalisé d'abord de manière sémantique, on réunit les classes avec des rôles en commun, ensuite on essaie de réduire au maximum les relation et cardinalités entre paquets. de ces règles simples, on en obtient la séparation ci-dessous.



A notée que ce diagramme n'est pas le plus récent, mais ici, il servira de support pour expliquer certains des changements fait pendant le projet, dans nos choix de conception.



1. **Usager** : Le packet Usager regroupe tous les acteurs primaires de notre système, les relations entre classe sont principalement des relations d'héritage. Ainsi est de manière logique *un Enseignant est un utilisateur*, la seule exception à ce raisonnement est l'invite d'où sa disparition dans le diagramme final. À la fin, la seule à interagir en dehors du paquet est *Usager* avec un lien 1 ... N.



2. **Cours** : Ce packet regroupe les classes composant un créneau, autrement appelé cours ou Slot en anglais. la ou le packet *Usager* était très uniforme dans les relation entre ces classes, le paquet *Creneaux* est beaucoup plus hétéroclite. Pour commencer par le plus simple, nous avons les relations entre *Matière* et *Classe* avec la classe *Cours*, ces relations sont simples, car il s'agit juste d'un lien 1 ... 1. cela s'explique par le bon sens, un cours sans sujet ni lieu n'a aucune raison d'exister et réciproquement. Le lien entre un *Enseignant* et un *Cours* est un peu plus complexe, déjà enseignant est à cheval entre ce packet est celui d'*Usager* rendant sa place incertaine, mais en plus là où dans les premières versions de notre diagramme la relation entre lui et *Cours* était simple, 1 ... N, elle s'est aujourd'hui muée en une agrégation afin de respecter les exigences de notre logiciel.  
La dernière relation est de la loin la plus complexe, elle aussi est de cardinalité 1..N, et elle aussi est une agrégation, ici entre *Promotion* et *Cours*, mais ici la particularité est qu'une *Promotion* est elle même une agrégation d'*Étudiant*, *Étudiant* qui lui fait partie du packet *Usager*. Ici est peut-être la faiblesse de notre conception tant cette ambivalence fut source de soucis.
3. **EDT** : Il s'agit un packet étrange car il n'est composé que d'une classe, qui lie les deux autres, il n'est pas l'unique lien entre les deux packet, mais il est l'unique lien entre les deux coeur de ceux-ci, *Usager* et *Cours*. son rôle est assez ambivalent et surtout indéfini dans la conception, il prendra au final la forme d'une facade dans le système final.



## Chapitre IV

# Axe dynamique

La tâche d'un réalisateur est de créer une dynamique entre ses acteurs.

— NICK CASSAVETES

L'axe dynamique traite de la dynamique des classes et acteurs, dans cette partie deux points ont reçu une attention particulière, le modèle et la vue.

### IV.1 M.V.C

Ce projet, de par les différents types de données et d'opération qui sont effectuées, présente une part de modélisation temporelle non négligeable : Se pose ainsi la question de la mise à jour de l'interface, lors de l'ajout d'un cours, la gestion des cas d'erreur en cas de données erronées de la part des utilisateurs, entre autres.

Afin de fixer les idées, il peut être utile d'étudier le modèle précisé dans le sujet : le modèle vue-contrôleur (M.V.C.).

Il est à noter que dans le modèle M.V.C. traditionnel, les requêtes en lecture étaient également l'apanage du contrôleur, nous avons décidé, d'un autre approche, afin d'obtenir une version plus réactive : dans notre approche, la vue *observe* le contrôleur, lequel la notifie lorsque des changements surviennent. Enfin, pour limiter la complexité des appels à la D.A.O, nous utilisons une façade, sorte de "guichet unique" pour nos requêtes.

Afin d'y voir plus clair, un diagramme de séquence paraît approprié.

### IV.2 Entité

Un élément important à sous-ligner est l'ambivalence d'états que peut avoir une entité, en effet une entité persisté ou non possède deux états différents, en terme d'implantation ces états sont reflétés avec l'ID.

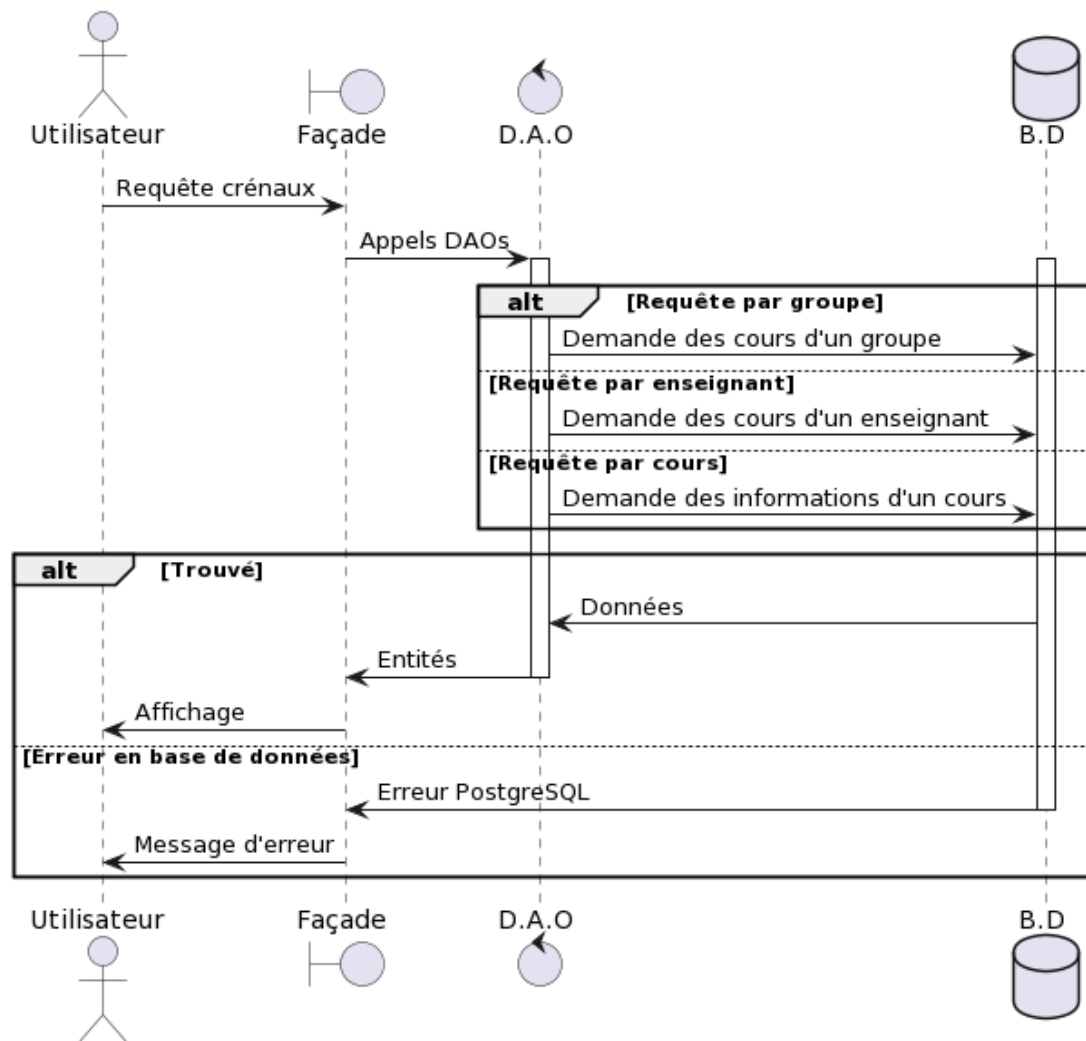


FIGURE IV.1 – Diagramme de séquence lors d’une lecture en base après notification

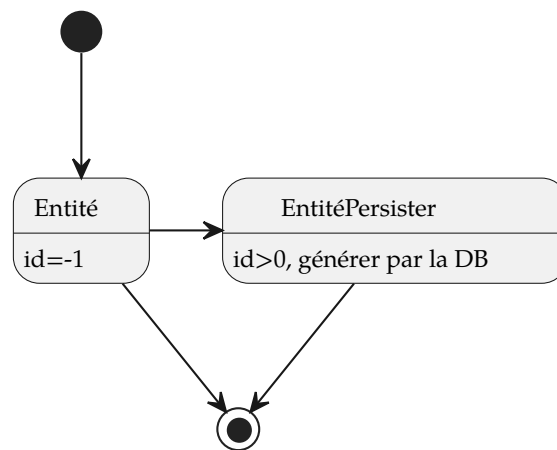


FIGURE IV.2 – Diagramme d'états d'une entité



## Chapitre V

# Implantation

*We who cut mere stones must always be envisioning cathedrals.*

— CRÉDO DES TAILLEURS DE PIERRE

En tant d’informaticien, dire que cette partie n’était pas attendue avec impatience par notre équipe serait un mensonge, aussi étions-nous très heureux de commencer le développement. néanmoins, comme indiquer dans la chronologie, cette étape arriva après quatre itérations, soit plus d’un mois, il s’agit d’un délai important, car au final l’une des étapes les plus importants de la conception et sa confrontation au réel et l’évolution que cela implique. Malgré tout et conscient du temps limite qui nous était imposé, c’est avec enthousiasme que nous avons développé. Le module Ingénierie logicielle dans sa partie java passe beaucoup de temps as parlé de POO, programmation défensive et design patern, et c’est donc avec une attention toute particulière à ces sujets que nous avons construits notre système.

### V.1 Entité

Les entités sont sans doute l’aspect le plus simple de l’implantation, c’est d’ailleurs pour cette raison que l’implantation de ceux-ci a était la première étape dans notre développement. Peu de choses sont as noté de cette partie, sinon bien sur les difficultés rencontrées lors des DAO avec la persistance des entités. L’un des éléments complexe de ces entités est l’intrication entre les *Group* et les *Etudiant*, cette intrication est bien sûr le reflet de la réalité, implique un choix d’implantation afin d’éviter une redondance d’information qui ne serait être persister. Le choix final, c’est porter sur les *Group* qui sont composés d’étudiant.

### V.2 Facade

La façade est un design patern connue, son but est de fournir une A.P.I afin de simplifier l’usage du modèle et de le découpler du controler.

Dans notre cas, l’usage est multiple :

1. **Formatage** : dans le cadre d’un ajout d’entité, il est nécessaire de fournir des méthodes qui s’assurent de l’uniformité du formatage afin d’éviter une interface hétérogène.
2. **Exception** : de manière naturelle, notre DAO renvoi un certain nombre d’exceptions et *Optional*. Étant admis qu’une interface qui crashe suite à l’échec d’une recherche est inacceptable, il faut traite ces erreurs. Cependant, si on pourrait penser pouvoir traite ces erreurs dans le controleur, ces oubliés la multiplicité *des* controleurs, aussi la gestion de ces erreurs serait traiter plusieurs fois, ne

respectant pas ainsi le principe DRY, et provoquant de facto un couplage important entre la DAO et l'IHM. De plus, nous avons pour projet d'étendre le nombre Exception, il ne s'agit donc pas d'un problème anodin.

Maintenant des limites sont as sous-ligner, en effet cette façade est de loin le plus gros fichier Java du projet. La raison n'est pas tant sa complexité, mais sa richesse de surcharge, tant est si bien qu'avec plus de huit-cent lignes la lecture et l'orientation est complexe, rajouter à ça le patern d'observateur observé, et vous obtenez une facade sans doute trop grosse. Plus important que le nombre de lignes est le scope de cette facade, c'est facile, il est difficile de trouver une classe sans interaction avec cette facade, ainsi nous nous approchons dangereusement d'un anti-patern le *god object*. Mais il s'agit d'un choix conscient, motive par la faible échelle de notre projet et la praticité de cette facade.

### V.3 Interface Homme Machine

Pendant que les DAOs furent développer en adéquation avec la base de données, la question fut poser de comment pourriez-vous afficher notre application d'emploi du temps autrement que par terminal, la réponse est évidente une interface.

Le sujet ne nous a pas imposé de bibliothèque précise en ce qui concerne l'IHM, donc nos premières déductions, on était peut être une bibliothèque html/java, ou quelque chose de similaire, pour afficher proprement les données dans une page web ou une application standalone. Au fil du temps les conseils autour de ces questions sont claires, utiliser *JavaFx* pour votre application. *JavaFx* se fut, et après quelques recherches nous sommes parvenu tout d'abord à trouver une bibliothèque qui réplique un Google Calendar en *JavaFx*, son nom : *CalendarFx*

Une bibliothèque qui s'occupe entièrement d'afficher un planning complet, gérant les événements à certaines dates et heures, différentes vue sur la semaine ou la journée. Au vue du temps limité dont nous disposons, cette bibliothèque semblait presque trop parfaite. Donc sur 1 semaine, le travail à été répartis entre nous, quelqu'un s'occupe de continuer la base de données, quelqu'un d'autre s'occupe d'avancer les DAOs en parallèle, et le dernier s'occupait d'intégrer *CalendarFx* dans le projet.

Hors la vision parfaite et utopique que nous a vendu cette bibliothèque s'est vite estompé, l'intégration prenait beaucoup plus de temps que prévu et était bien trop complexe pour très peu d'avancement, on remarqua également qu'elle manquait cruellement de documentation et de ce qui pourrait être utile quant au projet de planning scolaire.

On essaye donc de continuer et d'intégrer *CalendarFX* au projet juste pour avoir un exemple à montrer lors de la prochaine séances de révision. L'avis général a été assez mitigé, donc à partir de là, nous nous sommes concertés et avons décidé de retirer complètement *CalendarFX* du projet, étant donné qu'elle est trop complexe à intégrer pour de simple exemple, ç'aurait été bien plus complexe et problématique de l'intégrer à tout notre système de gestion et de planning.

Finalement, on supprime tout ce qui concerne cette bibliothèque pour repartir de zéro côté IHM, et la semaine qui suit sera surtout un travail de rattraper le retard engendré par *CalendarFX* et de créer notre propre interface de Planning à partir de *JavaFx*. Concernant la structure de l'interface, on utilise des fichiers XML et CSS pour définir, respectivement, quel composant de *JavaFX*, nous allons utiliser ainsi que leur aspect graphique. Simplement dit, nous créons des classes intermédiaires pour manipuler facilement notre emploi du temps et afficher son contenu dans une grille (resp. *GridPane*).

Toute la conception de l'interface à partir de *JavaFx* s'est faite en respectant le paterne MVC, comme décrit dans la section FaçadeV.2, celle-ci sert de pont entre le Modèle et le reste de l'application. Donc notre paquet Contrôleur, qui s'occupe de gérer toutes les actions dynamiques de l'interface, communique avec la Façade pour tout ce qui concerne l'ajout, la modification, suppression ou lecture d'un cours, ou d'un emploi du temps. Puis pour ce qui concerne la Vue, la nuance vient du fait que nous utilisons principalement des fichiers XML, donc pour mimer le fait que la Vue soit actualisée en fonction du Modèle,



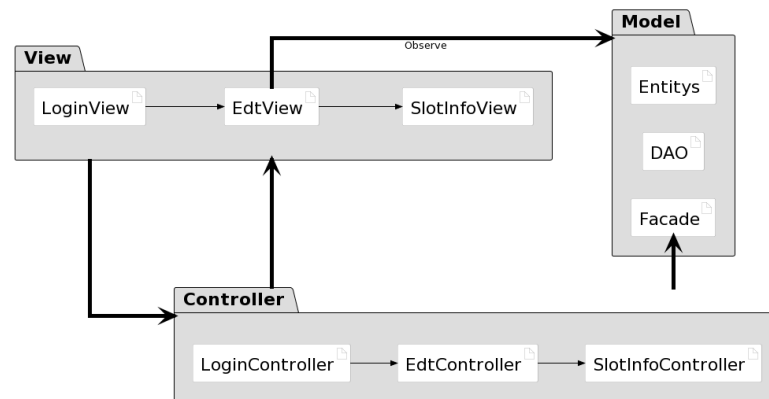


FIGURE V.1 – Modèle Vue Contrôleur

nous utilisons le paterne *Observateur / Observé* sur le Contrôleur qui observe le Modèle et actualise la Vue en fonction des changements effectués.

Finalement, pour ce qui est des données présentes sur l'emploi du temps, nous avons fait en sorte que des données soient présentes pour le mois de janvier 2023, utile pour les tests et les démonstrations sur des cas plus réels et concrets.

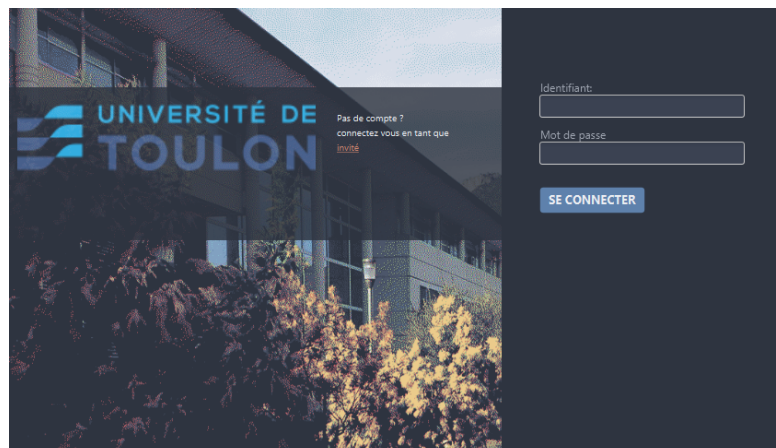


FIGURE V.2 – Vue d'Authentification

Lorsque l'utilisateur lance l'application, il fera face à la vue d'authentification, il pourra choisir de se connecter comme invité ou bien entrer un e-mail et un mot de passe pour se connecter à son compte. Ce même compte peut être un administrateur, un professeur ou un étudiant. Le procédé est différent s'il s'agit d'un invité.

L'interface du planning s'adapte donc en fonction de l'utilisateur authentifié, par exemple un Administrateur aura la possibilité de voir tous les emplois du temps d'une Promotion spécifique, mais aussi d'ajouter/supprimer/modifier des cours de celle-ci.

Autre exemple, les étudiants et les professeurs n'ont accès qu'à leurs emplois du temps respectifs, tandis que les invités peuvent uniquement visionner les emplois du temps d'une promotion à sélectionner depuis l'interface.

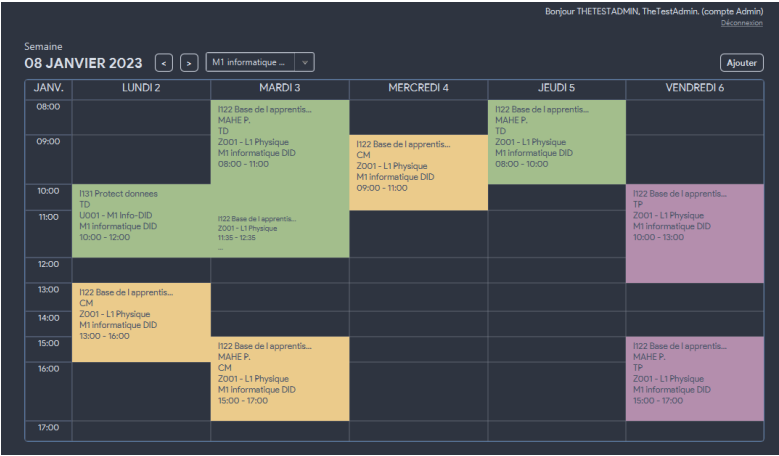


FIGURE V.3 – Emploi du Temps vu par un Administrateur



FIGURE V.4 – Emploi du Temps vu par un Étudiant

Par défaut, les cours ont un code couleur attribué en fonction du type de cours (c.-à-d. CM, TD, TP, EXAM, RÉUNION, etc).



Si la durée de ce cours est inférieure à 2 h, alors certaines informations normalement présentes seront retirées par simple choix de lisibilité, pour remédier aux manques d'informations, il est possible de sélectionner directement le cours en question pour y afficher toutes ces informations (et accessoirement les modifier si on est connecté sur un compte administrateur).

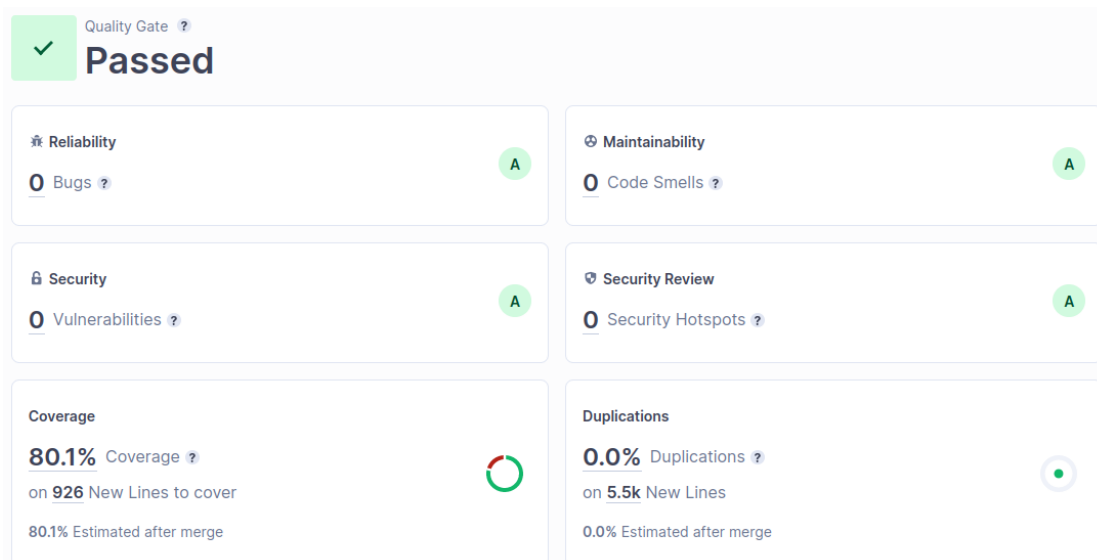
The image displays two screenshots of a web-based course management interface. The top screenshot shows a form for selecting or viewing a course. It includes dropdown menus for 'I122 Base de l'apprentissage', 'Mahe PIERRE', 'M1 informatique DID', and 'Z001 - L1 Physique'. There is a date field set to '28/12/2022' and a time range from '09:00' to '11:00'. A progress indicator shows '3 / 33h'. At the bottom are 'Ok' and 'Fermer' buttons. The bottom screenshot shows a 'MODIFIER' (Modify) form for the same course. It includes a 'Supprimer' (Delete) button. The dropdown menus show 'I131 Protect donnees', 'Mahe PIERRE', 'L1', and 'U001 - M1 Info-DID'. The date is '19/12/2022' and the time range is '08:00' to '11:00'. A progress indicator shows '0 / 61h'. At the bottom are 'Sauvegarder' (Save) and 'Annuler' (Cancel) buttons.

Ce panneau d'informations relatif aux cours permet de facilement afficher toutes les informations d'un cours sélectionné, ou bien pour l'administrateur, de facilement modifier un cours. Par exemple, admettons un cours a lieu le lundi 5 de 09 h à 10 h pour la promotion M1 DID, si l'administrateur choisit de définir ce cours pour la promotion de L1 Physique, puis sauvegarde ce changement. La façade du modèle s'occupe d'effectuer les changements sur le dit modèle, puis le Contrôleur en question fait vu d'un changement effectué sur le modèle, et actualise donc l'élément à changer sur l'emploi du temps. En l'occurrence, le cours de M1 Did, ne serait plus visible pour cette promotion, et serait visible pour la promotion L1 Physique le lundi 5 de 09 h à 10 h.

Le pattern *Observateur / Observé* ne notifie l'emploi du temps que des changements qui ont été effectués ainsi que le type du changements à appliquer sur la grille de notre emploi du temps (resp. Ajout / Modification / Suppression). Autrement dit si je change de promotion le cours du Lundi 5, la Façade effectue l'appel des daos correspondantes pour effectuer le changements en base de données, puis notifie la Façade si les changements sont bien valides, et enfin la Façade notifie tout ces *Observateur* qu'il y a eu un

changements sur le cours du Lundi 5, et ce changement est de type *Modification*, puis le contrôleur et la vue font le reste.

## V.4 Test Unitaire



Pour commencer cette section, parlons d'un étudiant en Licence d'informatique. Lors de ces études d'informatique, cet étudiant typique va réaliser un certain nombre de TP dans diverse matière, souvent sans connexion, où il doit, dans un temps limité, résoudre des courts problèmes. il y a bien des épiphénomènes, les projets, mais bien trop rare pour avoir des conséquences durable. Pourquoi cette digression ? C'est pour sous-ligner les raisons de l'absence d'éducation et le rejet envers les test-unitaire des étudiants n'est pas par fainéantise mais par expérience. En effet pourquoi tester sont code quand le prochain TP auras lieux dans deux jour, mais surtout pourquoi tester sont code quand le problème appartiens au royaume de des math, un lieu par définition absolument logique est donc predictable.

La raison est simple, la réalité est trop complexe, les individu imprévisible est par conséquent même un logicielle aussi simple que celui d'un emploi du temps requerrais un travaille gargantuesque pour un résultat incertain. c'est la le domaine de l'ingénierie. doit ton pour autant abandonner la rigueur. Non.

Heureusement pour nous, la sciences a trouver la solution il y a des siecle, les test empirique. et leur reflet direct en informatique. les test unitaire. Pour résumer nous croyons sincèrement aux rôle essentielle des test unitaire dans la développement d'un logicielle. De plus un code tester est un code bien compartimenter au couplage faible. ces pour ces raison et plein d'autre qu'une des condition essentielle de validation de nos Pull Request(cf Gesttion de projet)était le test systématique est rigoureux des méthodes ajouter.

La conséquence de ce choix est une confiance importante dans notre C.I, après tout un des éléments essentielle au jugement d'une de nos PR était ça capacité as passer tout les test précédant, et as rester dans les frontière de nos objectifs de code coverage.

## V.5 Gestion de la persistance : Base de données

Le projet visant à gérer les emplois du temps d'une université, se pose la question de la persistance des données, de leur format, des méthodes d'accès et de l'intégrité. Le sujet du projet, mentionne l'usage d'une

base de données, sans plus de détail. Très tôt dans la conception, notre choix s’est porté sur PostgreSQL, et ce pour plusieurs raisons :

1. Fiabilité PostgreSQL est connu pour sa fiabilité et sa gestion conformes aux règles ACID, et au centre d'un écosystème mature .
2. Orienté objet : PostgreSQL est un SGBD relationnel multi-modèle, et comporte des notions d'objet, dont l'héritage, sur lequel on reviendra en détail.
3. Types complexes : PostgreSQL gère des types complexes et dispose de types temporels intégrés, notamment des intervalles de temps.
4. Expérience passée : Nous avons, au cours de notre cursus, déjà été amenés à travailler avec cet outil.

La modélisation et particulièrement le nous ont permis de dégager des multiples relations entre acteurs. En somme, la donnée d'un cours peut se résumer en une phrase :

Le vendredi 25 novembre 2021 de neuf heures à midi, les étudiants du Master 1 Informatique assistent à une séance de *travaux dirigés*, dispensée par MM. MALLOFRE et LOISEAU dans le cadre du module *projet collaboratif (I143)*, en salle U.001.

Les éléments en *italique* caractérisent ce que l'on a ainsi baptisé créneau (*slot*) tout au long de ce projet. Nous avons en outre choisi de n'allouer qu'une seule salle à un créneau.

PostgreSQL gère les intervalles de temps. Cela nous permet, en pratique, de contrôler la bonne cohérence des données sans nous encombrer de détails techniques. Nous nous servons de cette facilité afin de s'assurer que deux plages horaires ne se chevauchent pas dès l'insertion.

À l'aide de déclencheurs (triggers) nous nous assurons en outre que ces contraintes sont aussi respectées pour les groupes et les enseignants concernés par les cours, avant toute insertion ou mise à jour.

Enfin, afin de faciliter la gestion de l'authentification, on a recours à l'héritage de tables, qui nous permet d'obtenir le rôle et les informations d'un utilisateur à l'aide d'une simple requête.

Finalement, on a obtenu le schéma suivant :

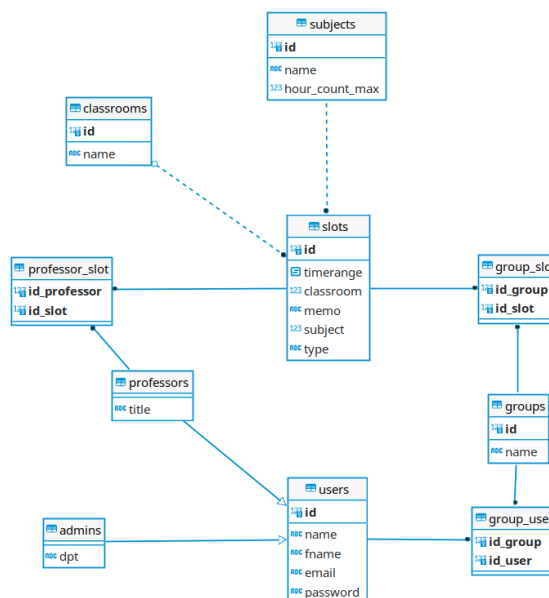


FIGURE V.5 – Modèle entité relation simplifié de la base de données

Afin de pouvoir accéder à notre base de données en java, nous utilisons le pilote JDBC approprié. Nous avons recours à un agrégateur de connexions (*connection pool*), qui nous permet de gérer les échanges à la base de données de façon efficace. Il permet en outre de gérer le cas d'une fuite de connexion (si une connexion à la base n'est pas fermée une fois le transfert terminé), ou d'une connexion interrompue (notamment le cas lorsque la base se trouve à distance).

Afin de segmenter les responsabilités dans notre application, nous utilisons une couche de DAO (Data Access Objects ) qui gère les erreurs, l'instanciation et la persistance en lien avec la base. Lors de la persistance en base, celle-ci génère et nous fournit un identifiant unique, que l'on enregistre dans les entités.

## V.6 Intégration continue

Afin de tester notre code convenablement sans les aléas des environnements des uns et des autres, nous avons jugé utile d'intégrer un système de contrôle de notre code avant son intégration au sein du projet. Des outils, comme Sonar, que nous avons adopté permettent ainsi de contrôler la qualité du code, sa robustesse, d'identifier des failles et les éventuels fragments qui ne seraient pas testés.

Pour ce faire, la seule solution qui permet un contrôle effectif consiste à contrôler le code que les tests exécutent. C'est le rôle de JaCoCo (Java Code Coverage) qui va ainsi permettre de dégager trois indicateurs principaux :

1. Le taux de couverture linéaire, qui mesure la quantité de code exécuté lors des tests par rapport à la masse du projet. Une règle de bonne pratique consiste à viser un taux situé entre 60 et 80 %. C'est à ce taux que l'on fait référence le plus souvent.
2. La couverture conditionnelle (branch coverage), qui vérifie que tous les cas soient testés lors d'une décision. C'est le cas des boucles et rupture de séquences (if, else, while)
3. La complexité cyclomatique qui estime le nombre de décisions prise par un fragment de code donné.

Il nous faut donc s'assurer que l'agent s'exécute lors des vérifications et que les résultats soient remontés à Sonar, ce qui n'est pas sans quelques déconvenues.

L'exécution se déroule sur une machine mise à disposition par Github, ce qui nous impose d'installer un serveur PostgreSQL lors des vérifications. L'utilisation d'un container docker s'est avérée trop complexe en raison de la multiplicité des acteurs en présence.

## Chapitre VI

# Gestion de projet

*Responding to change* over following a plan

— AGILE MANIFESTO

Le projet a été réalisé dans un contexte agile, plus spécifiquement avec le framework Scrum. Scrum est une application des méthodes agile, comme peut l'être aussi l'extreme programming, néanmoins cette méthode se concentre sur l'idée d'itération ou de sprint. Un sprint est une unité de temps arbitraire (un jour/une semaine/un mois) où l'on se fixe pour objectif de réaliser un nombre N de User Story, ce nombre dépend de la difficulté, de la priorité et le niveau de risque. l'idée principale étant de mesurer la capacité de travail de l'équipe afin de mieux prévoir et trouver les goulots d'étranglements. Néanmoins, même si de nombreuses méthodes et idées de Scrum étaient adaptées et bienvenues, ce n'était malheureusement pas le cas des sprints.

En effet, les sprints partent d'un postulat, la comparaison entre Unité de temps similaire, néanmoins ce postulat n'est pas raisonnable dans le contexte du premier semestre du M1. D'abord, car ce semestre est variable en temps libre avec des périodes de vacances, des périodes de cours et des séminaires d'une semaine. Mais surtout, ce Semestre est composé de 6 autres matières, ou 3 d'entre elles demandant un projet ou un équivalent, à de diverses échéances. Il est dès lors impossible de comparer le travail fourni par l'équipe sur deux semaines différentes.

Par conséquent, le choix de l'équipe a été de consigner et prévoir par itération, mais aussi et *surtout* de travailler de manière *continue* et *incremental* en acceptant la nécessité du changement.

### VI.1 Chronologie

**Itération n° 0 :** La première itération avait trois objectifs :

- ✓ Choisir et mettre en place l'environnement de travail  $\text{\LaTeX}$ , le choix du workflow `git` avec github actions et enfin la mise en place des outils de communication.
- ✓ Définir des acteurs principaux de notre système et interviews pour mieux les comprendre.
- ✓ Établir une veille d'information sur les solutions déjà existantes.

**Itération n° 1 :** La deuxième itération est un peu spéciale, en effet celle-ci a été marquée par une absence de cours. par conséquent le choix fait fut celui d'une continuation travaillée de l'inception.

- ✓ Scission des UC, à partir des interviews de la précédente Itérations

**Itération n° 2 :** La troisième itération fut très courte, 4 jours, par conséquent la majorité des modifications provient du feedback de Sopra

Correction des UC à partir des remarques de Sopra

- ✓ Première version d'un diagramme de classe (*information view*)
- ✓ Réalisation d'une *Buisness view*

**Itération n° 3 :** La quatrième itération fut le moment de la création des *User Story*, et de leur chiffrage il s'agissait aussi du moment de la création du diagramme de kanban sur github.

- ✓ User story et chiffrage
- ✓ Création du diagramme de kanban

**Itération n° 4 :** La cinquième itération est le moment où le développement de l'application a vraiment commencer. avec le début de l'implantation des classes avec leur package respectif. avec comme toujours, les tests. il s'agit aussi du début des tests en condition réelle de l'I.C, et de nos premiers pas avec le workflow Github

- ✓ Implantation classe des packages User et Slot
- ✓ Mise en pratique du workflow Github

**Itération n° 5 :** La sixième itération fut centrée autour de la Base de données, un travail précédant avait été fait en termes de conception, mais l'implantation fut complexe, c'est aussi par conséquent le début du développement de la DAO.

- ✓ Première version de la base de données
- ✓ Commencement du travail sur la DAO

**Itération n° 6 :** La septième itération est la prolongation directe de la cinquième avec un travail sur la DAO est sur la Base de Données, c'est aussi le moment où fassent à l'impératif de temps important une étude sur la viabilité du framework *CalendarFx* as été .

- ✓ Travail sur la base de données
- ✓ Continuation du travail sur les DAO
- ✓ Etude des choix pour la GUI

**Itération n° 7 :** La huitième itération est l'itération de la première IHM, avec l'abandon de *CalendarFX* est l'adoption de *JavaFX*, avec aussi des modifications sur la DAO avec le choix de *Hikari* pour la gestion des pools de connections.

- ✓ Première IHM
- ✓ Continuation du travail sur les DAO avec Hikari

Fin du travail sur la base de données

**Itération n° 8 :** La neuvième itération est la première pendant les vacances, pas sans contrainte, mais un peu plus flexible. c'est le moment où les DAO et la base de données furent terminées, avec une première tentative de connexion entre les deux avec une façade. Néanmoins, cette période fut aussi celle d'un bug sur l'Intégration Continue.

- ✓ amélioration de IHM
- ✓ fin du travail sur les DAO et la base de données
- ✓ Implantation de US1

**Itération n° 9 :** La dixième itération as aussi eu lieu pendant les vacances, elle avait pour objectif de préparer la réalisations de **toute** les US la semaine suivante

- ✓ fix de I.C
- ✓ continuation travail d'implantation avec les US2 US3 US4 US5

**Itération n° 10 :** La Onzième et dernière itération qui avait évidemment pour objectif de finir ce projet



- ✓ rédaction rapport et manuel
- ✓ Fin de de l'implantation avec la US6

FIN DU PROJET  
9 Janvier 2023

## VI.2 GitHub flow

Le choix d'un workflow ou manière de travail est un élément essentiel à toute gestion de projet. Chacun des membres de notre équipe a vécu un projet avec une répartition du travail arbitraire, individuel et en parallèle avec une réunion du travail catastrophique à la fin.


Bien sûr, un fix simple à ce problème est *Git*, néanmoins il ne s'agit que d'un outil et comme tous les outil, la manière de l'utiliser est au moins aussi, sinon plus, importante que l'outil lui-même. Nous avons, en cours, étudié l'un des workflow les plus connus, le *feature flow*, ce workflow, bien qu'efficace et familier de la majorité de notre équipe, présente une faiblesse ennuyante. Il est en effet difficile de lire le travail des autres. D'abord un développeur peut travailler sur plusieurs Feature, donc branche, et ensuite et surtout le travail peut être merger de manière unilatérale et sans vérification de toute l'équipe.



Le GitHub flow est une solution à ces problèmes, principalement utiliser sur les dépôts Open Source de Github, ce workflow est particulièrement adapté à un travail en groupe, puisqu'il est la base de projet avec plusieurs milliers de contributeurs. Bien sûr, notre groupe n'a pas besoin d'une telle scalability, non, l'intérêt principal qui nous à pousser vers ce choix est la sécurité et la communication qu'ils entraînent. Ce workflow est en effet caractérisée par une certaine rigidité des branches, cette rigidité découle de l'importance des *pull request* (PR).

Ces PR sont un véritable hub de discussion du travail en cours, avec des fonctionnalités comme les draft, les commentaires, le support de markdown et enfin et surtout des *code review*. Des review avec un twist, puisque que nous avons décidé que toutes les branches seront accepter à l'**unanimité**, pour rendre les choses plus claire, ci-dessous un exemple du workflow.

### VI.2.1 description du workflow

#### Expand the number of exceptions #78

 Theo-Hafsaoui opened this issue yesterday · 1 comment

 Theo-Hafsaoui commented yesterday · edited · Member 

As of now we only have one exception, which make the readability of our code poor. we should try to expand the number of exception to solve this issue, here are some proposal  
DAO exception:

- EntityNotFound: fail to find entity  
msg: Might be because the object \$ClassOfObject is not persisted see if id is greater than 0 id(\$idOfObject)
- FailedToSaveEntity: fail in saving  
msg: Might be because the entity failed to satisfied an unicity constraint
- FailedToUpdateEntity: failure in update  
msg: Might be because the object \$ClassOfObject is not persisted see if id is greater than 0 id(\$idOfObject) or it might be because the entity failed to satisfied an unicity constraint
- FailedToLinkTwoEntity: for join table  
msg: Might be because one if the entity is not persisted see ID1(\$idOfObject1) and ID2(\$idOfObject2)

Entity exception:

- IException: No change

Facade exception:

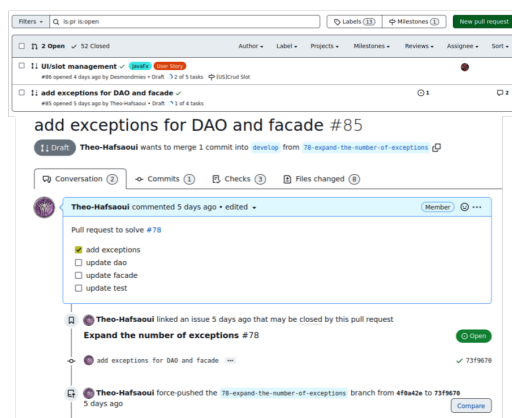
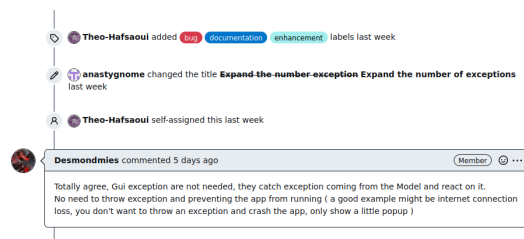
- NotEntityNotFound: because if the getter return null that's the same msg
- AddressException: if the format of the email is not correct from the official java email package
- SomeParameterAreNotPersisted: if a parameter is not the database  
msg: You need to persisted these entity:\$listOfNotPersistedEntities

Gui exception:

-Might be contentious but I believe, None. Do you agree @anastynome @Desmondmies

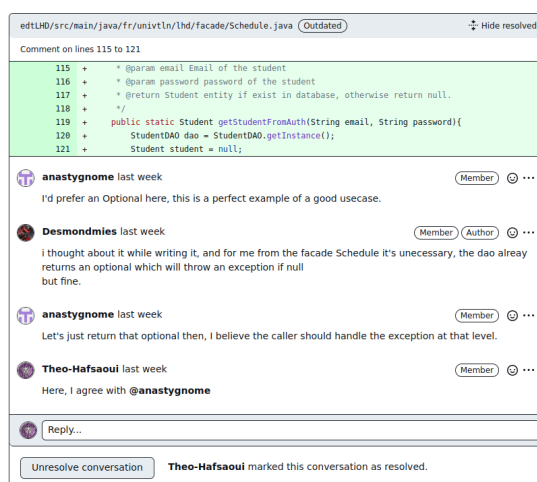
Etape 1 :La première étape de notre méthode de travail est l'ouverture d'une *issue*. En général cette issue est le résultat d'une réunion est et toujours accompagner par plusieurs autres. Certaines issues sont des *meta-issue*, des issues tellement grosse qu'il serait dangereux et surtout bloquant de les réalisées en une seule Pull Request. Par conséquent, ces Issues ne sont réalisées que si un autre nombre d'issues sont résolue auparavant, github permet d'ailleurs de le faire automatiquement

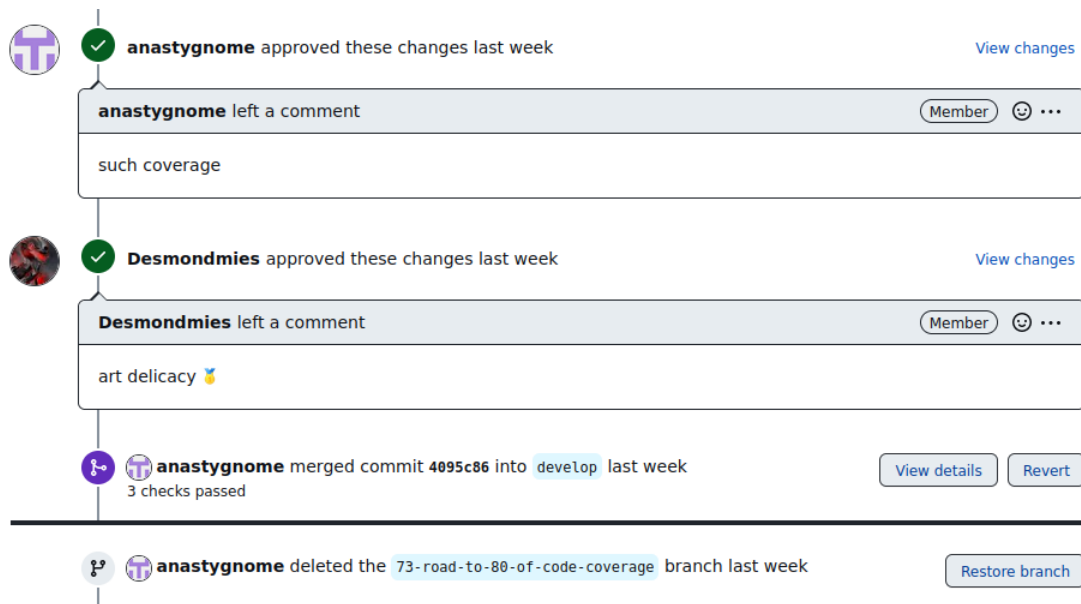
**Etape 2 :** La deuxième étape est celle de la communication, en effet l'issue n'est qu'une proposition amenée naturellement à évoluer en fonction des remarques et contraintes des membres de l'équipe. On retrouve en général une décomposition du travail, sous forme de check-liste et parfois des diagrammes UML pour enlever les ambiguïtés, ces diagrammes sont d'ailleurs toujours présents sur les issues des User Story



**Etape 3 :** La troisième étapes est l'ouverture de la pull request, c'est là aussi un hub de discussion, mais d'une nature différente de l'issue. Là où la priorité des discussions d'une issue sont de lever les ambiguïtés et définir les objectifs, les Pull Request elle dans leurs discussions se concentrent sur l'aspect technique, performance, lisibilité, documentation et bien sûr les tests unitaire.

**Etape 4 :** La quatrième étape est la review, en effet les discussions précédentes étaient sur une Pull Request en mode *Draft*, c'est-à-dire qu'il était sur un travail en cours. La review est différente, car si les discussions sur le draft sont facultatives, la review est absolument obligatoire. Cette review peut finir par deux résultats différents. Le premier résultat est bien sûr, l'acceptation de la PR par toutes les membres de l'équipe, il s'agit du résultat le plus commun grâce aux discussions au moment des rétrospectives et le travail de fond réalisé dans les issues. Le Deuxième résultat assez commun lui aussi est la demande de modification, les raisons sont diverses, mais en général il s'agit d'un problème de cohérence avec le reste du code ou une optimisation.





La dernière étape est bien sûr l'acceptation de la PR et le rebase, avec en général un squash, sûr développe.

Certains points sont tout de même à noter :

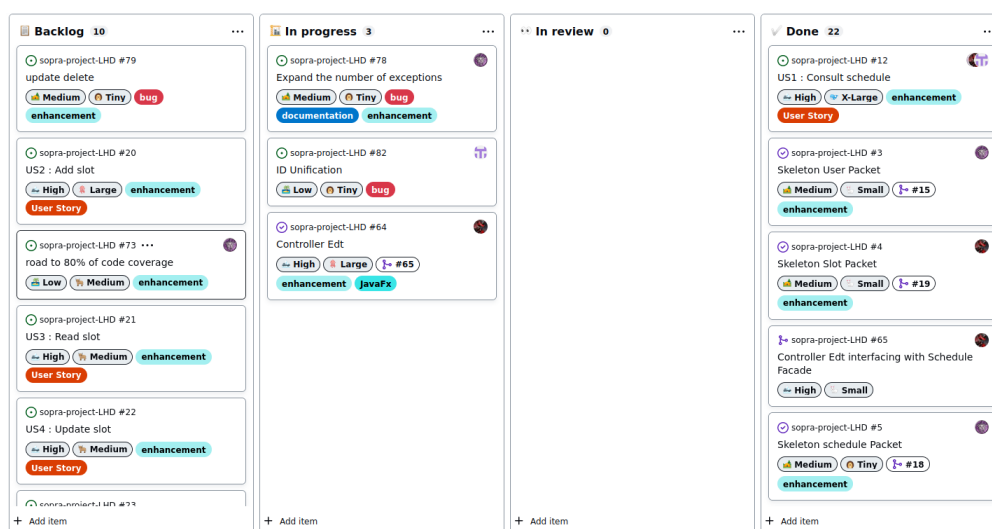
- *Anglais* : Le choix de la langue sur Github n'est pas un choix anodin, mais une véritable volonté de professionnaliser notre projet. Après tout l'anglais n'est pour aucun d'entre nous notre langue natale, un effort de concisions des idées étaient donc nécessaires pour pouvoir communiquer dans nos Pull request
- *Github* Le choix de la plateforme de Microsoft découle de l'intégration entre les différentes briques de notre projet, par exemple la création d'une issue crée un ticket dans le kanban et sa résolution faisait glisser ce ticket dans la bonne colonne.

### VI.2.2 limite

Comme évoquait précédemment ce workflow offre de nombreux avantages, mais il n'est pas sans inconvénients.

- D'abord il s'agit d'une méthode lourde. En effet, chaque PR est soumise à la disponibilité des membres de l'équipe, par conséquent l'indisponibilité paralyse l'avancement dans le cas d'une PR critique.
- Il ne s'agit pas d'une faiblesse du workflow github, en général, mais dû notre spécifiquement. en effet, dans un souci de lisibilité, chaque PR ou presque à était squash, à un ou plusieurs commits. rendant la lecture de l'historique plus facile, mais surtout fausse. ce qui peut donner une vision biaisée de l'historique.

### VI.3 Kanban



Bien sûr, un workflow n'est pas suffisant, la production d'un logiciel est un art complexe, particulièrement lorsque l'on souhaite paralléliser les tâches. À cette fin, c'est naturellement que nous nous sommes dirigés vers un autre outil que Github propose le diagramme de kanban.

**Le principe :** Ce diagramme se décompose en trois grandes parties. Todo (À faire), In progress (en cours), Done (fait). Par exemple, si on reprend notre workflow vu précédemment, l'ouverture d'une issue est la création d'un ticket qui va dans la colonne Todo (Backlog car nous utilisons Scrum). L'ouverture d'une PR nous amènent directement dans la colonne In Progress et enfin la validation de la PR fini dans la colonne Done.

Simple et souvent associé à Scrum il est néanmoins très différent dans son approche du temps, qui ici est continue au contraire de Scrum qui lui à une approche itérative. Cette différence est sans doute pour nous l'intérêt essentiel de ce diagramme. En effet, comme expliqué dans l'introduction de ce chapitre, suite aux multiples contraintes de ce semestre, le choix de notre équipe à était d'abandonner le format itératif de Scrum pour une version continue. Par conséquent, un risque important était le manque de vision global du projet, après tout, il s'agit d'une des forces de Scrum avec sa capacité de prévision. Kanban est une solution directe à ce problème, s'il n'est pas aussi efficace pour prédire, il est absolument parfait pour s'assurer de la cohérence et de la synchronicité du travail de chacun des membres de l'équipe.

Commencer par le Kanban est néanmoins légèrement mensonger, car il est au final le résultat du travail réalisé sur le backlog et la sprint table. Ces deux tableaux, on était réalisés à l'initiative des intervenants et fut indispensable dans la priorisation des tickets, ce fut aussi pour l'équipe l'occasion de confronter nos prévisions, une étape essentielle, car avec l'abandon des itérations notre capacité de prédiction devait être le plus aiguisée possible.

Sprint	User Stories	Chiffrage 1 jour = 2h			
<b>0</b>	dépôt git	1			
	modélisation bdd	3			
	installation postgre	1			
	<b>Total</b>	5	Reste	10	
<b>1</b>	Consulter edt	15			
	<b>Total</b>	15	Reste	0	
<b>2</b>	Lire créneaux	4			
	Filtrage Cours	4			
	Ajouter créneaux	3			
	Modifier créneaux	2			
	Supprimer créneaux	2			
	<b>Total</b>	15	Reste	0	
<b>3</b>	Auth	3			
	Consulter EDT particulier	2			
	Lire usager	3			
	Ajouter usager	2			
	Modifier usager	1			
	<b>Total</b>	11			
<b>4</b>	Supprimer usager	1			
	Progression cours	2			
	<b>Total</b>	3			

Catégorie	User Story	Priorité	Chiffrage 1 jour = 2h		
CRUD Créneaux	Setup env	P0	5		
	S'authentifier	P3	3		
	Ajouter un créneaux	P2	3		
	Lire un créneaux	P1	4		
	Modifier un créneaux	P2	2		
	Supprimer un créneaux	P2	2		
CRUD Usagers	Ajouter un usager	P4	2		
	Lire un usager	P4	3		
	Modifier un usager	P4	1		
	Supprimer un usager	P4	1		
	Consultation EDT Extérieur	P1	15		
	Consultation EDT particulier	P3	2		
	Progression Cours	P5	2		
	Filtrage Cours	P2	4		

Ref Priorité					
P0					
P1					
P2					
P3					
P4					
P5					
...					

Néanmoins, malgré tout les avantages de ces tableaux, ceci était par définition statique et lourd à entretenir. Heureusement pour nous, github propose une solution. Peut-être ne l'avons-nous pas assez souligné précédemment, mais le passage d'une colonne à l'autre est absolument automatique, ce qui ici est une solution à notre problème de tableaux statique, et en effet dans le reste du projet, nous utilisons le Kanban *mais* avec différent filtre, en effet github permet d'appliquer des labels de taille et de priorité pour ensuite les trier par ces paramètres. Devant ces features, il à donc était naturelle pour nous de complètement basculer sur Github, mais sans abandonner les sprints Table juste en les convertissant de manière naturelle dans notre workflow.

[illegible]

## VI.4 Analyse & conclusion

Juger de la gestion d'un projet, est une tâche presque impossible tant les raisons du succès d'un projet sont difficile à cerner. D'ailleurs la définition de succès est profondément subjective, particulièrement pour nous qui n'avons aucun clients à satisfaire, par conséquent et en pleine consciences de la subjectivité de cette affirmation, nous déclarons que ce projet est un succès.

Maintenant dire que ce projet est un succès ne nous avance pas beaucoup sur l'analyse de notre gestion de projet. Certain pourrais même dire qu'un échec aurais rendu l'analyse plus simple est nous aurais éviter une exposition inutile au biais du survivant. Néanmoins nous croyons que nos expérience passer nous permette une certaine objectivité sur la gestion du projet et sur ces forces.

1. **Communication** : Il nous semble que la force de notre gestion de projet se trouve dans l'importance accordée à la communication entre ces membres. L'exemple le plus frappant de ce choix est sans doute les code reviews, et l'accord unanime qu'exigent une PR, un choix plus que rare et qui découle directement de nos expériences des projets passés où la racine de nos soucis était la communication.
2. **Continue** : comme dit dans l'introduction de ce chapitre l'un des choix fait dans l'inception du projet est l'abandon de l'aspect itératif de Scrum au profit d'une vision plus continue. ce choix directement traduit par le diagramme de kanban, ce diagramme nous a permis une grande flexibilité et agilité sur notre capacité à répondre au changement. Ce qui par exemple nous a permis de rattraper notre retard sur l'IHM après le choix de calendarFX

Néanmoins certaines limites sont à noter de ces points forts. D'abord la communication est une épée à double tranchant, certes le risque d'erreur, d'incompréhension et de duplication du travail est réduite mais au détriment d'une forte rigidité et par conséquent d'une lenteur dans l'ajout de nouvelle US. Il nous a par exemple fallu attendre la dixième itération pour réaliser la première US. L'intégration Continue bien que à la frontière entre Implantations et notre Gestion de projet, et pourtant une part essentielle de notre manière de travailler pour ce projet et si les avantages indéniables ont été démontrés plus tôt dans ce rapport, cette IC n'est pas sans désavantages dans notre workflow puisque son fonctionnement est une étape critique pour la vision et la fluidité de notre gestion de projet. Ainsi malheureusement pendant 2 semaines suite à des conflits avec PostgreSQL, un poids supplémentaire était ressenti dans notre organisation.

En conclusion, nous nous accordons pour dire que l'aspect le plus intéressant de ce projet est sans doute la gestion de celui-ci et sommes très satisfaits du résultat et connaissance acquise dans ce projet.

## Chapitre VII

# Conclusion analyse & ouverture

En guise de conclusion, cette aventure de 2 mois nous a inculqués de nombreuses compétences, que ce soit côté *Java* ou côté *Gestion de Projet*. La découverte de *JavaFX* et la structuration d'un projet complet à plusieurs nous a apporté une réelle plus-value en ce qui concerne *Java*. Enfin grâce à la supervision de Laurent LOISEAU & Jérémy MALLOFRE durant les différentes semaines dédié au projet et à sa gestion, nous avons concrètement appliquer les principes agiles, cela nous a permis de mieux nous organiser pour les tâches à effectuer et terminer le projet dans le délai imposé de 2 mois. En guise de rétrospective, un projet de cette ampleur aurait pris bien plus de temps et aurait été moins bien organisés sans leur aide. Comme dit précédemment notre *workflow Github* à également été une décision majeur quant à l'organisation du projet, et réussir à s'en tenir jusqu'au terme du projet est un accomplissant et des compétences acquises en plus pour chacun d'entre nous. Les seuls point faibles à retenir serait peut être, le début de projet légèrement tardif, ce qui a forcé le gros travail de développement sur la période de fin d'années, en groupe et à distance, ce qui n'est jamais évident à gérer. Malgré tout nous avons réussi à utiliser les outils à notre porté pour rester dans les temps et rendre un produit fini en adéquation avec la demande client, laissant évidemment possibilités d'améliorer ou changer certains aspects de l'application selon les retours du client.

Comme par exemple des améliorations côté interface :

1. **Message d'erreur** : A l'heure actuel certains messages d'erreur ne sont pas encore affichés par l'interface pour donner un retour direct à l'utilisateur, si une action à provoquer une erreur la plupart du temps il ne se passera rien. L'application continue comme si rien ne c'était passé, une amélioration évidente est de prévenir l'utilisateur du problème actuel, que ce soit un soucis en base de données par exemple l'insertion d'un cours à un créneau indisponible, ou tout simplement la perte de connexion avec la base de données.
2. **Rendre l'application parfaitement Responsive** : Un problème majeur lors de l'intégration de *JavaFX*, le fait que l'application devrait être totalement responsive est un chose agréable à avoir pour l'utilisateur. Or actuellement certains cours sont bloquant pour la grille de l'emploi du temps, ce qui rend l'affichage pour de petit écran très complexe voir impossible. La solution idéale serait de rendre l'application compatible Smartphone ainsi que Pc, pour couvrir un large éventail de taille d'écran et donc assurer le côté responsive de l'application.
3. **Drag & Drop** : Un bon point de facilité d'utilisation pour l'administrateur, pouvoir directement glisser les cours sur l'emploi du temps pour rapidement changer leur créneaux ou bien les glisser d'un groupe à l'autre ou d'une semaine à une autre.
4. **Multi-Sélection** : Encore un autre point très ergonomique pour l'administrateur, pouvoir sélectionner plusieurs cours, puis effectuer des actions sur chacun d'entre eux, tous les supprimés, modifier des champs, etc.

5. **Template** : L'utilisation de template de cours pour faciliter l'ajout répété de cours similaire, par exemple un écran de type de cours préfait, pour permettre à l'administrateur de direction sélectionner un cours pré-rempli et modifier quelques champs pour l'adapter à son besoin, un simple ajout mais qui serait fort utile pour de longue session de gestion.

Certains aspects plus technique quant aux améliorations évoqués pour ce projet sont déjà documentés sur notre *RoadMap* associées. En outre, ce fut une expérience remplie de concept et de notion rarement traités, une bonne expérience qui nous apportent compétences et connaissances pour la gestion de nos projets futurs ainsi que la structuration d'un projet en prévoyant son ampleur et en chiffrant ses différents aspects afin de le mener à bien dans le temps imparti.



## **Annexes**

## Table des figures

II.1. Diagramme de cas d'usage . . . . .	4
III.1. Diagramme de classes des entités . . . . .	10
IV.1. Diagramme de séquence . . . . .	14
IV.2. Diagramme d'états . . . . .	15
V.1. Modèle Vue Contrôleur . . . . .	19
V.2. Vue d'Authentification . . . . .	19
V.3. Emploi du Temps vu par un Administrateur . . . . .	20
V.4. Emploi du Temps vu par un Étudiant . . . . .	20
V.5. Schéma de la B.D. . . . .	23