Sith: Développement de nouvelles applications

Skia (Florent Jacquet)
Lo-J (Guillaume Renaud)

Dernière version: 15 janvier 2017

Table des matières

Remerciements

Lo-J (Guillaume Renaud)

Je remercie tout d'abord Monsieur Frédéric Lassabe qui nous a permis d'effectuer cette TO52 lors de notre cursus à l'UTBM, nous permettant ainsi de mêler nôtre travail scolaire à nôtre envie de participer à l'amélioration de la vie associative de l'UTBM.

Je tiens aussi à remercier Florent Jacquet qui m'a aidé tout au long de ce travail et à qui j'ai pu poser mes différentes questions pour apprendre et comprendre plus rapidement que si j'avais été seul.

Skia (Florent Jacquet)

Je remercie également Frédéric Lassabe , non seulement pour la TO, mais également pour la précédente TW. Sans ces deux UV hors emploi du temps, jamais un projet comme ce site de l'AE n'aurait pu voir le jour. Cela a demandé beaucoup d'investissement, et il est plus qu'appréciable de pouvoir obtenir quelques crédits en retour.

Je tiens également à remercier l'ensemble de l'équipe info de l'AE qui s'est motivée ce semestre à organiser des réunions hebdomadaires afin de reprendre le projet du mieux possible. C'est maintenant à eux que va être confié le projet, et il est agréable de constater qu'ils n'ont pas attendu le dernier moment pour se pencher sur la question.

Introduction

Après le développement de la base du nouveau site de l'AE, le projet *Sith*, au Printemps 2016, la mise en production a pu avoir lieu avec succès fin Août 2016.

Mais le site était encore très incomplet, et il était nécessaire d'y ajouter un grand nombre de fonctionnalités moins critiques, celles-ci n'ayant pas de rapport avec l'argent, mais tout de même très utiles pour le fonctionnement de l'AE.

Parmis elles, se trouvait notamment une application de gestion des stocks, qui a été confiée à Lo-J, puisqu'elle concernait en premier lieu le *Bureau des Festivités* et qu'il en était le président. Il était donc parmis les mieux placé pour évaluer le besoin et développer l'outil, d'autant qu'il fallait le concevoir depuis le début, ces fonctions n'étant pas du tout présentes dans l'ancien site.

Du reste, Skia a eu la responsabilité de développer les autres applications, ou bien de gérer leur développement lors qu'il était fait par quelqu'un d'autre.

1 Eboutic

Développeur principal : Skia

1.1 But

Fournir une boutique en ligne, avec paiement sécurisé, compatible avec l'API de paiement du Crédit Agricole.

— Gérer les cotisations
— Gérer les rechargements de compte AE
— Gérer différents groupes de vente

1.2 Principaux problèmes

1.2.1 Interaction avec l'API

C'est la principale contrainte de cette application. On doit interagir avec les serveurs du Crédit Agricole, et pour cela, ces derniers n'aident pas beaucoup.

Ils fournissent un PDF peu clair [1] expliquant l'implémentation d'un site marchand, en plus des nombreux autres PDF de documentation disponibles à l'adresse `https://e-transactions.avem-groupe.com/pages/global.php?page=telechargement`.

Une implémentation de référence uniquement en PHP, et contenant que peut de fonctionnalités par rapport à ce que dit le PDF peut aussi être obtenue, mais n'est guère utile excepté pour la vérification cryptographique de la signature de la réponse. Mais encore, il faut arriver à traduire les fonctions propres à PHP, et ce n'est pas toujours une mince affaire, mais fort heureusement, les algorithmes sont encore assez standards et l'on trouve vite de l'aide quant à ces fonctions.

De plus, certaines informations concernants les numéros d'identification de marchand son incohérents d'une documentation à l'autre, et le plus simple à ce niveau est encore de contacter le support.

1. disponible dans le dossier `doc/Etransaction/` des sources du site

1.2.2 Accès concurrentiels

En production, le projet Sith tourne à l'aide d'**uWSGI**, qui s'occupe lui de gérer les différents processus du logiciel. Cela se traduit par des accès concurrentiels à la base de donnée lors de l'appel de deux pages simultanément qui ont besoin d'accèder aux mêmes ressources.

Le problème n'en est la plupart du temps pas un, mais il devient très critique lorsque la page appelée permet par exemple de recharger un compte AE. Il ne faut alors surtout faire l'opération en double.

Pour protéger ces accès en double, on peut alors utiliser des transactions, et **Django** fournit une abstraction très pratique : `with transaction.atomic():`.

L'Eboutic, avec sa réponse de la banque, est très sujette à ces accès concurrents, et cela a posé quelques problèmes dans les débuts. La plupart ont été résolu, mais il arrive encore dans les comptoirs d'avoir une vente en double, sans pour autant avoir le débit du compte qui soit doublé. Cela ne pose pas foncièrement de problèmes, puisque le solde du compte est tout de même valide, et c'est un problème très compliqué à debugger, puisqu'il survient très rarement, mais il faudrait tout de même arriver à le résoudre un jour.

2 Le SAS

Développeur principal : Skia

2.1 But

Fournir un système de galerie de photo :
— Upload en ligne via un formulaire pour tous les cotisants.
— Modération pour l'équipe du SAS.
— Système d'identification des membres pour retrouver rapidement ses photos.
— Affichage des photos dans les différents album et sur la page "photo" du profil d'un utilisateur.

2.2 Principaux problèmes

2.2.1 Gestion des fichiers

L'envoie en grande quantité de photos nécéssite une gestion des fichiers solide, en même temps qu'un formulaire d'envoie efficace, capable d'envoyer plusieurs dizaines de photos en une seule action de l'utilisateur.

L'envoie est donc fait à l'aide de requêtes AJAX pour envoyer les photos une par une et éviter alors le timeout.

Concernant les fichiers une fois envoyé, ils sont en réalité traités par la classe `SithFile` qui gère tous les fichiers du site. Cela ne fait qu'une seule classe à développer, de même qu'un seul système de fichier avec une seule arborescence, ce qui est beaucoup plus robuste.

La difficulté a aussi été de permettre le déplacement des fichiers, par couper-coller, tout en faisant de même dans le système de fichier réel, afin d'avoir une arborescence cohérente même en cas de perte de la base de données.

2.2.2 Optimisation des pages

La génération d'un grand nombre de requêtes SQL est un des principaux problèmes de ralentissement d'un site. Le SAS, avec ses très nombreuses photos,

qui requierent une validation des droits, a posé un gros problème à ce niveau.

Certaines pages ont pu mettre jusqu'à plus de 10 secondes à générer, ce qui est inconcevable pour une galerie de photos, mais ce temps à pu être réduit à moins de 3 secondes.

L'astuce à été d'utiliser des actions utilisateurs, comme l'upload de nouvelles photos, pour faire plus de traitement que nécessaire, afin de mettre en "cache" une grande partie des actions, comme par exemple la génération des miniatures des albums.

Une autre technique pour gagner du temps est de mettre en cache certaines requêtes en forcant les *QuerySet* à s'évaluer dans une *list* **Python** que l'on stocke afin d'obtenir sa longueur, au lieu de lancer d'abord un *count*, puis une itération des résultats, qui en utilisant directement l'ORM, conduit à réaliser deux requêtes SQL.

Enfin, le passage à **HTTP/2** permettrait d'améliorer encore les performances côté utilisateur puisqu'il n'y aurait plus qu'un seul *socket* d'ouvert pour transférer toutes les photos d'une page par exemple, sans avoir pour autant à toucher au code.

3 Les élections

Développeur principal : Sli

3.1 But

Fournir un système d'élections :
— Gestion des différentes élections comprenants à chaque fois une liste de postes pour lesquels les gens candidatent, ainsi qu'une gestion des listes, pour pouvoir classifier et répartir les candidatures.
— Gestion d'une page de vote, permettant aux gens autorisés de pouvoir voter.
— Affichage des résultats une fois le vote terminé.
— Pas compatible avec la législation française : trop contraignant et pas utile, puisque validation officiel en AG.

3.2 Principaux problèmes

3.2.1 Automatisation d'un widget particulier pour les formulaires

La demande est venue du **BdF** qui a voulu autoriser pour certains poste un nombre de vote supérieur à 1. Cela signifie que l'on passe d'un choix simple, type `radio` à un choix multiple, type `checkbox`, tout cela étant paramètrable dans l'élection.

Ce genre de choix n'étant pas disponible dans **Django** de base, il a fallut développer le *widget* à utiliser dans le formulaire qui permette cette configuration tout en validant bien les données reçues par rapport au modèle, et éviter ainsi de pouvoir "tricher" en envoyant des requêtes erronées.

3.2.2 Revue du code d'un autre développeur

Sli étant le principal développeur de cette application, un gros travail de revue de code a dû être effectuer afin de garantir une certaine cohérence avec le reste du projet.

C'est un travail très long et fastidieux, car il faut bien revérifier chaque ligne, sur chaque fichier, tout en faisant des commentaire lorsque quelque chose ne va pas. **Gitlab** a, à ce niveau, grandement facilité la tâche, à l'aide de ses outils de *merge request* assez avancés.

En plus du code en lui-même, il a fallut porter une attention particulière aux migrations. Ces fichiers générés automatiquement par **Django** sont responsables du maintient d'une base de donnée cohérente malgré les évolutions des modèles. Même si le code parait donc valide, il est impératif de surveiller que la chaîne de dépendance des dites migrations ne soit pas cassée, au risque de problèmes potentiels au moment de la mise en production, ce qui entraînent à coup sûr un *downtime*.

4 Les stocks

Développeur principal : Lo-J

Cette application s'occupe de la gestion des stocks des comptoirs de type « BAR ». Elle permet de suivre les quantités restantes afin de pouvoir déterminer de manière automatisée quels sont les produits qu'il faut acheter et en quelle quantité.

4.1 Liste des modèles

4.1.1 Stock

Un Stock possède un nom et est lié à la classe Counter. Ainsi, chaque Comptoir peut avoir son propre Stock.

4.1.2 StockItem

Un StockItem possède un nom, une quantité unitaire, une quantité effective, une quantité minimale et est lié à la classe Stock ainsi qu'à la classe ProductType. De cette manière, chaque élément appartient à un Stock et il est catégorisé de la même manière que les Products (qui sont les objets utilisés pour la vente dans les comptoirs).

4.1.3 ShoppingList

Une ShoppingList possède un nom, une date, un booléen (fait ou à faire), un commentaire et est liée à un Stock. Chaque ShoppingList est donc liée à un Stock ce qui permet d'avoir des listes de courses spécifique à chaque comptoir.

4.1.4 ShoppingListItem

Un ShoppingListItem possède un nom, une quantité demandée, une quantité achetée et est lié à la classe StockItem, à la classe ShoppingList et à la classe

ProductType. Cela permet de pouvoir faire plusieurs listes de courses différentes en même temps et d'en garder un historique.

4.2 Fonctionnement

Au départ, si le comptoir de type « BAR » n'a pas de stock, la seule chose qu'il est possible de faire est d'en créer un. Ensuite, il va falloir créer les objets StockItem en indiquant pour chacun les quantités qu'il y a dans le stock.

De plus, la personne (généralement le Responsable du lieu de vie) qui aura la responsabilité d'informatiser les stocks devra aussi définir la quantité unitaire, effective et minimale.

Par exemple, les Cheeseburger vendus aux différents comptoirs sont achetés par boite de 6, la quantité unitaire sera donc 6, la quantité effective correspondra au nombre de boites restantes dans le stock (c'est à dire dans la réserve du lieu de vie, une boite sortie du stock est considérée comme consommée) et enfin, la quantité minimale servira de valeur seuil.

Une fois l'état de la réserve retranscrit dans le site, il reste encore gérer les stocks de manière quotidienne. Pour ce faire, l'application se décompose en 3 parties :

— Création automatique des listes de courses
— Approvisionnement du stock
— Prise d'éléments dans le stock

Lorsque l'on accède à la partie qui s'occupe de la gestion des listes de courses, il y a un bouton permettant de créer une liste de courses en fonction de l'état des stocks à cet instant, puis un premier tableau contenant les listes de courses qu'il faut faire et enfin un second tableau servant d'historique des listes de courses déjà effectuées.

Pour chaque liste de course ainsi créée, qu'elle soit « faite » ou « à faire », il est possible de cliquer sur son nom pour voir le détail de ce qu'elle comprend.

4.2.1 Création automatique des listes de courses

En cliquant sur le bouton permettant de créer une nouvelle liste de courses, il faut remplir un formulaire. Les informations à donner dans ce formulaire sont le nom de la liste de course (par exemple, une liste spéciale pour Leclerc), ensuite, apparaissent tous les StockItem ayant une quantité effective inférieure au seuil fixé par leur quantité minimale. Il faut donc donner pour chacun de ces éléments une quantité à acheter. Enfin, un dernier champ de commentaire peut être compléter, il sert à demander l'achat d'éléments qui n'apparaissent pas dans le Stock, par exemple, des couteaux, fourchettes ou encore tasses...

Lors de la validation de ce formulaire, la liste de courses est créée et est ajoutée au tableau contenant les listes de courses à faire.

4.2.2 Approvisionnement du stock

Au retour des courses, il faut ranger les produits achetés dans la réserve. À ce moment-là, il faut aussi mettre à jour le stock. Une opération « Mettre à jour le stock » est disponible pour chaque liste de courses du tableau « À faire ».

En effectuant cette action, il va falloir indiquer les quantités effectivement achetées. En effet, les quantités demandées ne sont pas forcément celles achetées, c'est donc les quantités effectives qu'il faut ajouter au stock. Une fois ce formulaire validé, la liste de couses passera de l'état « à faire » à l'état « faite ».

4.2.3 Prise d'éléments dans le stock

La réserve étant accessible aux barmen afin qu'ils puissent réapprovisionner les réfrigérateurs à tout moment, l'interface permettant de prendre des éléments dans le stock a été ajoutée dans les onglets de l'interface des ventes (là où le barman inscrit le code du compte du client qui souhaite commander quelque chose). Ainsi, en revenant de la réserve, le barman doit indiquer le nombre de chaque produit qu'il a rapporté.

Dans un souci de simplicité pour le gérant du lieu de vie, ce formulaire de prise des éléments dans le stock est aussi accessible depuis son interface de gestion.

4.3 Améliorations à apporter

— Il n'est pas encore possible de modifier les quantités demandées pour un ou plusieurs des produits d'une liste de course. Il faudrait rendre cela possible car actuellement, il faut supprimer la liste de courses et la refaire en changeant les quantités souhaitées.

— Il faudrait améliorer la manière dont on ajoute les éléments non définis en tant que StockItem dans la liste de course. Un objet ShoppingListItem avec une référence « Null » vers la classe StockItem pourrait être créé à la place de compléter le champ de commentaires.

— Dans les améliorations sur le long terme, il faudrait que la décrémentation des quantités de chaque élément dans le stock soit automatique. En repensant une partie de l'architecture de l'application, on pourrait faire en sorte que chaque vente faite au comptoir face diminuer les quantités restantes (cela remplacerait le formulaire de « Prise d'éléments dans le stock », mais cela ne serait pas applicable à tous les produits mis en vente. Par exemple, pour les cacahuètes que nous vendons au bol et non par paquet)

— Avec le système de notifications qui a été mis en place sur le site, on pourrait faire en sorte que le ou les responsables des lieux de vie reçoivent une notification lorsque la liste de courses contient plus de 5 éléments

5 La laverie

Développeur principal : Skia

5.1 But

Cette application doit fournir un système de gestion de laverie. Cela comprend :

— Un système de planning et de réservation de créneaux
— Un système de vente de jetons de laverie, lié aux comptoirs et au compte AE, permettant aux permanenciers de cliquer les jetons en même temps qu'ils vérifient l'état de la cotisation.
— Un système d'inventaire, pour gérer les différentes machines dans les différents lieux, et gérer également le retour des jetons après utilisation.

5.2 Principaux problèmes

5.2.1 Génération de plannings

Il y a là beaucoup de cas à prendre en compte. Lorsque que quelqu'un veut réserver directement un "Lavage + Séchage", un simple "Lavage", ou un simple "Séchage", il faut toujours vérifier la disponibilité des créneaux en fonction du nombre de machine de chaque type présent dans la laverie en question [1], et cela représente vite un grand nombre de combinaisons à vérifier.

De plus, la réservation doit rester ergonomique, et s'afficher dans un format le plus lisible possible pour un humain. Là dessus, un tableau est le plus approprié, avec chaque jour représenté par une colonne, et chaque créneau par une ligne. Mais cela ne représente malheureusement pas la temporalité, et la génération du tableau devient alors plutôt compliquée, et d'autant plus si l'on veut qu'il soit sémantiquement correct en HTML.

1. Belfort ou Sevenans, en l'occurrence

5.2.2 Gestion des timezones

La gestion et le stockage des crénaux implique l'utilisation de champs de type `DateTime`. **Django** les gère très bien, particulièrement au niveau des *timezones*, où ce dernier n'hésite pas à lancer un warning lorsque l'objet *date* passé ne contient pas d'information de fuseau horaire.

Mais avec notre décalage d'une heure par rapport au temps UTC, tous les horaires se retrouvent décalés, et gérer cela convenablement sans sortir d'avertissement a été plutôt compliqué. La solution a été de forcer un peu partout la *timezone* à UTC, afin de ne pas créer de décalage, mais en conservant tout de même l'information de fuseau horaire, et sans tout casser lors du passage à l'heure d'hiver.

6 La communication

Développeur principal : Skia

6.1 But

Cette application a plusieurs but :
— Donner la possibilité au responsable communication d'éditer les différents textes, messages, et pages statiques du site.
— Fournir un système de news.
— Fournir un système de newsletter : le Weekmail.

6.2 Principaux problèmes

6.2.1 Envoie de mails

Un outil de *newsletter* nécessite l'envoie de mail. C'est là quelque chose de relativement compliqué à tester, d'autant plus lorsqu'il s'agit de mailing-list contenant l'intégralité des étudiants de l'UTBM.

Django fournit toutefois un outil très pratique : il contient plusieurs *backend* d'emails, dont entre autre un *SMTP*, et un *console*. Le *SMTP* est bien évidemment utilisé en production pour envoyer effectivement les mails, mais il est compliqué à utiliser en développement, car il suppose que le développeur a à sa disposition un serveur de ce type. On utilise alors le backend *console*, qui affiche simplement dans le thread d'execution de **Django** une version texte de l'email envoyé, avec d'une part les entêtes, d'autre part le corps de message.

Mais autant pour tester l'envoie d'un mail unique à une adresse unique, cela fonctionne parfaitement bien, ce n'est toutefois pas suffisant pour tester un envoie massif à plusieurs mailings, avec en plus encore d'autres adresses en *Bcc* pour les gens ne faisant pas partie des mailings "classiques", mais souhaitant quand même recevoir le **Weekmail**.

6.2.2 Amélioration de l'outil de recherche

Pour la gestion de l'AE, il est nécessaire de pouvoir rechercher et trouver efficacement n'importe quel membre, en tapant au choix son nom, prénom, ou surnom, voire une combinaison de ces trois champs.

Mais une fonction de recherche aussi complexe est très difficile a mettre en place efficacement sans un traitement préalable, d'où la nécessité d'indexer les entrées à chercher. Un indexeur étant très complexe, mais également très courant, il n'a pas été difficile de trouver une application déjà existante fournissant ces fonctionnalités.

Le choix s'est porté sur **Haystack**, en l'utilisant avec l'indexeur **Whoosh**, plutôt efficace pour des bases raisonnables, et surtout écrit en pure **Python**, donc ne nécessitant pas d'installation compliquée en parallèle du site.

Le résultat est plutôt satisfaisant, mais il faudrait encore améliorer les résultats en utilisant les fonctions de *boost* pour certains champs. De plus, une certaine lenteur se fait encore sentir avec certaines recherches trop communes ou générales.

7 Conclusions personnelles

7.1 Skia

Développer de nouvelles application m'a permis d'apréhender d'autres problématiques, comme la gestion des fichiers dans le SAS, ou bien des contraintes de concurrence et d'atomicité sur l'Eboutic.

Mais la plus grosse partie de mon travail ce semestre a surtout été de superviser une équipe de développement naissante, de relire les "Merge request", et de m'assurer de la cohérence du code des contributeurs avec le reste du projet.

J'ai églament pu approfondir mon utilisation de Gitlab à travers ses outils de gestion de projet, de revue de code, et de gestion des permissions sur les différentes branches.

7.2 Lo-J

Je suis très heureux d'avoir pu participer à ce projet de TO52 sur le développement de modules sur le site de l'Association des Étudiants. J'ai pu apprendre à travailler avec un nouvel environnement informatique tout en contribuant au développement d'outils pour l'association dont je suis Président, le Bureau des Festivités.

7.2.1 Django

Ayant déjà travaillé avec le framework Spring et Java durant mon stage ST40, j'ai pu m'appuyer sur des notions générales afin d'apprendre et de comprendre le fonctionnement du Python et de Django que je ne connaissais pas du tout.

Mon apprentissage a été assez long au départ, car il y avait beaucoup d'informations à intégrer. Django est un framework très pratique qui permet d'effectuer de nombreuses tâches assez rébarbatives de manière automatique certes, mais encore faut-il comprendre ce qu'il se passe en arrière-plan. C'est cet apprentissage qui m'a pris le plus de temps.

Mon deuxième point de difficulté a été les formulaires. Là encore, Django est très pratique dès lors qu'il s'agit de faire un formulaire avec tous les champs d'un même Model. Cependant, il m'a fallu de l'aide et du temps pour comprendre

comment faire pour ajouter d'autres champs en plus de ceux du Model au formulaire et pour comprendre comment récupérer les valeurs associées à chacun d'eux.

7.2.2 Git

J'ai eu aussi à apprendre le fonctionnement de Git que j'avais déjà pu manipuler quelque peu mais il me manquait quand même beaucoup d'éléments.

Aujourd'hui, je pense pouvoir dire que j'ai progressé dans ce domaine mais il me reste encore bien des choses à apprendre pour être capable de l'utiliser de manière efficace.