Ecole Nationale des Ponts et Chaussées

École des Ponts ParisTech
2023 - 2024

Rapport de TDLOG

Gabriel MARIADASS

Raphaël VIARD

Projet techniques de développement logiciel

Application de revente de places

Encadrant :
E. Concas

Table des matières

Introduction

L'achat de places se fait souvent sans problème sur les sites des organismes concernés : places de concert, de spectacle ou encore de match de foot. Cependant, en cas d'erreur ou d'imprévu, il n'existe pas de plateforme largement connue permettant de revendre des places de façon sécurisée. C'est donc ce que nous avons voulu créer à travers ce projet, pour que chaque place achetée ne soit pas perdue.

1 Objectifs initiaux

Nos **objectifs initiaux** étaient les suivants : créer un site internet permettant la revente et l'achat de places de concert ou encore de spectacles. Nous voulions également qu'il soit possible de personnaliser les recherches, en les triant par exemple par date, par nom d'évènement ou par lieu. Nous voulions également qu'il soit possible de personnaliser légèrement son profil, ainsi que de visiter la page d'autres utilisateurs, notamment pour voir quels tickets ces utilisateurs vendent.

De plus, ce type de site internet nous paraissait cohérent avec les différentes **vidéos du cours** : utilisation de classes Python pour les utilisateurs, les tickets, les transactions, et autres grâce à SQLAlchemy comme nous le verrons. Le fait de coder un site internet nous permettait également de coder des interfaces graphiques, notamment en HTML, CSS et JavaScript. Nous pouvions également utiliser un réseau local pour lancer notre application Flask, en lien avec les vidéos sur la programmation réseau et web. Enfin, pour stocker toutes les informations de notre site, c'est-à-dire toutes les informations concernant les utilisateurs et les tickets, nous avions besoin de créer et de gérer une base de données, ce que nous avons réalisé. Ces divers éléments que nous venons d'évoquer nous ont donc laissé penser, à juste titre, que ces objectifs et ce **choix de projet** s'inscrivaient totalement dans le cours (notamment les vidéos) de TDLog.

Comme rapide aperçu de notre étude initiale, voici **quelques cas non exhaustifs** d'utilisation de notre site web (chacun correspond à un onglet ici) que nous avions prévu :

— Utilisateur 1 : Il visite le site pour mettre en vente un ticket à une date pour laquelle il sera finalement en voyage. Il peut choisir son prix de revente.

— Utilisateur 2 : Il vient sur le site car il n'a pas eu de place pour un concert qu'il souhaitait, et dont les places sont épuisées. Il vient en triant les évènements par nom pour voir si des places sont disponibles.

— Utilisateur 3 : Il vient car il avait mis un ticket en revente, qui n'a pas été acheté, et il souhaite le retirer de la vente car il connait quelqu'un de proche qui en a besoin.

2 Choix techniques

— **Flask** - Pour le développement de notre site internet local, nous avons finalement opté pour le framework web Flask (voir difficultés rencontrées). Nous étions conscients qu'il s'agit d'un « mini » framework, offrant le strict nécessaire pour construire des applications web sans les fonctionnalités plus avancées que d'autres plateformes plus volumineuses peuvent offrir. Cependant, il s'agit d'un outil léger et minimaliste, parfaitement adapté à notre objectif de développement dans le cadre académique. De plus, comme d'autres frameworks, Flask est associé à un serveur de développement intégré lors de l'installation, facilitant ainsi le développement rapide du site web avec la visualisation progressive des modifications apportées, un critère de choix important pour nous.

— **Langages** - Nous avons utilisé plusieurs langages :
  — Python, HTML, JavaScript, CSS : Ces 4 langages sont fondamentaux pour le développement web. Python est utilisé pour le backend, traitant les requêtes et gérant la base de données ; HTML est le langage de création des pages web, caractérisé par des balises entourées de crochets "<>" ; JavaScript est un langage permettant la manipulation dynamique et interactive des éléments de la page HTML, intégré au sein de la balise "<script>" ; enfin, CSS est utilisé pour définir la présentation visuelle de la page web écrite en HTML, au sein de la balise "<style>".
  — jQuery : Via CDN, nous incluons la bibliothèque JavaScript jQuery à plusieurs reprises. Un CDN (Content Delivery Network) est un réseau de serveurs distribué géographiquement, permettant de fournir le contenu de manière plus rapide et efficace. Nous l'utilisons dans les balises « <link> » car au lieu de faire référence à une référence locale du fichier sur le serveur, nous utilisons une URL hébergée sur un CDN. La bibliothèque jQuery simplifie l'interaction avec les éléments HTML, la manipulation des événements et l'exécution d'opérations.
  — Bootstrap : Nous utilisons également Bootstrap, un framework CSS open source qui offre une collection d'outils visuels et de composants prêts à l'emploi dans le site web, évitant ainsi de les redéfinir dans des balises « <style> ». Nous incluons Bootstrap via CDN, comme pour jQuery. Bootstrap est utilisé notamment pour notre barre de navigation (« navbar ») et des boutons Bootstrap (« btn », « btn-primary », « btn-danger », etc.).

— **Jinja2** - Le moteur de template Python Jinja2 n'est pas exclusif à Flask, bien que Flask soit l'un des frameworks web les plus populaires utilisant Jinja2 comme moteur de template par défaut. Un moteur de template est un outil qui permet de générer des pages web en combinant des modèles statiques avec des données dynamiques. Un moteur de templates fonctionne de la manière suivante :
  — Modèle (à ne pas confondre avec models.py qui contient les classes des objets) - Ils contiennent le code statique et les balises pour indiquer où insérer les données dynamiques.
  — Données dynamiques - Les données dynamiques sont celles qui peuvent changer à chaque requête et à chaque utilisation. Ces données peuvent être extraites de la base de données (voir SQLALCHEMY) ou d'entrées d'utilisateur.
  — Moteur de template - Le moteur de template prend le modèle statique et les données dynamiques où il remplace les balises par les valeurs réelles des données dynamiques afin de générer une version finale du document.
  — Rendu - La page web finale, comme page HTML complète, est ensuite rendue de manière dynamique au navigateur.

  Cela nous a permis d'intégrer facilement des éléments dynamiques tels que des fenêtres modales ou des boutons de manière propre et efficace. Par exemple, les variables associées aux balises peuvent être incluses dans le modèle statique comme user.name qui pourrait être utilisé pour afficher le nom d'un utilisateur.

— **SQLALCHEMY et SQLite** - Tout comme Jinja2, SQLAlchemy n'est pas exclusif à Flask, bien que souvent associé à Flask en raison de son utilisation fréquente dans les applications web développées avec ce framework. Il s'agit d'un ORM (Object-Relational-Mapping) qui simplifie le travail de gestion des bases de données car elle fournit des outils en Python afin d'exécuter les requêtes. Dans notre cas, SQLAlchemy offrait un support pour la base de données SQLite, qui servait donc de table afin de stocker ce dont nous avons besoin, c'est-à-dire les valeurs des attributs des différents éléments associés aux classes dans models.py.

— **Flask migrate** - Flask migrate est une extension de Flask pour faciliter la gestion des migrations de base de données dans le cadre de l'utilisation de SQLAlchemy qui interagit, dans notre cas, avec la base de données relationnelle SQLite comme nous l'avons vu. Les migrations sont des changements de structure de la base de données.

— **Flask login** - Comme Flask migrate, Flask login est une extension de Flask qui simplifie la gestion de l'authentification des utilisateurs sur notre site web, ce qui était nécessaire dans le cadre de notre objectif.

3 Réalisation

Pour concrétiser cette application Flask, nous avons créé tous les répertoires et fichiers nécessaires au bon fonctionnement de celle-ci. La structure de notre **projet Flask** est la suivante (nous ne mentionnons pas les répertoires standards tels que venv) :

— run.py : Ce fichier permet de démarrer l'application Flask ; il contient la commande pour lancer le serveur de développement Flask.

— config.py : Ce fichier stocke les configurations globales du projet, indépendamment de l'application Flask, telles que les informations nécessaires à ce niveau structurel pour l'utilisation de la base de données.

— instance et migrations : Le premier fichier contient le fichier .db de notre base de données, tandis que le second, créé automatiquement lors de l'opération de migration, permet de gérer les modifications structurelles de la base de données.

— app : ce répertoire est celui associé à notre application, sa structure est détaillée ci-dessous.

La structure de notre **app** est la suivante :

— init : Ce fichier permet d'initialiser une instance de l'application. Il permet de configurer des paramètres spécifiques à l'application et d'enregistrer des extensions (comme l'initialisation de Flask-Migrate, Flask-Login, ou encore de la base de données).

— config.py : Ce fichier est utilisé pour stocker les configurations spécifiques à l'application.

— models.py : Dans notre contexte d'une application Flask utilisant l'ORM SQLAlchemy, models.py est le fichier où nous définissons nos classes de modèle qui correspondent aux tables de la base de données. Les deux classes utilisées ici sont seulement les classes Ticket et User. Pour savoir à qui appartient un ticket, il suffit de regarder l'attribut Username. De plus, pour savoir si le ticket est en vente, il faut regarder l'attribut To sale qui est un booléen. Si il est sur true, cela signifie qu'il est à vendre. En effet, un ticket présent sur le site n'est pas forcément à vendre, par exemple lorsqu'un utilisateur achète un ticket, il le stocke sur son compte et il n'est plus à vendre.

— routes.py : Ce fichier permet de définir les routes de l'application, déclarées à l'aide du décorateur @app.route(). Dans le contexte de Flask, une route correspond à une URL spécifique à laquelle l'application répond avec une fonction particulière. La vue associée à une requête est alors cette fonction qui est associée à une route particulière.

— static : Ce répertoire permet de stocker les fichiers statiques tels que les fichiers CSS, JavaScript ou encore les images.

— uploads : Ce répertoire, qui aurait pu être intégré dans static, mais laissé en dehors pour des raisons pratiques, permet de stocker les fichiers PDF associés aux tickets mis en vente par les utilisateurs.

— templates : Ce répertoire contient les différents fichiers HTML que l'on énumère ci-dessous.

Voyons maintenant les différentes pages HTML contenues dans le dossier template.

— index.html : Cette page est la page d'accueil du site. Nous y retrouvons un petit message d'accueil, ainsi que la barre de navigation (présente sur toutes les pages), contenant des liens vers les différents onglets, ainsi que des boutons pour s'inscrire ou se connecter à notre site.

— onglet1.html : Cette page est une page affichant tous les tickets que l'on peut acheter. Avant de la charger, nous cherchons donc dans la base de données ceux qui sont en vente et qui n'appartiennent pas à l'utilisateur connecté. Il est également possible d'effectuer une recherche ciblée par date, nom d'évènement ou lieu.

— onglet2.html : Cette page sert à mettre en vente un ticket. Nous pouvons donc y inscrire le nom de l'évènement, sa date, son lieu, son prix de vente ainsi qu'un fichier PDF correspondant au ticket.

— onglet3 : Cette page a plusieurs tâches : il est possible d'y modifier sa biographie, son mot de passe, d'augmenter son solde, ainsi que de supprimer un ticket du site, de ne plus en vendre certains ou de changer leurs prix.

— PageUser.html : Cette page nous donne les informations d'un utilisateur autre que nous : la biographie de son profil ainsi que les tickets qu'il met en vente sur notre site. À noter que cette page n'est atteignable que dans onglet1.html, lorsque l'on regarde les différents tickets que l'on peut acheter, il suffit de cliquer sur le nom de l'utilisateur associé.

— connexion.html et inscription.html : Ces pages, comme leur nom l'indique, permettent de se connecter ou de s'inscrire à notre site. Pour l'inscription, cela crée un nouveau User dans notre base de données.

— Solde-insuffisant.html : Cette courte page s'affiche lorsqu'un utilisateur tente d'acheter un ticket mais que son solde n'est pas suffisant.

— deconnexion.html : Cette page demande à l'utilisateur actif de confirmer sa déconnexion.

L'ensemble de ces pages nous permet donc de faire fonctionner notre site. Une autre partie importante de l'application Flask est les routes, que nous allons présenter. Les principales seront donc les suivantes :

— /onglet1 : On récupère les données du formulaire pour le tri par date/lieu/nom d'évènement ou code unique, et on va chercher dans la base de données les tickets ayant toutes les caractéristiques que l'utilisateur a demandées dans sa recherche.

— /onglet2 : lorsqu'on rajoute un ticket, on récupère toutes les données du formulaire présent dans la page HTML, puis on crée le ticket qu'on rajoute dans la base de données.

— /onglet3 : On récupère les tickets de l'utilisateur actif depuis la base de données : ceux qui sont en vente et ceux qui ne le sont pas, pour ensuite les afficher dans la page HTML.

— /acheter-ticket : On récupère le ticket souhaité du formulaire, on vérifie que il n'est pas à l'utilisateur actif, et si c'est le cas, on modifie les soldes des deux personnes concernées, et on attribue le ticket à l'acheteur. Pas d'ajout ni de retrait dans la base de données.

— /inscription : On récupère les données du formulaire de la page HTML pour ensuite créer un nouvel User et le mettre dans la base de données.

— /connexion : On récupère l'username et le mot de passe du formulaire, on vérifie qu'ils correspondent bien à un utilisateur, et si c'est le cas, on connecte l'utilisateur en question.

— /deconnexion : déconnecte l'utilisateur.

— /mettre-en-vente : On récupère les éléments du formulaire qui constitueront le ticket (Lieu, Nom, date, prix, PDF), et on génère un code unique pour le ticket, on enregistre le PDF dans le dossier uploads, puis on crée le ticket qu'on met dans notre base de données.

— /download-pdf set à télécharger les fichiers PDF lorsque l'on possède un ticket.

— /ticket-en-vente : Sert à récupérer au format JSON les tickets à vendre de l'utilisateur actif.

— /mettre-a-jour-solde : Récupère l'ajout de solde du formulaire de la page HTML, puis augmente le solde de l'utilisateur actif de cette quantité. Ici, le solde est fictif, et n'importe qui peut augmenter son solde d'autant qu'il le souhaite.

— /PageUser : cherche dans la base de données l'utilisateur avec le nom donné, puis récupère tous les tickets en vente de cet utilisateur et les envoie en argument dans la page HTML PageUser afin de pouvoir les afficher.

— /supprimer-ticket : retire le ticket sélectionné de la base de données.

— /modifier-vente : permet de modifier les informations d'un ticket de l'utilisateur actif. Mets donc à jour la base de données.

Ainsi, ces différentes routes et pages HTML nous ont permis de réaliser notre application Flask, le tout en gérant notre base de données Flask-SQLAlchemy.

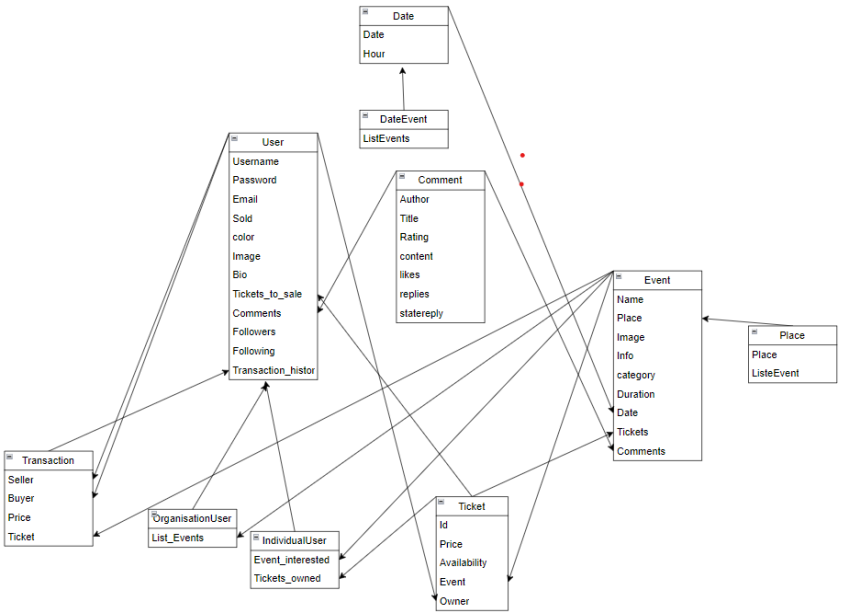
4 Difficultés rencontrées

Au commencement de ce projet, nous n'avions en tête que des objectifs initiaux qui n'étaient alors pas très précis. Nous étions très libres soudainement, sans réel énoncé, et nous ne savions pas quels outils choisir ni par quelle étape commencer.

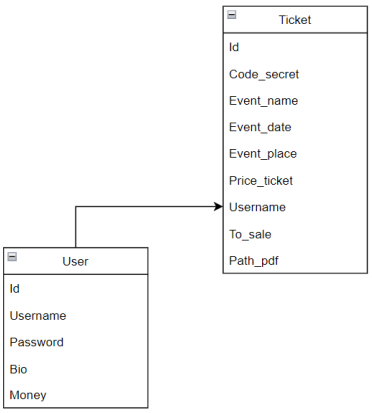
Premièrement, comme nous voulions développer un site internet dès le départ, nous avons choisi de nous intéresser premièrement au framework web Django. Nous avons consacré du temps au début afin d'essayer de comprendre la structure et l'utilité des répertoires et des fichiers d'un projet Django. Django est plus complexe que Flask, et la quantité de fonctionnalités intégrées peut être excessive pour le cadre d'un tel projet académique. De plus, l'apprentissage et la prise en main du développement d'un projet Django sont bien plus difficiles pour un tel framework. Ainsi, voyant au bout d'un moment que cela ne menait à rien et que nous n'avions pas réellement de résultats ni de support de développement que nous maîtrisions, nous sommes passés au framework web plus simple, Flask. Cependant, cette étape n'a pas été inutile car les notions que nous avions découvertes sont largement reprises par Flask, mais de manière plus simple. Par exemple, les structures des projets sont similaires : nous avons des répertoires pour les applications, pour la configuration, pour les fichiers statiques, et des fichiers ou encore pour la base de données. La structure des applications est également similaire car elles contiennent par exemple les migrations, les modèles, ou encore une configuration. À la différence de Flask, Django sépare les vues et les routes dans deux fichiers différents, ce qui est un exemple de complexité supplémentaire de ce framework.

Nous avons ensuite entendu parler de Cookie cutter, qui permettait de générer des projets open source utilisant des modèles prédéfinis pour créer des structures de projet déjà établies. Il pouvait aussi bien être utilisé avec Django, que nous avions délaissé, et Flask, que nous avions choisi. Cependant, finalement, nous n'avons pas utilisé de générateur de structure tel que Cookie cutter, mais nous sommes partis de zéro. Cela nous a permis de comprendre chacun des éléments de notre code et de ne pas avoir de difficulté à identifier l'utilité de chaque élément. Nous pensions qu'il s'agissait d'une meilleure approche dans ce cadre académique, mais plus généralement également car il est utile de comprendre l'ensemble de la structure pour développer le projet plus rapidement. Il s'agissait peut-être d'une faible perte de temps au départ, mais cela nous a grandement fait gagner du temps lors du développement.

Nous avons ensuite commencé à créer un système de classes Python pour représenter toutes les choses dont nous avions besoin. Cela comprenait des commentaires (que l'on peut faire sur un utilisateur ou un événement), des dates, des événements, des tickets, des transactions et des utilisateurs. Les utilisateurs pouvaient être des particuliers ou des professionnels [1]. Nous avons ensuite fait quelques simulations avec ces classes Python, de création d'utilisateur, de création de tickets, et de vente/achat de tickets entre utilisateurs. Voici un diagramme UML des classes et de leurs différents liens :



Nous pouvons voir que ce modèle de classe est très complexe et entremêlé. Nous nous sommes rendus compte que ce schéma était trop complet, ce qui compliquait le développement, et pouvait comporter des boucles de classes non désirables. Pour réaliser notre site internet, nous avons donc ensuite décidé de simplifier ce modèle et donc de ne pas utiliser toutes les classes Python mentionnées plus haut, mais seulement deux classes, Ticket et User, représentées dans le diagramme UML suivant :



Path-pdf est le chemin vers le fichier PDF. Le code secret est un code unique généré pour chaque ticket. Les clés primaires pour la base de données sont id, mais nous nous servons de username pour user et de id ticket comme pseudo clé primaire dans notre code pour des raisons pratiques de meilleure compréhension et de meilleure lisibilité. Username est également une clé étrangère pour la classe Ticket.

À la suite de l'ensemble de ces étapes et de ces difficultés que nous avons su repérer et surmonter, nous avons enfin commencé le réel développement de notre site web avec les choix techniques la réalisation présentés plus haut. Il y a une réelle histoire et démarche derrière ces choix et ce développement.

1. par exemple, une salle de concert

5 Pistes d'amélioration

En raison du manque de temps et du fait que nous ne sommes que deux, nous avons envisagé des fonctionnalités supplémentaires qui ne sont pas actuellement présentes sur le site web.

Tout d'abord, du point de vue **technique**, nous avons pensé à remplacer certaines requêtes, par exemple lors du tri pour rechercher des tickets, par des requêtes **AJAX**, comme mentionné dans les vidéos portant sur la programmation réseau et web. Si le site compte un grand nombre de tickets, cela permettrait d'éviter de tous les recharger et de n'afficher que le nécessaire. Une autre amélioration à laquelle nous avons pensé est de classer les événements par genre (par exemple, concert de Rock, match de foot, etc.), pour permettre aux utilisateurs de voir quels types d'événements les intéressent, et de créer une liste d'événements recommandés pour chaque utilisateur en fonction de certains critères, tels que le lieu ou le type d'événement. On peut même envisager l'utilisation d'algorithmes **d'intelligence artificielle** pour accomplir cette tâche. Un autre point est que nous pourrions ajouter des éléments esthétiques sur les différentes pages. En effet, le site est pour l'instant très simple et va à l'essentiel, mais nous pourrions imaginer diverses animations ou des interfaces graphiques pour rendre l'ensemble plus attrayant.

Ensuite, du point de vue **fonctionnel**, une autre fonctionnalité à ajouter serait de créer deux sous-classes de la classe User, une pour les particuliers et une pour les organisateurs d'événements. Les utilisateurs particuliers seraient en fait les utilisateurs classiques utilisés jusqu'ici, et les organisateurs d'événements pourraient mettre un grand nombre de places d'un coup sur notre site, sans possibilité d'achat de ticket. Ils pourraient également avoir des fonctionnalités plus poussées, comme des pages spéciales détaillant tous leurs événements. L'exemple typique d'organisateur d'événement pourrait être une salle de concert. Nous pourrions également envisager d'ajouter un système de visualisation du trafic et des statistiques par les utilisateurs afin de savoir si un grand nombre d'utilisateurs est intéressé par leurs tickets ou non. De plus, nous pourrions établir un système d'enchères possible pour la revente d'un ticket, même si cela n'est pas forcément très éthique et toujours légal ou autorisé par les organisateurs d'événements. Par manque de temps, nous n'avons pas pu ajouter un système d'intégration et de tests de notre site web, nous aurions pu notamment utilisé selenium ou encore Flask-testing qui permettent d'écrire des tests en python dans un fichier séparé, et ainsi permettre de s'assurer la bonne santé du site Internet. Cependant notre application reste assez limitée et nous avons pu largement tester nous même l'ensemble des fonctionnalités afin de s'assurer qu'il n'y avait aucun problème.

Conclusion

En fin de compte, ce projet nous a permis de participer au développement d'un logiciel, conformément à nos attentes. En plus de renforcer nos connaissances culturelles et techniques acquises grâce aux vidéos, ce projet nous a réellement confrontés aux problématiques d'un développeur. Nous avons mené à bien ce projet, qui était à la fois réalisable et non trivial, nous procurant une certaine satisfaction malgré des objectifs initiaux qui n'étaient pas forcément très précis. Ainsi, nous avons appris de nouvelles notions, renforcé nos acquis par l'expérience, fait face à de réelles problématiques de développement, appris à être réalistes et pragmatiques dans le choix des objectifs et dans leur réalisation, tout en travaillant sur notre capacité à travailler en équipe. Le cheminement que nous avons suivi nous a poussés à choisir les outils techniques les plus adaptés dans le cadre de ce projet pour surmonter les difficultés rencontrées, tout en envisageant des améliorations possibles compte tenu de la limite de temps imposée.

Concernant notre retour d'expérience :

— **Gabriel** - J'ai personnellement trouvé cette matière très intéressante.

Tout d'abord, les travaux pratiques que nous devions rendre en début de semestre à la fin des séances nous ont permis de mettre rapidement et efficacement en place les acquis que nous apprenions grâce aux vidéos, facilitant ainsi l'assimilation. De plus, les vidéos étaient très intéressantes et m'ont permis d'apprendre un grand nombre de nouvelles notions, renforçant ainsi ma culture générale dans ce domaine qui m'intéresse particulièrement pour mon avenir professionnel. Ce projet, en plus de la culture générale acquise, m'a permis d'approfondir réellement les thématiques et d'acquérir de nouvelles compétences techniques qui me seront sûrement utiles. Je trouve que la liberté accordée pour ces projets est une bonne chose car elle nous permet d'avoir une démarche complète, nous permettant de nous corriger nous-mêmes lorsque nous ne partons pas dans la bonne direction. Par exemple, au début, nous voulions trop comprendre Django sans réellement le mettre en application ; finalement, nous ne l'avons pas utilisé bien que cette étape aurait été utile dans le cheminement. De même, nous avions développé des classes Python très complexes, tandis que nous avons fait des modèles assez simples en fin de compte. L'ensemble de ces problèmes sur lesquels nous avons su rebondir, s'apparentant à une démarche de recherche que nous avions déjà eue en prépa grâce aux TIPE, ou dans une moindre mesure en première avec les TPE, nous a permis d'acquérir de l'expérience et nous aidera sûrement à cerner d'une meilleure manière les démarches à avoir lors de nos futurs projets. La réalisation d'un tel projet et le fait d'en voir le résultat sont également très satisfaisants, mais je pense que cela est l'une des caractéristiques de l'informatique et de pourquoi nous pouvons tant l'apprécier.

— **Raphaël** - J'ai trouvé ce projet, et de façon plus générale le cours de TDLog, plutôt intéressant. Tout d'abord, cela nous a permis de découvrir plusieurs choses pratiques et utiles pour la suite de notre vie professionnelle, comme les applications Flask, Django, ou encore les bases de données. J'ai aussi trouvé les vidéos instructives et plaisantes, et les tests étaient de difficulté normale pour quelqu'un ayant vu les vidéos. Les TP du début d'année nous permettaient de mieux comprendre en pratique les vidéos. L'un de mes plus grands apprentissages de ce cours est d'avoir découvert la programmation orientée objet en Python. Le projet nous a aussi permis de voir que, en général, il y a toujours des imprévus auxquels nous devons faire face, et nous devons donc trouver des compromis. Un exemple de cela s'est produit lorsque nous avons compris que maîtriser une application Django serait trop chronophage pour le temps que nous avions pour ce projet. Les vidéos m'ont permis au cours du semestre d'acquérir une réelle culture informatique que je n'ai jamais eue. J'ai aussi constaté lors des séances projet que le professeur était à l'écoute lorsque nous lui posions des questions, savait nous guider tout en nous laissant une certaine liberté, ce qui était nécessaire dans un projet de cette envergure. La cohésion de notre groupe était bonne, et nous nous entendions bien ; nous étions la majorité du temps sur la même longueur d'onde, et je me suis rendu compte qu'il est bien plus efficace de travailler à plusieurs, notamment pour disposer des bonnes idées d'un plus grand nombre de personnes. Enfin, je pense garder un bon souvenir de ce projet, qui m'aura appris un large éventail de compétences techniques, tout en me permettant d'améliorer ma capacité à travailler en groupe sur un projet.