NESTOR

Explorer le programme à option d’une autre façon

André da Glória Santiago, Marie Fasel, Sébastien Franzone, Vincent Joris

Sous la supervision de Davide Picca

Projet de développement logiciel

Printemps 2024

UNIL

Table des matières

[Introduction 3](#_Toc167831273)

[Contexte 3](#_Toc167831274)

[Motivations et principe général 3](#_Toc167831275)

[Enquête et résultats statistiques 3](#_Toc167831276)

[Description détaillée des fonctionnalités 4](#_Toc167831277)

[Aspects techniques 5](#_Toc167831278)

[Structure de la base de données 5](#_Toc167831279)

[Récolte des données 6](#_Toc167831280)

[Architecture logiciel 6](#_Toc167831281)

[Nestor 7](#_Toc167831282)

[Gestion du projet 7](#_Toc167831283)

[Diagramme de Gantt 7](#_Toc167831284)

[Déroulement du projet 8](#_Toc167831285)

[Documents d’archive 8](#_Toc167831286)

# Introduction

## Contexte

Au cours de leur cursus à l’Université de Lausanne, les étudiants en Lettres jouissent d’une grande liberté dans le choix de leur parcours. Les plans d’étude proposent souvent plusieurs cours à choix pour valider un même module, permettant aux étudiants de suivre des enseignements variés qu’ils peuvent sélectionner selon leurs intérêts. En outre, le plan d’étude de Bachelor prévoit un *programme à options*, représentant 20 crédits ECTS sur les 180 à valider pour obtenir le papier. Ce *programme à options* propose une myriade de cours, dans toutes les disciplines de la Faculté, ainsi que dans certaines Facultés externes, que les étudiants peuvent sélectionner à leur guise.

## Motivations et principe général

Le nombre de cours à choix s’élève à environ 1000. Devant une telle masse de possibilités, il est souvent difficile pour les étudiants de trouver des cours qui les intéressent. Ils s’arrêtent généralement aux disciplines et aux intitulés des cours, parfois énigmatiques ou peu représentatifs de la matière enseignée. L’idée des créateurs de ce logiciel, tous quatre étudiants en Lettres à l’UNIL, est de faciliter ce choix et d’offrir un aperçu des cours qui ont un potentiel intérêt pour l’utilisateur.

Pour faire cela, l’utilisateur spécifie des branches, cours, et enseignants qui lui plaisent et reçoit en retour une sélection de cours susceptibles de l’intéresser.

## Enquête et résultats statistiques

Afin d’avoir une meilleure idée des besoins des étudiants, nous avons procédé à une enquête sous la forme d’un *Google Form*. Les participants à ce sondage devaient répondre à 4 questions en leur attribuant une note de 1 à 5, qualifiant la pertinence des concepts énoncés. Les questions étaient les suivantes :

1. Est-ce que ce concept d’application vous parait être utile pour des étudiants de l’UNIL ?
2. Nous prévoyons de donner la possibilité de trier les cours qui s’affichent par module (les catégories de chaque cours comme présenté dans le plan d’étude). Pensez-vous que ceci est pertinent ?
3. Nous prévoyons de donner la possibilité de trier les cours qui s’affichent par enseignant. Pensez-vous que ceci est pertinent ?
4. Nous prévoyons de donner la possibilité de trier les cours qui s’affichent en fonction de l’horaire. Pensez-vous que ceci est pertinent ?

Finalement, le questionnaire se terminait par un champ libre pour que les sondés puissent laisser leurs remarques et suggestions.

Par l’intermédiaire de ce formulaire, nous souhaitions connaître l’intérêt d’une telle application, ainsi que les fonctionnalités désirées par les utilisateurs.

Le nombre de participants s’élève à 13, ce qui est relativement peu pour prétendre être un échantillon représentatif de l’ensemble du corps estudiantin des Lettres. Toutefois, nous le considérons comme suffisant pour notre projet et pour les délais très courts que nous avons pour le mener à bien.

RÉSULTATS

À posteriori, le point faible de notre questionnaire pour évaluer la représentativité de notre échantillon est l’absence d’une question concernant les branches étudiées par les participants. Cependant un point important vient pallier cette faiblesse : la récupération, certes optionnelle, du nom du sondé ainsi que le fait que les participants sont issus de notre cercle proche nous permet de nous assurer de la diversité des branches représentées dans l’échantillon. Les disciplines représentées sont, à notre connaissance, Informatique pour les sciences humaines, Anglais, Linguistique et Cinéma.

Dans l’ensemble, les notes sont relativement bonnes. La première question présente une moyenne de 4.38, la seconde de 4.23, la troisième de 4.15 et la quatrième de 4.76. Une seule réponse au sondage présente des notes qui détonnent avec le reste de l’échantillon, avec notamment une note de 2 à la première question. La personne n’ayant pas laissé de commentaire dans le champ libre, il n’est pas facile de commenter davantage ce résultat.

Deux participants au sondage ont donné l’idée d’ajouter un filtre de tri par crédits ; idée pertinente qui nous a semblé bon d’ajouter à notre modélisation.

## Description détaillée des fonctionnalités

Le projet verra le jour sous la forme d’une première version, constituée du minimum viable de fonctionnalités. Les versions suivantes offriront d’autres options destinées à améliorer l’expérience de l’utilisateur.

VERSION ALPHA

Pour cette première version, l’application se présente sous la forme d’une simple interface. Avant d’accéder à celle-ci, l’utilisateur spécifie les disciplines qu’il étudie. L’application exclut du *programme à option* les cours des deux disciplines de l’utilisateur. Un calcul de similarité entre la description de chaque cours du *programme à option* et celles des cours des branches exclues est alors performé. Le résultat est ensuite présenté à l’utilisateur, lui montrant les cours du *programme à option* affichant le meilleur score de similarité.

Cette première version calcule les résultats sur la seule base des branches étudiées par l’utilisateur. Elle ne prend en compte ni ses goûts, ni les horaires et les crédits propres à chaque cours. Elle ne répond pas non plus au besoin suivant : dans la plupart des cursus, les cours ne sont pas obligatoires et varient d’une année à l’autre. Il est donc fréquent que les étudiants remplissent leur *programme à option* en prenant des cours dans leurs propres disciplines. Dans cette première version, pour une question de simplicité d’implémentation, tous les cours d’une discipline étudiée sont exclus des propositions faites à l’utilisateur.

FONCTIONNALITÉS ADDITIONNELLES

Afin de répondre aux besoins énoncés ci-dessus, diverses fonctionnalités doivent être ajoutées. La complexité de ces ajouts varie du simple filtre par attribut, pour les crédits, au problème plus délicat des cours d’une discipline déjà étudiée par l’utilisateur. Il est donc nécessaire de réfléchir à la modélisation de ces fonctionnalités en amont de la première version, puisque la structure des données se doit d’être adaptée aux besoins d’implémentation futurs.

Filtre par crédits

Cette fonctionnalité permet à l’utilisateur de n’afficher que les cours qui octroient le nombre de crédits désiré.

Filtre par horaire

Cette fonctionnalité permet à l’utilisateur de spécifier les horaires auxquels il souhaite avoir des propositions de cours et de n’afficher que les cours correspondants.

Filtre par semestre

Cette fonctionnalité permet à l’utilisateur de spécifier le semestre au cours duquel il souhaite avoir des propositions de cours et de n’afficher que les cours correspondants.

Filtre par langue

Cette fonctionnalité permet à l’utilisateur de spécifier une langue et de n’afficher que les cours donnés dans cette dernière.

Filtre par intervenant

Cette fonctionnalité permet d’intervenir sur la sélection des cours comparés avec ceux du *programme à option*. Elle permet de ne retenir dans l’ensemble des cours comparés que ceux donnés par un ou plusieurs enseignants indiqués par l’utilisateur.

Filtre par bloc

Cette fonctionnalité permet d’intervenir sur la sélection des cours comparés avec ceux du *programme à option*. Elle permet de ne retenir dans l’ensemble des cours comparés que ceux propres à un module particulièrement apprécié par l’utilisateur.

# Aspects techniques

## Structure de la base de données

Les fonctionnalités, autant de la version alpha que de la version avancée, demandent une structure des données adaptée. Il faut donc construire la base de données en ayant toujours en vue les besoins imposés par ces fonctionnalités. La structure de la base de données est représentée par le schéma UML suivant :

Une image contenant texte, Police, nombre, capture d’écran

Description générée automatiquement

Fig. 1 Schéma UML de la base de données.

Les entités *horaire* et *intervenants* ont été créés pour faciliter l’implémentation des filtres horaire et professeur. Elles entretiennent une relation *many-to-many* avec l’entité cours, permettant ainsi à un cours d’avoir plusieurs plages horaires et vice-versa ; il en va de même pour les enseignants.

À une étape postérieure du projet, nous avons réalisé qu’inclure les modules, ici *bloc*, dans les données compliquait fortement la récolte des données. Nous avons donc décidé de renoncer à ce filtre.

## Récolte des données

Pour obtenir les données, nous avons d’abord cherché à obtenir une copie de la base de données utilisée par l’Université de Lausanne. Les démarches administratives ont été lentes et n’ont pas abouti. Aussi, nous avons dû trouver une solution alternative pour peupler notre base. Nous avons opté pour un script Python.

Le script est écrit en programmation orientée objet. Il ne contient qu’une seule classe, UnilScrape (fig. 2). Pour ce projet, nous nous sommes concentrés sur la Faculté des Lettres. Toutefois, ce code a été conçu de sorte à ce qu’il soit adaptable à d’autres Facultés. Il faut pour cela modifier directement les paramètres dans le script.

La classe UnilScrape est initialisée avec le chemin de la base de données et la faculté à scraper. La méthode scrape commence par configurer les plages horaires et les branches à exclure. Elle collecte ensuite les données HTML du site web et extrait les informations des cours. Les informations des branches et des cours sont ensuite insérées dans la base de données.

## Architecture logiciel

Nous avons opté pour de la programmation orientée objet pour l’architecture de notre logiciel. Cette architecture est représentée par le schéma suivant :

Une image contenant texte, diagramme, capture d’écran, Parallèle

Description générée automatiquement

Fig. 2 Schéma orienté objet de l’architecture du logiciel.

* La classe *Search* permet la recherche dans la base de données. Elle prend en compte les différents filtres appliqués.
* La classe *course\_group*, *GroupCours* dans le code, récupère les données obtenues par l’objet *Search*.
* La méthode *similarity* calcule la similarité entre les deux groupes de cours en utilisant un modèle Doc2Vec pré-entraîné et la mesure de similarité cosinus. Le premier groupe est constitué des cours suivis par l'utilisateur, c'est-à-dire les cours des deux branches sélectionnées. Le deuxième groupe est composé du reste des cours disponibles dans le programme à option.
  + Pour procéder au calcul de similarité, les noms des cours des deux ensembles de données sont extraits. Ensuite, les vecteurs Doc2Vec sont inférés pour les documents de chaque groupe en utilisant le modèle pré-entraîné. Une fois les vecteurs Doc2Vec obtenus, une matrice de similarité cosinus est calculée entre les vecteurs des deux groupes.
  + L'agrégation des similarités est ensuite effectuée en fonction du type de cluster spécifié (moyenne, maximum, minimum, médiane). Cela permet d'obtenir une mesure de similarité unique pour chaque cours du premier groupe par rapport à l'ensemble des cours du deuxième groupe.
* La méthode *export\_data* permet d’exporter les données dans un fichier Excel ou CSV.
* La méthode *build\_model* construit un nouveau modèle Doc2Vec en utilisant les données de cours. Les objectifs et le contenu des cours sont combinés pour former le corpus. Ce corpus est ensuite utilisé pour entraîner un modèle Doc2Vec, qui est sauvegardé pour une utilisation future. Cette méthode est privée et n’est utilisable exclusivement qu’au sein du backend.

# Nestor

Le nom Nestor fait référence au roi Nestor de Pylos, un personnage de la mythologie grecque. Dans l’Iliade et l’Odyssée, les épopées attribuées à Homère, il est dépeint comme un guerrier âgé mais toujours vigoureux et vaillant. Il est souvent consulté par les autres héros pour ses conseils avisés et sa sagesse.

Pour un logiciel destiné à fournir des conseils d’orientation académique, ce choix de nom est donc tout à fait approprié. En outre, la simplicité et l’originalité de ce nom le rendent facile à retenir.

# Gestion du projet

## Diagramme de Gantt

Pour organiser notre temps et notre travail, nous avons procédé à la création d’un diagramme de Gantt, ordonnant ainsi nos tâches de manière chronologique. Le travail a été réparti selon les aptitudes et préférences de chacun. Nous avons ainsi séparé les activités en différentes catégories : tout d’abord, la récolte des données et création de la base, ainsi que la création de la structure FLASK ; puis dans le *backend*, d’une part les tâches indispensables au *minimum viable project*, et d’autre part les différents filtres ; la création du *frontend* ensuite ; et finalement un onglet *autre*, qui comprend la phase de test et la rédaction du présent rapport.

Une image contenant texte, capture d’écran, ligne, nombre

Description générée automatiquement

Fig. 3 Diagramme de Gant, répartition des tâches dans un espace temporel défini.

## Déroulement du projet

Nous avions prévu une grande marge à la fin du projet pour anticiper d’éventuels problèmes et retards. Cela s’est avéré utile. La réalisation générale a pris du retard. Le peuplement de la base de données à été plus long qu’escompté, ce qui a impliqué un retard généralisé, principalement sur le backend.

La difficulté à choisir un système pour le frontend a également engendré du retard dans la réalisation de l’interface.

Nous n’avons pas prévu de temps pour gérer l’interaction entre le backend et le frontend. Nous avons donc dû ajouter cette tâche à la fin.

L’ambiance générale de l’équipe est bonne. Tous les acteurs du projet participent à la réalisation et aucun disfonctionnement n’est à signaler.

## Documents d’archive

**Une image contenant texte, capture d’écran, conception

Description générée automatiquement**

Première version de l’interface

Une image contenant texte, capture d’écran, nombre, Police

Description générée automatiquement

Deuxième version de l’interface. Accueil

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Deuxième version de l’interface. Interface filtre

Une image contenant texte

Description générée automatiquement

Deuxième version de l’interface. Filtre horaire