Projet Tranverse I - Checkpoint 1

Kerfalla Cissé, Nikita Missiri

24 mars 2022

Table des matières

1 Introduction

La durée totale des études en Systèmes d'informations et science des services étant de six semestres au minimum et de dix semestres au maximum, les étudiants en orientation interdisciplinaire sont dans l'obligation de choisir des cours à option afin de compléter les 60 crédits restants pour obtenir le baccalauréat universitaire. En effet, 114 crédits sont acquis avec les cours obligatoires plus 6 crédits provenant d'un cours obligatoire en mathématiques.

2 Description du projet et sa problématique

Choisir des cours à option peut facilement être une tâche qui se fait avec peine pour les étudiants, car il faut, d'une part, identifier les cours qui pourraient les intéressés parmi tous les cours disponibles à l'université et, d'autre part, vérifier que le suivi de ces cours puisse être possible en fonction de leurs horaires obligatoires du semestre. Le but du projet est donc de répondre à cette problématique en créant un service accessible sur une plateforme web, qui propose des cours à option pour le semestre aux étudiants du bachelor et qui génère un calendrier en fonction du cours choisi. Le calendrier contiendra en plus du cours à option choisi, les cours obligatoires du semestre.
La proposition des cours à option sera entièrement basée sur les horaires de disponibilité des étudiants et

de leur profil (centre d'intérêts, cours à option déjà suivis, année et période d'étude courante). Quelque soit l'année et le semestre, les cours obligatoires du bachelor n'apparaîtront pas dans la liste des cours à option proposé par la plateforme.

3 Cas d'utilisations

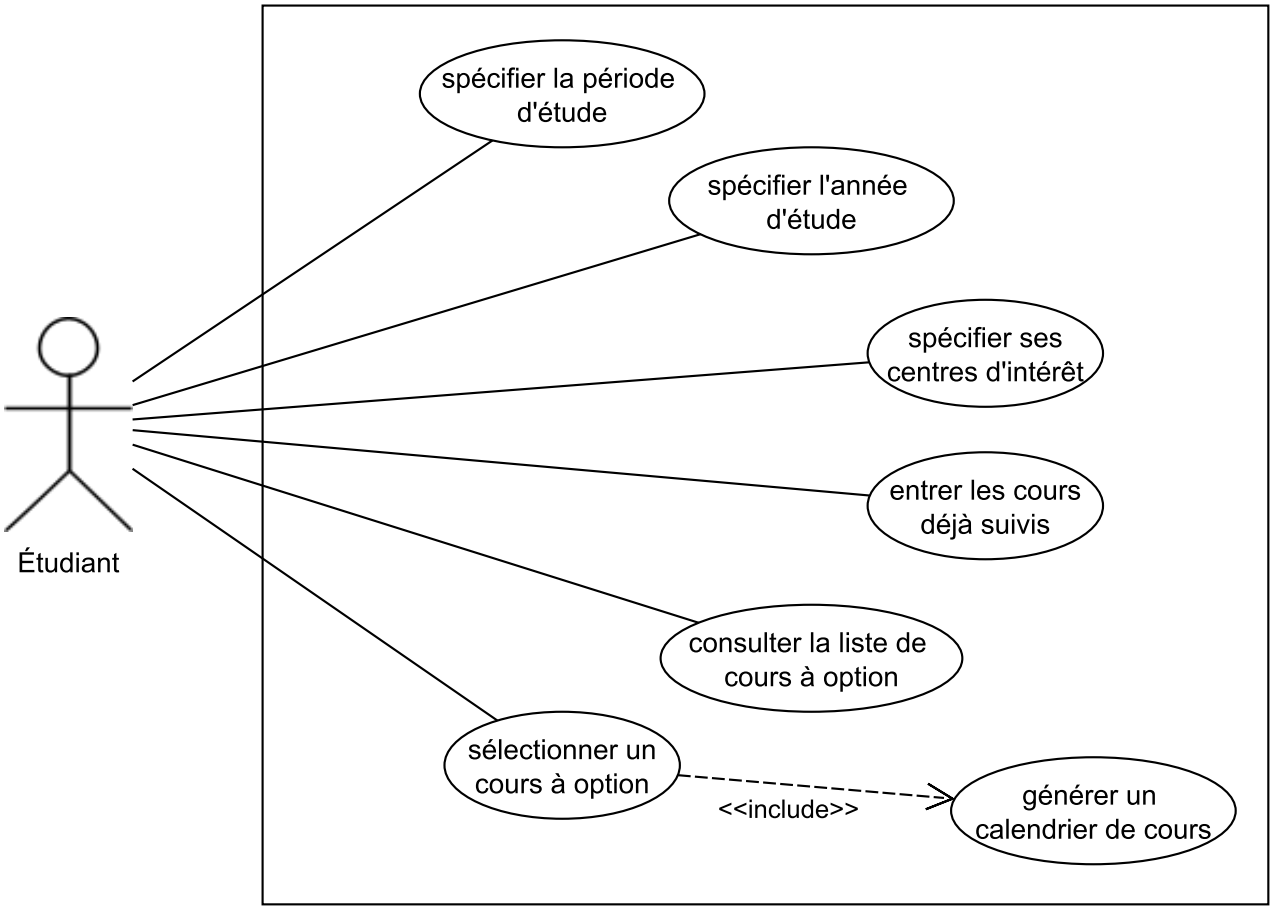

Figure 1: Cas d'utilisations

Ce diagramme des cas d'utilisation permet de délimiter et identifier les différentes fonctionnalités que le système doit avoir afin de satisfaire les besoins fonctionnels des utilisateurs. Le but étant que l'utilisateur (l'étudiant) puisse obtenir une liste de cours à option pour le semestre basé sur ses horaires de disponibilités et son profil. Pour ce faire, l'étudiant doit pouvoir entrer l'année d'étude et le semestre qu'il suit actuellement (exemple : semestre de printemps ou automne en 2ème année), ce qui détermine quels cours obligatoires seront suivis par l'étudiant durant ce semestre, en se basant sur la séquence de cours recommandée par le Centre Universitaire d'Informatique. Cela permettra également de définir quels sont les horaires de disponibilité de l'étudiant pour le semestre. Il pourra également spécifier les cours à option qu'il a suivi les semestres précédents à travers leur identifiant et ses centres d'intérêts par mots clés. Ce qui sera utile pour la création de son profil. À partir de ces paramètres, le système doit fournir une liste de cours à options, trié par ordre de recommandation et par la suite permettre à l'étudiant de sélectionner un cours à option. La sélection du cours à option va automatiquement générer (relation include) un calendrier qui contient en plus du cours à option choisi, les cours obligatoires du semestre. Cela permettra à l'étudiant d'avoir un aperçu de son planning en choisissant ce cours. Il est important de mentionner que quelque soit l'année d'étude et le semestre, les cours obligatoires du bachelor n'apparaîtront pas dans la liste.

4 Analyse des objectifs (besoin fonctionnels et non-fonctionnels)

4.1 Modélisation des objectifs

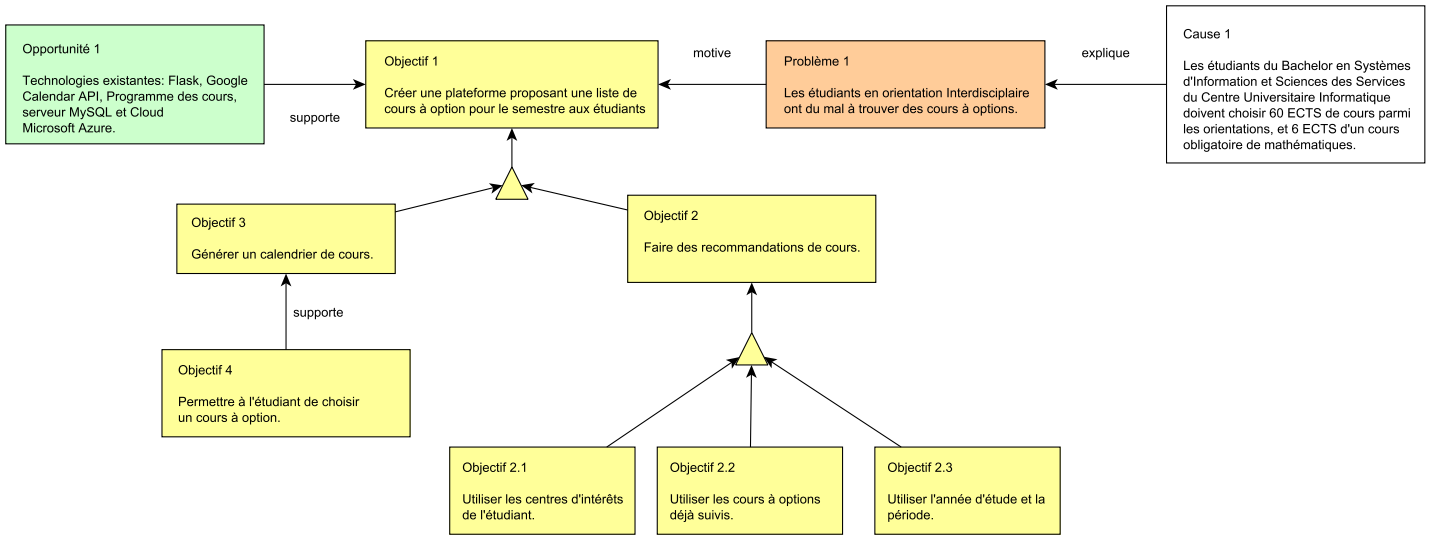

Figure 2: Modèle des objectifs

Le diagramme ci-dessus représente le modèle des objectifs. Il a pour but d'identifier, explorer et évaluer les différents objectifs que la plateforme vise à atteindre. Il fournit également divers solutions qui permettront de satisfaire l'objectif principal (Objectif 1).

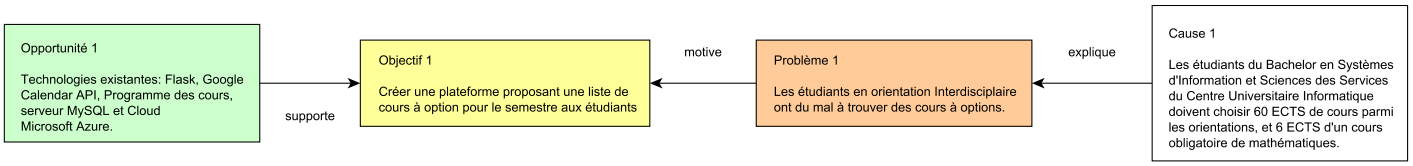

Figure 3: Motivation de l'objectif

Cette partie du diagramme reprend les différents éléments qui ont été mentionnées dans les sections 1 et 2. De plus, il identifie les différentes technologies et services web qui peuvent être utilisés pour la création de la plateforme (voir opportunité 1).

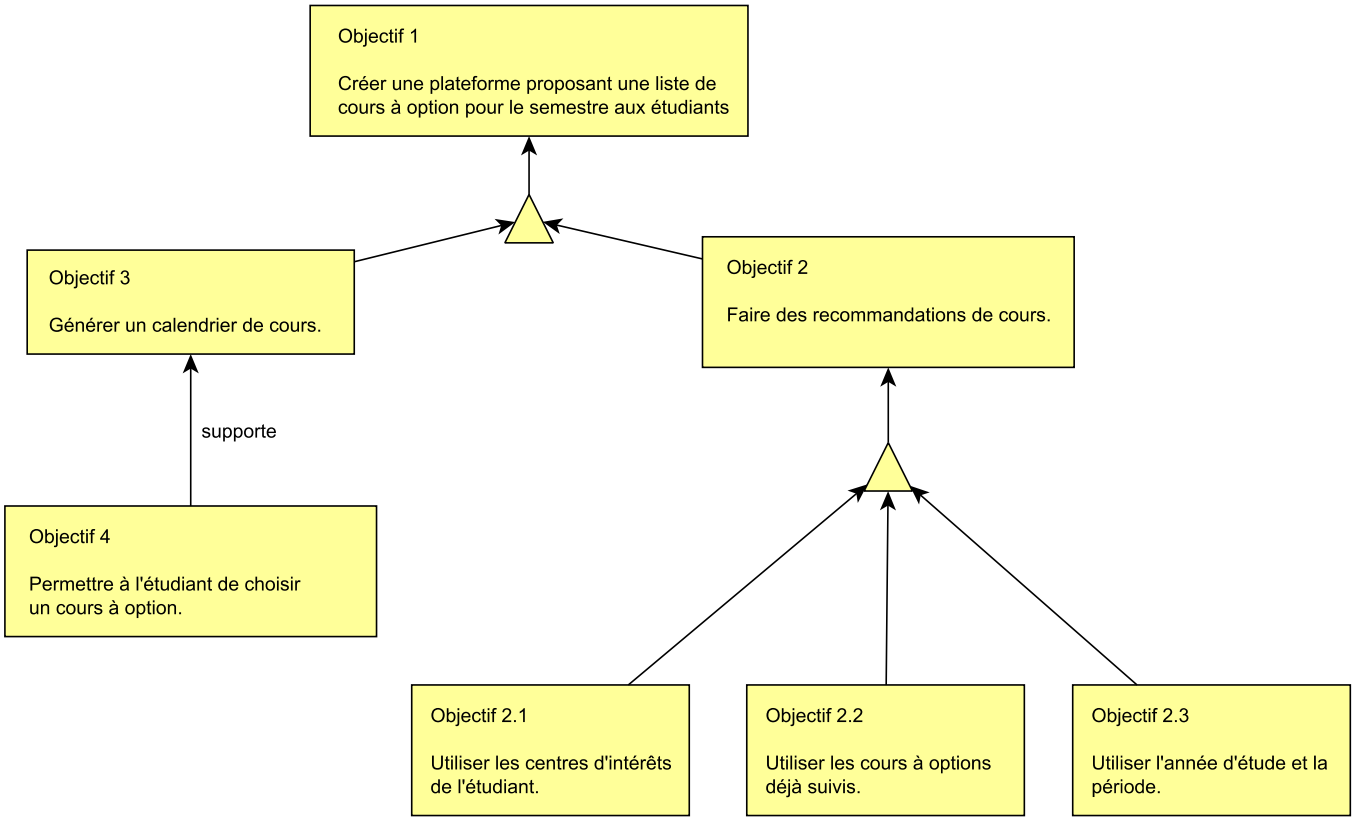

Figure 4: Les sous-objectifs

La figure ci-dessus illustre les différents sous-objectifs qui permettront de satisfaire l'objectif principal. Les décompositions de type ET définissent un ensemble de sous-objectifs qui sont nécessaires à la réalisation des deux super-objectifs (objectif 1 et 2).
La plateforme doit pouvoir générer un calendrier de cours (objectif 3) en fonction du cours à option choisit par l'étudiant (objectif 4). Le choix d'un cours par l'étudiant va dépendre des recommandations de cours fournis par la plateforme (objectif 2) qui elles même dépendent des résultats des objectifs 2.1, 2.2 et 2.3.

4.2 Modélisation des exigences fonctionnels et non-fonctionnels

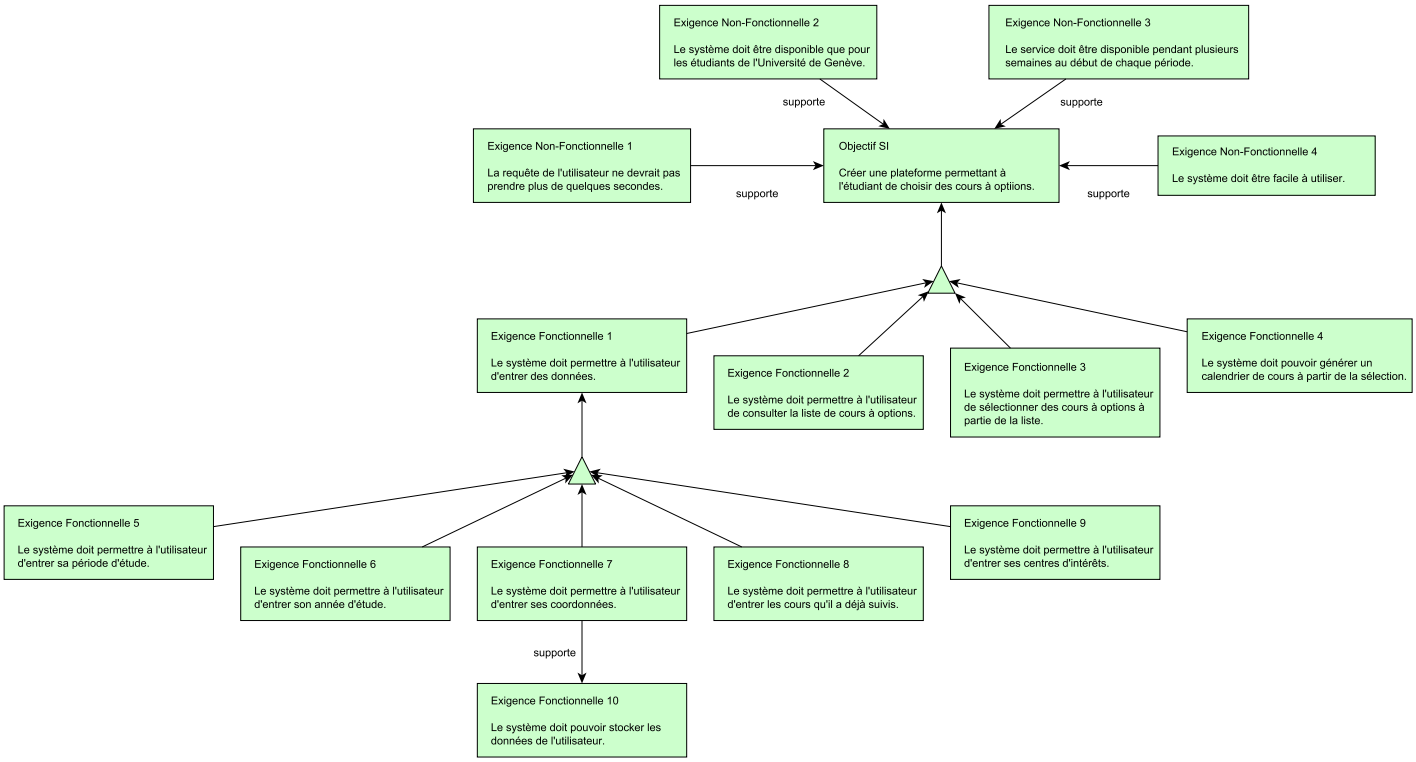

Figure 5: Modèles des exigences fonctionnels et non-fonctionnels

Afin de connaître les besoins fonctionnels et non-fonctionnels du système, une modélisation des exigences est construite à partir de l'objectif principal. Les exigences peuvent être identifiés à l'aide des cas d'utilisations. Le diagramme ci-dessus illustre le modèle des exigences.

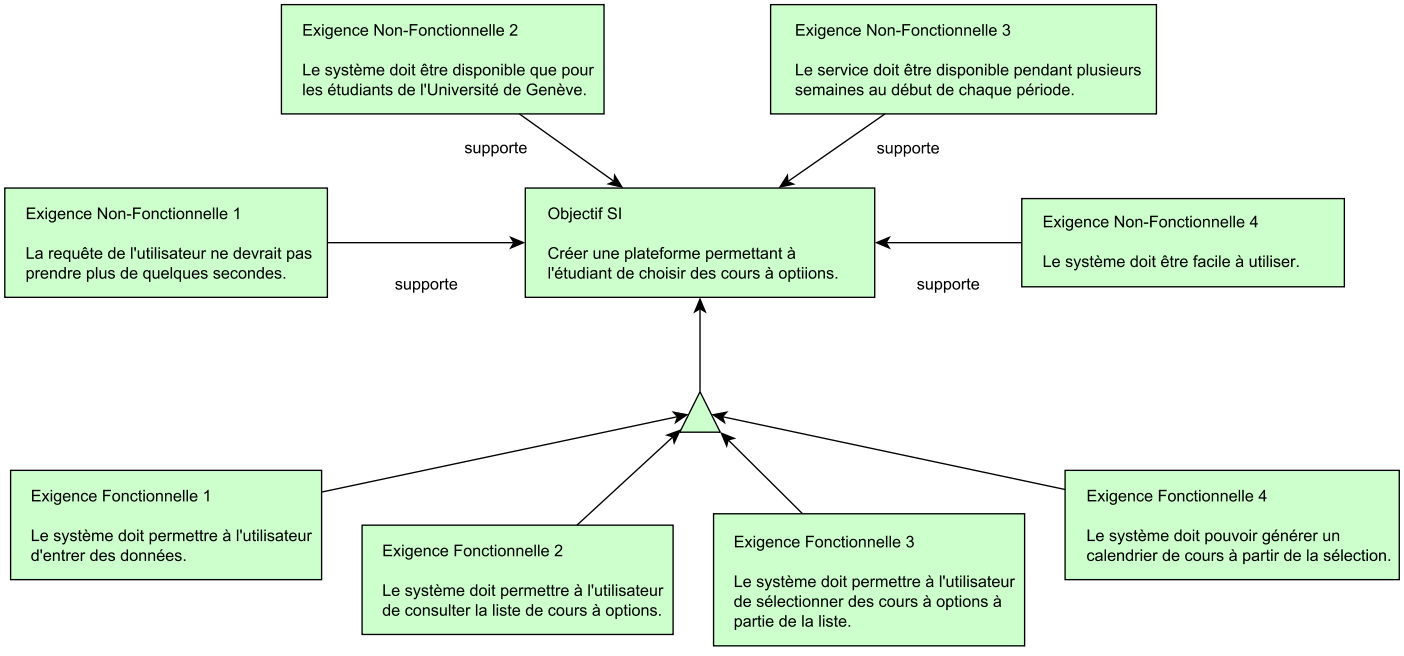

Figure 6: Les exigences liés à l'objectif principal

Pour créer une plateforme qui aide aux étudiants à choisir leurs cours à option, il faut que le système récupère des données entrés par l'utilisateur (Exigence Fonctionnelle 1). Ces données permettent ensuite

de générer une liste de cours à options (Exigence Fonctionnelle 2). Les cours sur cette liste doivent être cliquables par l'utilisateur (Exigence Fonctionnelle 3). Une fois cliqué sur un cours, le système génère un calendrier de cours avec les cours obligatoires du semestre courant et le cours sélectionné (Exigence Fonctionnelle 4). Nous pouvons identifier des exigences non-fonctionnelles qui viennent supporter l'objectif principal. Premièrement, le système doit être performant, c'est-à-dire que la requête de l'utilisateur, que ce soit lors de l'exécution de la liste des cours à options ou lors de la génération de calendrier de cours, ne doit pas prendre plus de quelques secondes (Exigence Non-Fonctionnelle 1). Pour une question de sécurité, le système ne doit être disponible qu'aux étudiants de l'Université de Genève, cela peut être fait avec une authentification Switch edu-ID (Exigence Non-Fonctionnelle 2). Le système doit être disponible pendant plusieurs semaines au début de chaque période, puisque ce seront les périodes durant lesquelles le système connaîtra un trafic important d'utilisateurs (Exigence Non-Fonctionnelle 3). Le système doit être facile à utiliser, c'est-à-dire que l'utilisateur doit pouvoir réaliser ses tâches sans difficultés (Exigence Non-Fonctionnelle 4).

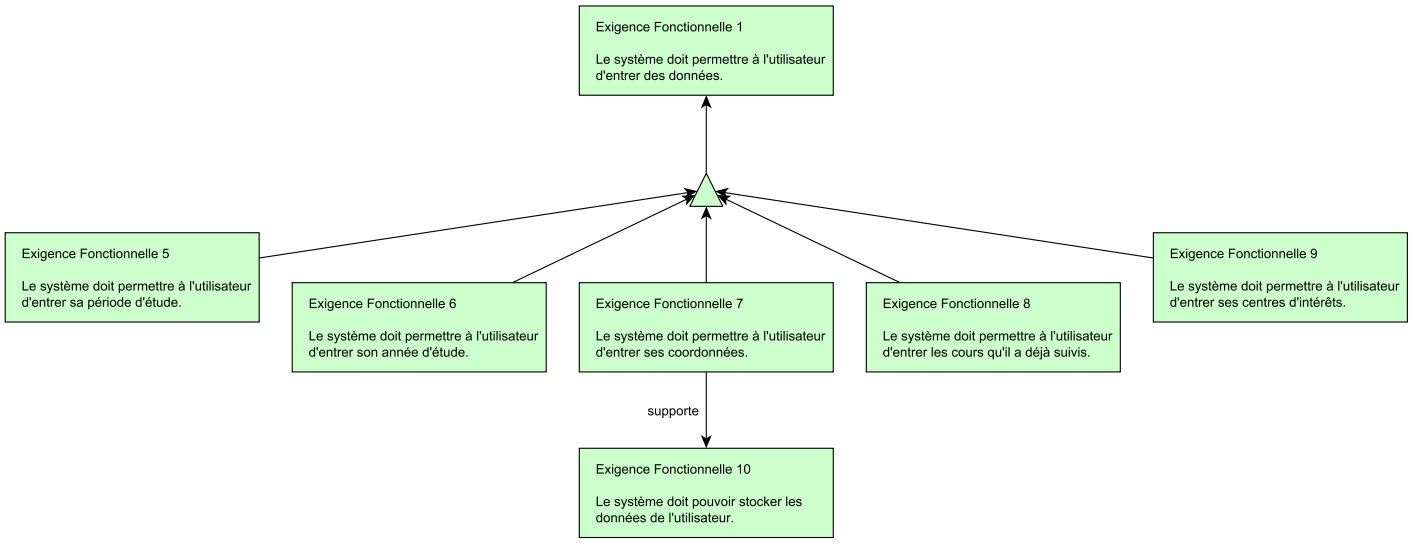

Figure 7: Les exigences fonctionnels liés à l'entrée des données de l'utilisateur (Exigence 1)

Pour réaliser l'exigence fonctionnelle qui est de permettre à l'utilisateur d'entrer des données, le système doit permettre à l'utilisateur d'entrer plus particulièrement son année d'étude, que ce soit la première, deuxième ou troisième année (Exigence Fonctionnelle 6), sa période d'étude qui est soit Automne, soit Printemps (Exigence Fonctionnelle 5), les cours qu'il a déjà suivi jusqu'à présent (Exigence Fonctionnelle 8), ses centres d'intérêts (Exigence Fonctionnelle 9), ainsi que ses coordonnées (Exigence Fonctionnelle 7). Cette dernière permettra notamment de stocker les données de l'utilisateur dans le système, et donc elle supporte le fait que le système doit pouvoir stocker ces données (Exigence Fonctionnelle 10).

5 Conception et Modélisation de la base de données

5.1 Le schéma conceptuel entités-relation

Le schéma conceptuel de la base de données est représentée de la manière suivante:

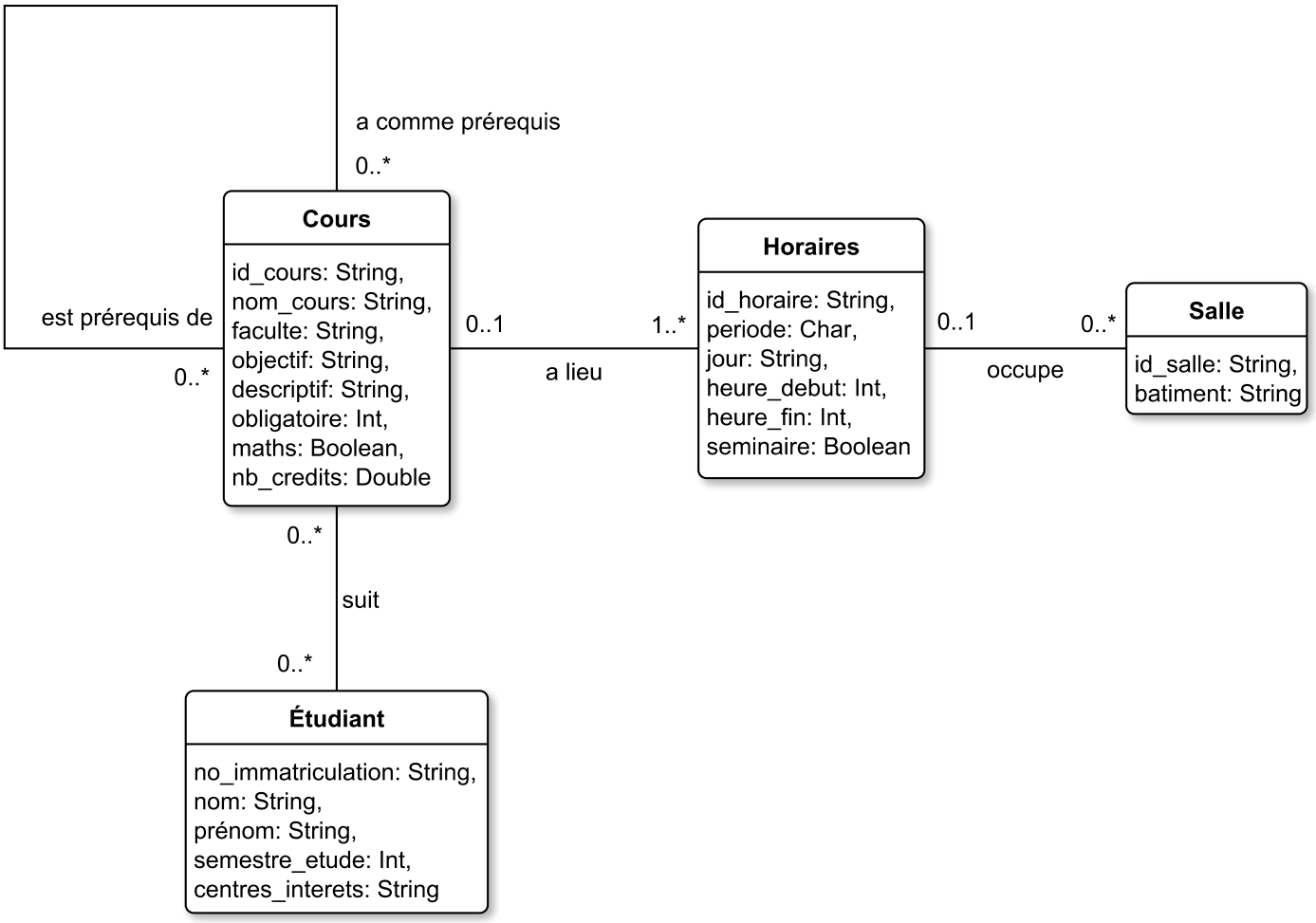

Figure 8: Schéma conceptuel

Nous avons pu identifier quatre classes pour le schéma conceptuel.

- Pour commencer, nous avons besoin des données de cours, puisque nous souhaitons afficher les cours qui pourraient intéresser l'utilisateur de notre système. On créé alors la classe *Cours*. Les données que nous considérons pertinentes à afficher sont le nom du cours, la faculté auquel il est rattaché, la description du cours (les objectifs et descriptifs) et le nombre de crédits (qui est un nombre réel, puisqu'il existe des cours avec des moitiés de crédits). Afin que chaque instance soit unique, l'OID (Object ID) est représenté par le code du cours (ici, `id_cours`). Comme les cours obligatoires, les cours de mathématiques obligatoires et les cours à option suivent tous la même structure, il a été décidé de regrouper ces trois catégories en une classe, mais de rajouter deux attributs supplémentaires:
  - un qui détermine si c'est un cours obligatoire. Cet attribut est un entier, parce qu'on stocke le semestre pour lequel il est recommandé à l'étudiant en Bachelor en Systèmes d'Information et Sciences des Services de le suivre, ce nombre allant de 1 à 6. Dans un premier temps, on fait supposition que l'étudiant suit ses études pendant 6 semestres. Les cours obligatoires de mathématiques et les cours à option auront 0 comme valeur d'attribut. Ceci permettra de filtrer non seulement les cours obligatoires, mais également de récupérer les cours obligatoires à suivre en fonction de la période dans laquelle se trouve l'étudiant, et ainsi déterminer les cours que pourrait prendre l'étudiant sans chevauchements, et d'en construire un calendrier de cours.
  - l'autre qui détermine si c'est un cours de mathématiques obligatoire. Cet attribut est tout simplement un booléen, qui retournera faux pour tous les cours, sauf pour les six cours qui sont considérés comme des cours de mathématiques obligatoires. Grâce à cela, il sera facile

de déterminer si l'étudiant a déjà choisi un cours de cette catégorie, sans lui en demander explicitement. Dans le cas où l'étudiant n'a pas sélectionné au moins un de ces cours, ces cours apparaîtront en premier dans la liste de recommandations.

Un cours peut avoir aucun ou plusieurs prérequis. Une association réflexive a donc été identifié.

- Étant donné qu'un cours peut être donné à plusieurs moments dans la semaine, nous avons décidé de modéliser la période dans laquelle un cours a lieu comme une classe à part, nommé *Horaire*, qui a comme attributs la période de l'année (A pour Automne ou P pour Printemps; dans la majorité des cas, un étudiant ne prend pas de cours annuels, mais pour la simplicité, ces cours seront séparés en deux), le jour de la semaine, l'heure de début et de fin, ainsi qu'un attribut booléen permettant de savoir si c'est un séminaire ou alors un cours. Ce dernier attribut permet d'assouplir la restriction des chevauchements s'il y a plusieurs séminaires indépendants dans la semaine, pour lesquelles la participation à un d'entre eux suffit, comme c'est le cas pour des cours avec des centaines d'étudiants. Chaque horaire est représenté par un ID unique, composé de la période, jour, heure de début et de fin. Afin d'éviter des problèmes, comme illustré plus loin, les instances de classe Horaire doivent être unique pour chaque cours, et donc le code cours est inclus dans l'ID de l'horaire. Par conséquent, puisqu'une instance de classe *Horaire* dépend également d'une instance de classe *Cours*, la multiplicité maximale sera alors de 1, car pour une période donné, un cours peut avoir lieu ou pas.
- À chacune de ces horaires, un cours est donné à une salle différente. Toutefois, la salle est identifiée comme une classe à part, qu'on a nommé *Salle*, car il existe des cours qui sont donnés dans plusieurs salles différentes à un même moment. C'est le cas notamment des séminaires pour des cours avec beaucoup d'étudiants. La multiplicité minimale a été établie à 0, pour modéliser les possibilités où la salle n'est pas indiquée pour un cours donné, ou qu'il ait lieu en ligne. Toutefois, une salle peut soit être occupé, soit ne pas être occupé pendant une certaine période, mais il ne peut jamais être occupé pour plus qu'un cours à la fois.
- Tout cela permet de modéliser facilement les cours et de générer un calendrier de cours, qui contient à la fois les cours obligatoires du semestre et le cours que l'utilisateur aura sélectionné. Afin de stocker les informations entrées par l'utilisateur, il faut une classe *Étudiant*. Cette classe a comme attributs le nom, le prénom et le numéro d'immatriculation de l'étudiant. Ce dernier est fourni par l'Université de Genève et joue le rôle de l'OID pour la classe. Pour chaque étudiant, des informations supplémentaires sont stockés, notamment le semestre d'étude qu'il suit (allant de 1 à 6), ainsi que ses centres d'intérêts. Les intérêts sont simplement stockés en type String, avec chaque intérêt séparé par une virgule. Nous avons jugé que par rapport aux objectifs de notre service, il n'était pas primordial que les intérêts soient stockés en tant qu'attributs composites.

5.2 Explications de la conception par des exemples

Ce qui suit sont des exemples de données que nous avons sélectionnés pour voir si la modélisation choisie fonctionne pour tous les cas.

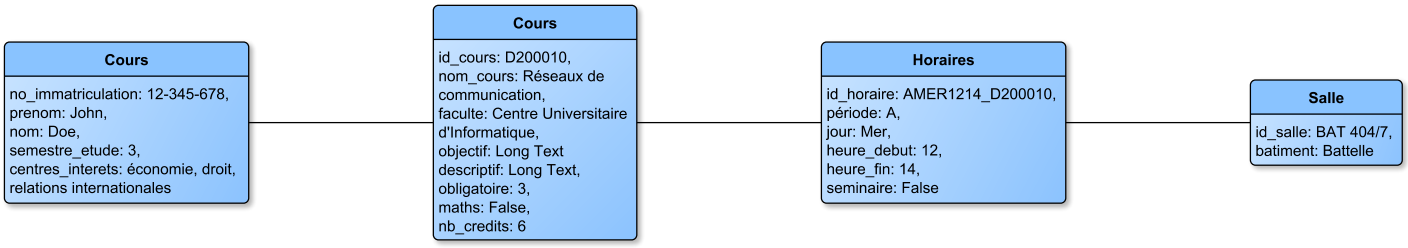

Figure 9: Exemple de cas standard

Dans ce premier cas, nous avons un étudiant qu'on a nommé John Doe, qui est au troisième semestre d'études (semestre d'Automne de la deuxième année d'études). Cet étudiant, suit automatiquement le

cours *D200010 Réseaux de communication*, donné par le Centre Universitaire d'Informatique, qui est un cours obligatoire de troisième semestre. Ce cours a lieu une fois dans la semaine, le mercredi après-midi en automne et ce n'est pas un séminaire. Ce cours est donné dans la salle BAT 404-407 à Battelle. C'est un exemple de cas courant dans notre base de données.

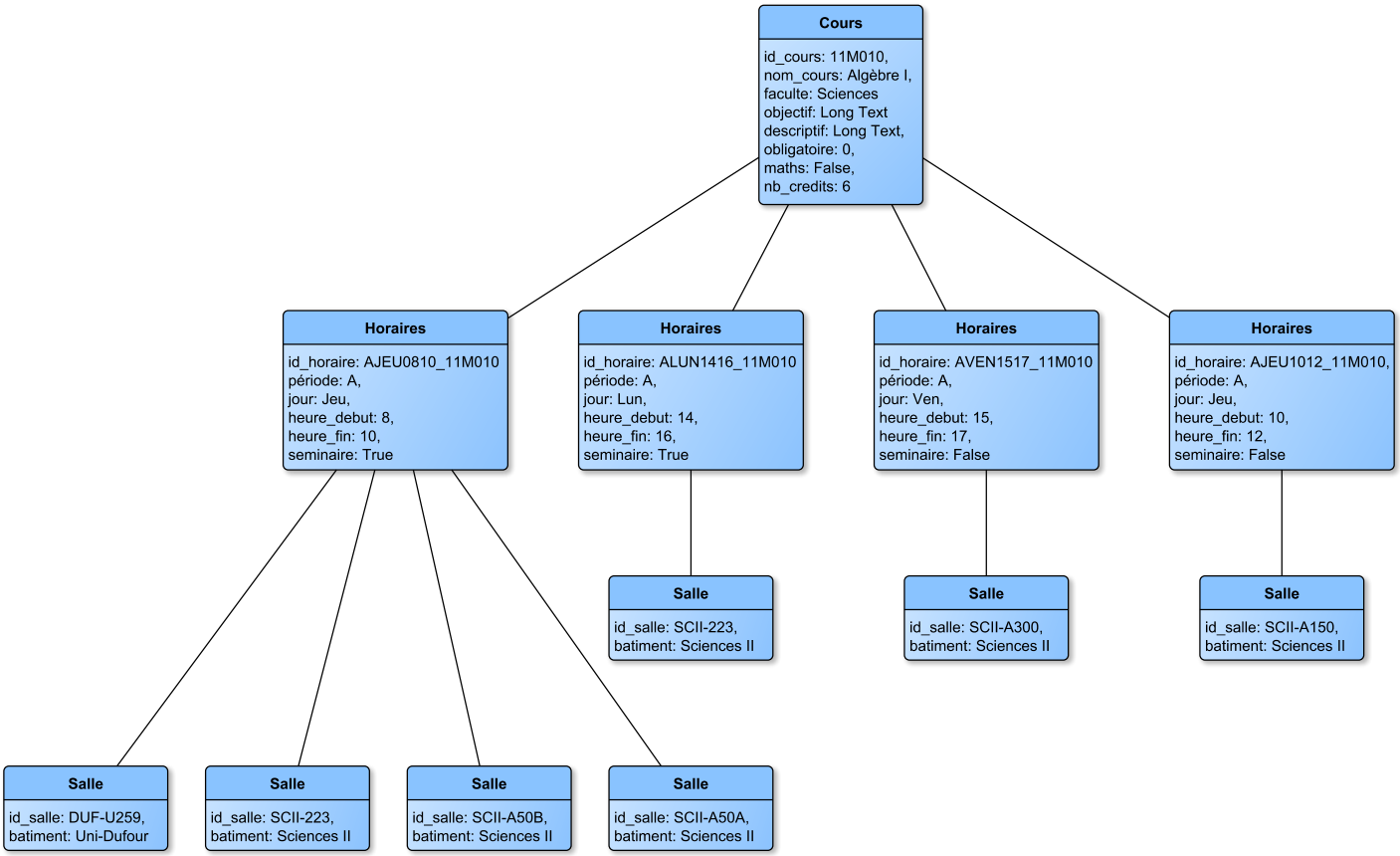

Figure 10: Exemple de cas extrême

Prenons maintenant un exemple extrême: celui du cours *11M010 Algèbre I*, donné par la Faculté des Sciences. Ce cours est donné plusieurs fois dans la semaine, dans plusieurs salles différentes. Premièrement, le cours lui-même est donné le jeudi matin en salle SCII-A150 et le vendredi après-midi en SCII-A300, une autre salle. Deuxièmement, cinq séminaires liés aux cours sont donnés dans la semaine. Un d'entre eux est donné le lundi après-midi, et les autres sont donnés en même temps le jeudi matin dans quatre salles différentes. On constate avec ce cas extrême que la modélisation choisie génère une arborescence des données.

Pour expliquer pourquoi on associe au code unique à chaque horaire le code d'un cours, prenons l'exemple suivant:

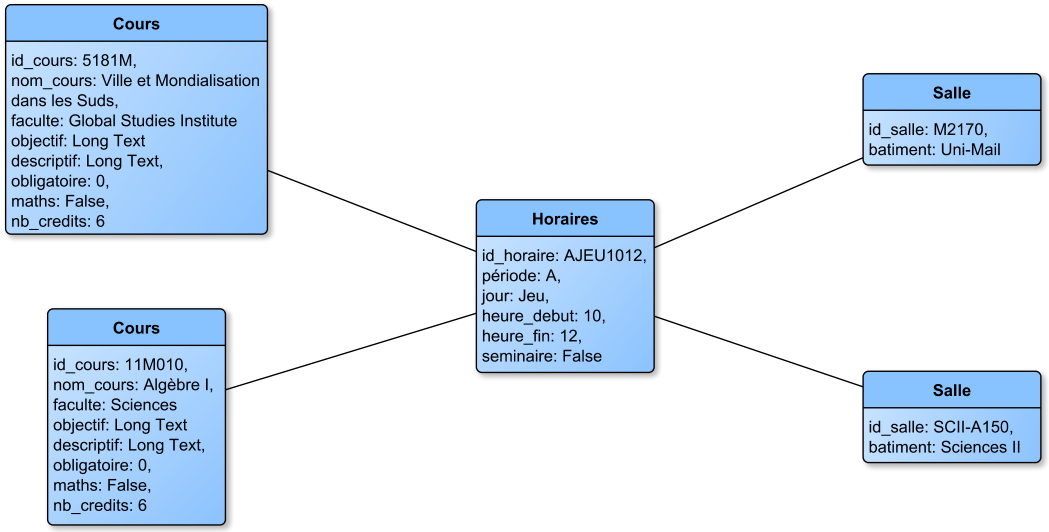

Figure 11: Explication des multiplicités (Problème)

Dans le cas où l'horaire est commun aux deux cours, il est nécessaire d'avoir une multiplicité maximale infinie du côté de la classe *Cours* (dans l'association entre *Cours* et *Horaires*), étant donné que pour un horaire donné, plusieurs cours peuvent être donné à ce même moment. Le problème surgit lorsqu'on souhaite connaître la salle pour un cours. En effet, lorsqu'on fait, par exemple, une requête SQL avec des jointures de tables pour connaître la salle où a lieu le cours *11M010 Algèbre I* le jeudi matin, plusieurs réponses ressortent, y compris la bonne qui est SCII-A150, mais aussi toutes les salles qui sont occupés dans cette même période, comme M2170, alors qu'elle est occupée pour le cours *5181M Ville et Mondialisation dans les Suds*.

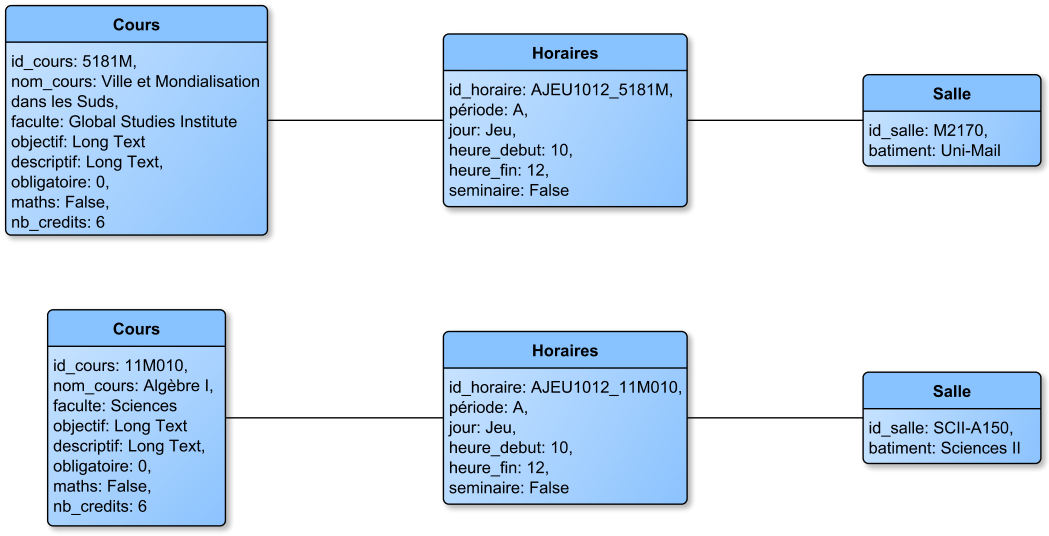

Figure 12: Explication des multiplicités (Solution)

Afin de recourir à ce problème, il vaut mieux créer des instances de classe *Horaires*, qui sont unique à chaque cours. Il y aura alors qu'un seul cours associé à chaque horaire, et donc la même requête ne retournera qu'une seule réponse, qui est la bonne.
Les salles, en revanche, peuvent être associés à plusieurs instances de la classe Horaire sans causer de problèmes, comme illustré ci-dessous:

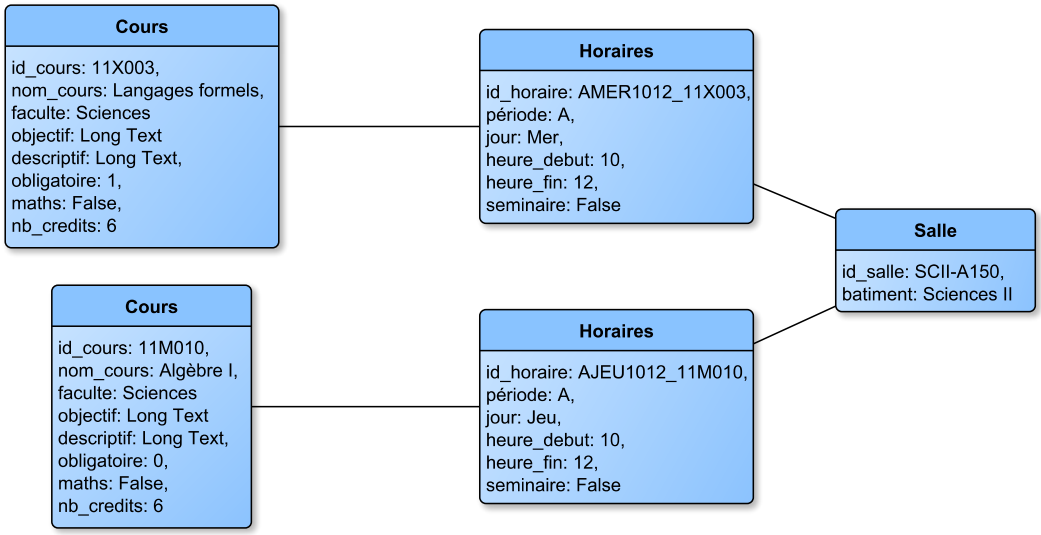

Figure 13: Exemple de salles identiques

Lorsqu'on fait une requête SQL dans un sens, pour savoir dans quelle salle a lieu le cours de *11X003 Langages formels*, par exemple, on obtient bien une seule et unique réponse. Dans l'autre sens, si on veut connaître les cours qui ont lieu dans la salle SCII-A150, les réponses sont aussi correctes.

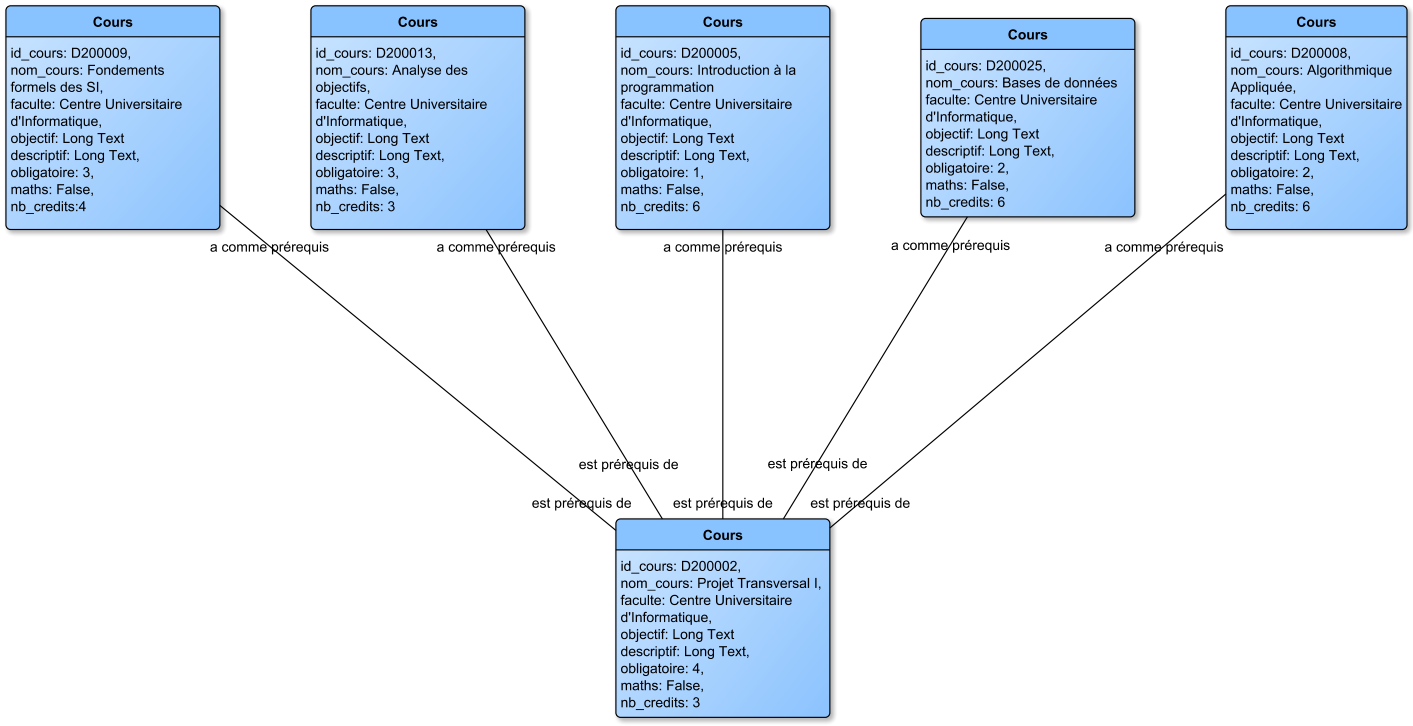

Figure 14: Exemple de prérequis

Concernant la prise en charge des prérequis pour la recommandation de cours, il faut voir l'ensemble de cours comme un graphe orientée acyclique non connexe, où les noeuds sont des instances de cours et les arcs représentent le fait que le premier cours est un prérequis du deuxième. L'exemple ci-dessus illustre bien ce propos.

5.3 Schéma logique de la base de données

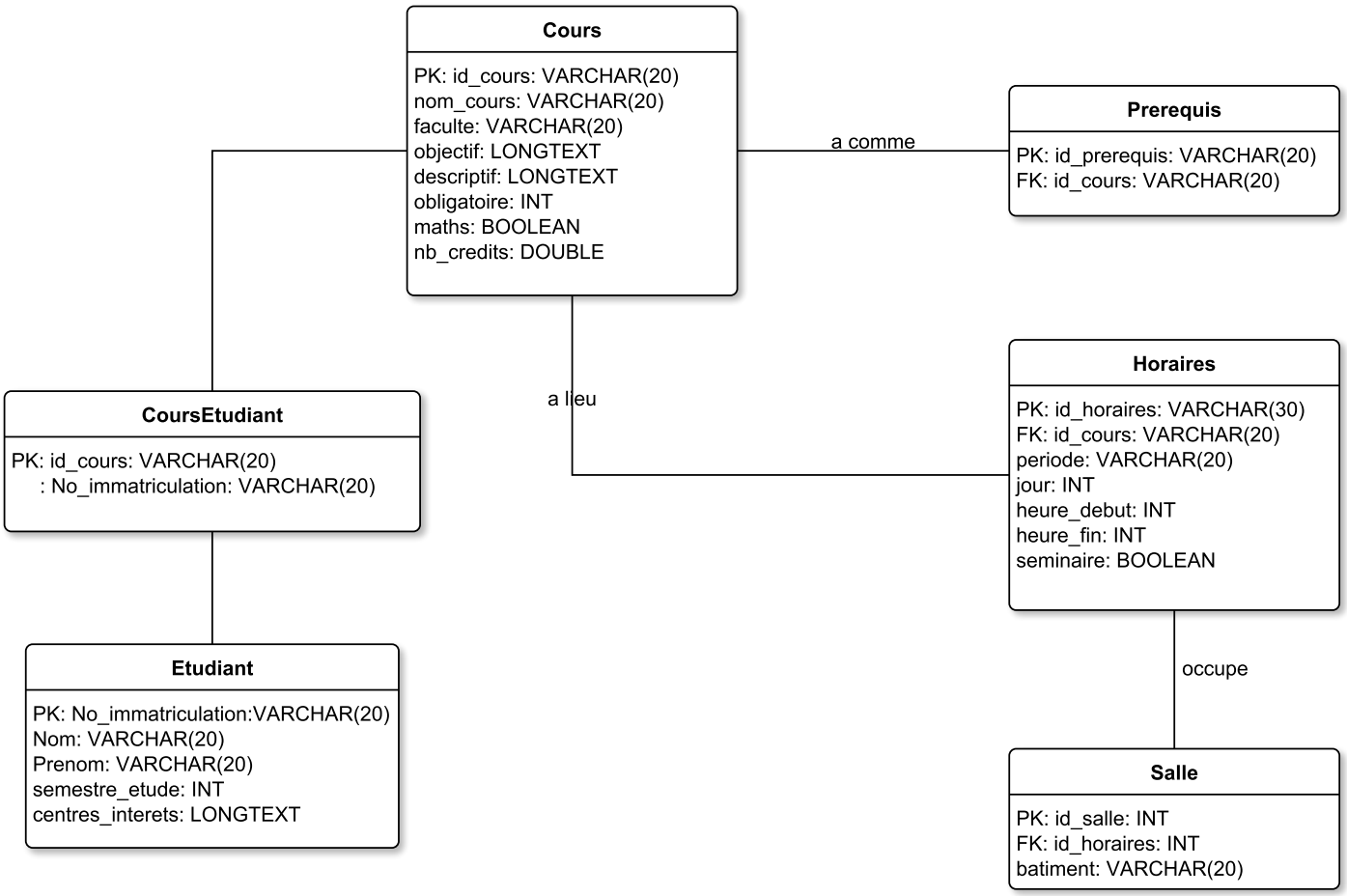

Figure 15: Schéma logique

Le schéma logique a été construit manuellement à partir du schéma conceptuel dans lequel on a identifié les différents éléments (classes) du domaine d'interprétation. La traduction du modèle conceptuel au modèle logique s'est fait en suivant les règles suivantes (voir la figure ci-dessous):

maxC / maxD	1	> 1
1	Ajouter la clé de C ou de D (à choix) comme attribut supplémentaire de l'autre relation	Ajouter la clé de D comme attribut supplémentaire de la relation C
> 1	Ajouter la clé de C comme attribut supplémentaire de la relation D	Créer une nouvelle relation R ayant comme attributs la clé de C et la clé de D

Figure 16: Règles de traduction du schéma conceptuel au schéma logique

Pour ce qui concerne la boucle sur la classe *Cours* dans le schéma conceptuel, elle a été traduite comme suit: nous avons crée une table *Prerequis* dans le schéma logique, qui contient comme clé primaire *(idprerequis)* qui correspond en fait à l'identifiant du cours qui a comme prérequis le cours possédant l'identifiant *idcours* (clé étrangère). On peut donc par la suite interroger la table *Prerequis* pour obtenir la liste des cours qui sont des prérequis d'un cours.

6 Identification des données de test

Les données de tests ont été recueillies depuis l'API du programme des cours de l'UNIGE situé à l'adresse suivante:
https://wwwit.unige.ch/cursus/programme-des-cours/api/teachings/find?academicalYear=2021&page=0&size=5000. Ces données seront utilisés pour la production du service.

7 Outils et service web utilisés

Pour ce projet, nous avons utilisé:

- **GitLab** - plateforme de développement collaboratif.
- **Python 3.10** - langage de programmation open source.
- **Visual Studio Code** – éditeur de code sur lequel on peut ajouter des extensions.
- **Flask** – framework permettant de créer des applications Web avec Python.
- **HTML, CSS et JavaScript** – pour l'interface utilisateur.
- **MySQL Workbench et Datagrip** – qui sont des EDI conçues pour travailler avec les bases de données et SQL.
- **Microsoft Azure** – service cloud qui permet de créer un serveur de base de données pour MySQL. Les différentes tables de la base de données ont été hébergé sur ce serveur.
- **API: Google Calendar** – service Web qui fournis un calendrier.
- **API: programme des cours UNIGE** – service Web qui fournit le programme des cours de l'UNIGE au format JSON.
- **Overleaf / LaTeX** – pour la rédaction des rapports.

8 Conclusion

En conclusion, nous pouvons dire que le projet est réalisable grâce à une conception bien défini de la base de données, du modèle des d'exigences et l'utilisation des divers outils et services web mentionnés dans la section précédente.