Sith AE UTBM

Bartuccio Antoine (Sli), Brunet Pierre (Krohpil), Jacquet Florent (S

août 27, 2022

À propos du projet:

CHAPITRE 1

Introduction

Le but de ce projet est de fournir à l'Association des Étudiants de l'UTBM une plate-forme pratique et centralisée de ses services. Le Sith de l'AE tient à jour le registre des cotisations à l'association, prend en charge la trésorerie, les ventes de produits et services, la diffusion d'événements, la gestion de la laverie et bien plus encore. Il est conçu de manière suffisamment générique pour être facilement adaptable à une autre association.

C'est un projet bénévole qui tire ses origines des années 2000. Il s'agit de la troisième version du site de l'association initiée en 2015. C'est une réécriture complète en rupture totale des deux versions qui l'ont précédée.

1.1 Pourquoi réécrire le site

L'ancienne version du site, sobrement baptisée ae2 présentait un nombre impressionnant de fonctionnalités. Il avait été écrit en PHP et se basait sur son propre framework maison.

Malheureusement, son entretiens était plus ou moins hasardeux et son framework reposait sur des principes assez différents de ce qui se fait aujourd'hui, rendant la maintenance difficile. De plus, la version de PHP qu'il utilisait était plus que déprécié et à l'heure de l'arrivée de PHP 7 et de sa non rétrocompatibilité il était vital de faire quelque chose. Il a donc été décidé de le réécrire.

1.2 La philosophie du projet

Pour éviter les erreurs du passé, ce projet met l'accent sur la maintenabilité. Le choix des technologies ne s'est donc pas fait uniquement sur le fait qu'elle soit récentes, mais également sur leur robustesse, leur fiabilité et leur potentiel à être maintenu loin dans le futur.

La maintenabilité passe également par le choix minutieux des dépendances qui doivent eux aussi passer l'épreuve du temps pour éviter qu'elles ne mettent le projet en danger.

Cela passe également par la minimisation des frameworks employés de manière à réduire un maximum les connaissances nécessaires pour contribuer au projet et donc simplifier la prise en main. La simplicité est à privilégier si elle est possible.

Le projet doit être simple à installer et à déployer.

Le projet étant à destination d'étudiants, il est préférable de minimiser les ressources utilisées par l'utilisateur final. Il faut qu'il soit au maximum économe en bande passante et calcul côté client.

Le projet est un logiciel libre et est sous licence GPL. Aucune dépendance propriétaire ne sera acceptée.

CHAPITRE 2

Technologies utilisées

Bien choisir ses technologies est crucial puisqu'une fois que le projet est suffisamment avancé, il est très difficile voir impossible de revenir en arrière.

En novembre 2015, plusieurs choix s'offraient à nous :

— Continuer avec du PHP
— S'orienter vers un langage web plus moderne et à la mode comme le Python ou le Ruby
— Baser le site sur un framework Javascript

Le PHP 5, bientôt 7, de l'époque étant assez discutable comme cet article et l'ancien site ayant laissé un goût amer à certains développeurs, celui-ci a été mis de côté.

L'écosystème Javascript étant à peine naissant et les frameworks allant et venant en seulement quelques mois, il était impossible de prédire avec certitude si ceux-ci passeraient l'épreuve du temps, il était inconcevable de tout parier là dessus.

Ne restait plus que le Python et le Ruby avec les frameworks Django et Ruby On Rails. Ruby ayant une réputation d'être très « cutting edge », c'est Python, un langage bien implanté et ayant fait ses preuves, qui a été retenu.

2.1 Backend

2.1.1 Python 3

Site officiel

Le python est un langage de programmation interprété multi paradigme sorti en 1991. Il est très populaire pour sa simplicité d'utilisation, sa puissance, sa stabilité, sécurité ainsi que sa grande communauté de développeur. Sa version 3, non rétro compatible avec sa version 2, a été publiée en 2008.

Note : Puisque toutes les dépendances du backend sont des packages Python, elles sont toutes ajoutées directement dans le fichier **requirements.txt** à la racine du projet.

2.1.2 Django

Site officiel
Documentation

Django est un framework web pour Python apparu en 2005. Il fourni un grand nombre de fonctionnalités pour développer un site rapidement et simplement. Cela inclu entre autre un serveur Web de développement, un parseur d'URLs pour le routage des différentes URI du site, un ORM (Object-Relational Mapper) pour la gestion de la base de donnée ainsi qu'un moteur de templates pour le rendu HTML. Django propose une version LTS (Long Term Support) qui reste stable et est maintenu sur des cycles plus longs, ce sont ces versions qui sont utilisées.

2.1.3 PostgreSQL / SQLite

Site officiel PostgreSQL
Site officiel SQLite

Comme la majorité des sites internet, le Sith de l'AE enregistre ses données dans une base de donnée. Nous utilisons une base de donnée relationnelle puisque c'est la manière typique d'utiliser Django et c'est ce qu'utilise son ORM. Dans la pratique il arrive rarement dans le projet de se soucier de ce qui fonctionne derrière puisque le framework abstrait les requêtes au travers de son ORM. Cependant, il arrive parfois que certaines requêtes, lorsqu'on cherche à les optimiser, ne fonctionnent que sur un seul backend.

Le principal à retenir ici est :

— Sur la version de production nous utilisons PostgreSQL, c'est cette version qui doit fonctionner en priorité
— Sur les versions de développement, pour faciliter l'installation du projet, nous utilisons la technologie SQLite qui ne requiert aucune installation spécifique. Certaines instructions ne sont pas supportées par cette technologie et il est parfois nécessaire d'installer PostgreSQL pour le développement de certaines parties du site.

2.2 Frontend

2.2.1 Jinja2

Site officiel

Jinja2 est un moteur de template écrit en Python qui s'inspire fortement de la syntaxe des templates de Django. Ce moteur apporte toutefois son lot d'améliorations non négligeables. Il permet par exemple l'ajout de macros, sortes de fonctions écrivant du HTML.

Un moteur de templates permet de générer du contenu textuel de manière procédural en fonction des données à afficher, cela permet de pouvoir inclure du code proche du Python dans la syntaxe au milieu d'un document contenant principalement du HTML. On peut facilement faire des boucles ou des conditions ainsi même que de l'héritage de templates.

Attention : le rendu est fait côté serveur, si on souhaite faire des modifications côté client, il faut utiliser du Javascript, rien ne change à ce niveau là.

Exemple d'utilisation d'un template Jinja2

```
<title>{% block title %}{% endblock %}</title>
<ul>
{% for user in users %}
```

(suite sur la page suivante)

(suite de la page précédente)

```
  <li><a href="{{ user.url }}">{{ user.username }}</a></li>
{% endfor %}
</ul>
```

2.2.2 jQuery

Site officiel

jQuery est une bibliothèque JavaScript libre et multiplateforme créée pour faciliter l'écriture de scripts côté client dans le code HTML des pages web. La première version est lancée en janvier 2006 par John Resig.

C'est une vieille technologie et certains feront remarquer à juste titre que le Javascript moderne permet d'utiliser assez simplement la majorité de ce que fourni jQuery sans rien avoir à installer. Cependant, de nombreuses dépendances du projet utilisent encore jQuery qui est toujours très implanté aujourd'hui. Le sucre syntaxique qu'offre cette libraire reste très agréable à utiliser et économise parfois beaucoup de temps. Ça fonctionne et ça fonctionne très bien. C'est maintenu, léger et pratique, il n'y a pas de raison particulière de s'en séparer.

2.2.3 Sass

Site officiel

Sass (Syntactically Awesome Stylesheets) est un langage dynamique de génération de feuilles CSS apparu en 2006. C'est un langage de CSS « amélioré » qui permet l'ajout de variables (à une époque où le CSS ne les supportait pas), de fonctions, mixins ainsi qu'une syntaxe pour imbriquer plus facilement et proprement les règles sur certains éléments. Le Sass est traduit en CSS directement côté serveur et le client ne reçoit que du CSS.

C'est une technologie stable, mature et pratique qui ne nécessite pas énormément d'apprentissage.

2.2.4 Fontawesome

Site officiel

Fontawesome regroupe tout un ensemble d'icônes libres de droits utilisables facilement sur n'importe quelle page web. Ils sont simple à modifier puisque modifiables via le CSS et présentent l'avantage de fonctionner sur tous les navigateurs contrairement à un simple icône unicode qui s'affiche lui différemment selon la plate-forme.

Note : C'est une dépendance capricieuse qu'il évolue très vite et qu'il faut très souvent mettre à jour.

Avertissement : Il a été décidé de **ne pas utiliser** de CDN puisque le site ralentissait régulièrement. Il est préférable de fournir cette dépendance avec le site.

2.3 Documentation

2.3.1 Sphinx

Site officiel

Sphinx est un outil qui permet la création de documentations intelligentes et très jolies. C'est cet outil qui permet d'écrire le documentation que vous êtes en train de lire actuellement. Développé en 2008 pour la communauté Python, c'est l'outil le plus répandu. Il est utilisé pour la documentation officielle de Python, pour celle de Django, Jinja2 et bien d'autres.

2.3.2 ReadTheDocs

Site officiel

C'est un site d'hébergement de documentations utilisant Sphinx. Il propose la génération de documentation à partir de sources et leur hébergement gracieusement pour tout projet open source. C'est le site le plus utilisé et sur lequel sont hébergées bon nombre de documentations comme par exemple celle de Django. La documentation sur ce site est automatiquement générée à chaque nouvelle modification du projet.

2.3.3 reStructuredText

Site officiel

C'est un langage de balisage léger utilisé notamment dans la documentation du langage Python. C'est le langage dans lequel est écrit l'entièreté de la documentation ci-présente pour que Sphinx puisse la lire et la mettre en forme.

2.4 Workflow

2.4.1 Git

Site officiel

Git est un logiciel de gestion de versions écrit par Linus Torsvald pour les besoins du noyau linux en 2005. C'est ce logiciel qui remplace svn anciennement utilisé pour gérer les sources du projet (rappelez vous, l'ancien site date d'avant 2005). Git est plus complexe à utiliser mais est bien plus puissant, permet de gérer plusieurs version en parallèle et génère des codebases vraiment plus légères puisque seules les modifications sont enregistrées (contrairement à svn qui garde une copie de la codebase par version).

2.4.2 GitLab

Site officiel
Instance de l'AE

GitLab est une alternative libre à GitHub. C'est une plate-forme avec interface web permettant de déposer du code géré avec Git offrant également de l'intégration continue et du déploiement automatique.

C'est au travers de cette plate-forme que le Sith de l'AE est géré, sur une instance hébergée directement sur nos serveurs.

2.4.3 Sentry

Site officiel
Instance de l'AE

Sentry est une plate-forme libre qui permet de se tenir informer des bugs qui ont lieu sur le site. À chaque crash du logiciel (erreur 500), une erreur est envoyée sur la plate-forme et est indiqué précisément à quelle ligne de code celle-ci a eu lieu, à quelle heure, combien de fois, avec quel navigateur la page a été visitée et même éventuellement un commentaire de l'utilisateur qui a rencontré le bug.

2.4.4 Poetry

Utiliser Poetry

Poetry est un utilitaire qui permet de créer et gérer des environements Python de manière simple et intuitive. Il permet également de gérer et mettre à jour le fichier de dépendances. L'avantage d'utiliser poetry (et les environnements virtuels en général) est de pouvoir gérer plusieurs projets différents en parallèles puisqu'il permet d'avoir sur sa machine plusieurs environnements différents et donc plusieurs versions d'une même dépendance dans plusieurs projets différent sans impacter le système sur lequel le tout est installé.

2.4.5 Black

Site officiel

Pour faciliter la lecture du code, il est toujours appréciable d'avoir une norme d'écriture cohérente. C'est généralement à l'étape de relecture des modifications par les autres contributeurs que sont repérées ces fautes de normes qui se doivent d'être corrigées pour le bien commun.

Imposer une norme est très fastidieux, que ce soit pour ceux qui relisent ou pour ceux qui écrivent. C'est pour cela que nous utilisons black qui est un formateur automatique de code. Une fois l'outil lancé, il parcours la codebase pour y repérer les fautes de norme et les corrige automatiquement sans que l'utilisateur ai à s'en soucier. Bien installé, il peut effectuer ce travail à chaque sauvegarde d'un fichier dans son éditeur, ce qui est très agréable pour travailler.

CHAPITRE 3

Le versioning

Dans le monde du développement, nous faisons face à un problème relativement étrange pour un domaine aussi avancé : on est brouillon.

On teste, on envoie, ça marche pas, on reteste, c'est ok. Par contre, on a oublie plein d'exceptions. Et on refactor. Ça marche mieux mais c'est moins rapide, etc, etc.

Et derrière tout ça, on fait des trucs qui marchent puis on se retrouve dans la mouise parce qu'on a effacé un morceau de code qui nous aurait servi plus tard.

Pour palier à ce problème, le programmeur a créé un principe révolutionnaire (ouais… à mon avis, on s'est inspiré d'autres trucs, mais on va dire que c'est nous les créateurs) : le Versioning (*Apparition inexpliquée*).

D'après projet-isika (c'est pas wikipedia ouais, ils étaient pas clairs eux), le versioning (ou versionnage en français mais c'est quand même vachement dégueu comme mot) consiste à travailler directement sur le code source d'un projet, en gardant toutes les versions précédentes. Les outils du versioning aident les développeurs à travailler parallèlement sur différentes parties du projet et à revenir facilement aux étapes précédentes de leur travail en cas de besoin. L'utilisation d'un logiciel de versioning est devenue quasi-indispensable pour tout développeur, même s'il travaille seul.

3.1 Un versioning pour les gouverner tous

On va vite fait passer sur les différents logiciels de contrôle de version avant de revenir à l'essentiel, le vrai, le beau, l'unique et l'ultime : Git.

Source Code Control System (SCCS)) : Développé en 1972 dans les labos d'IBM, il a été porté sur Unix pour ensuite donner naissance à RCS. **GNU RCS (Revision Control System)** : RCS est à l'origine un projet universitaire, initié au début des années 1980, et maintenu pendant plus d'une décennie par Walter F. Tichy au sein de l'université Purdue.

Ce logiciel représente à l'époque une alternative libre au système SCCS, et une évolution technique, notamment par son interface utilisateur, plus conviviale, et une récupération des données, plus rapide, par l'amélioration du stockage des différentes versions. Ce gain de performance provient d'un algorithme appelé en anglais « reverse differences » (ou plus simplement « deltas ») et consiste à stocker la copie complète des versions les plus récentes et conserver uniquement les changements réalisés.

CVS (Concurrent Versions System) : En gros, c'est la première fois qu'on essaie de fusionner des versions *concurrentes* (dis-donc, quel hasard que ce soit des concurrents vu le nom du système !) de fichiers sources. C'était pas forcément compliqué : en gros, il y avait un serveur qui prenait à chaque fois la dernière version de chaque fichier, les développeurs devaient toujours avoir la dernière version du fichier s'ils voulaient éditer celui-ci. Si c'était pas le cas, le serveur les envoyait paitre.

SVN (Subversion) : En gros, c'est comme CVS mais avec quelques améliorations du fait du refactoring complet fait par Apache. Subversion permet notamment le renommage et le déplacement de fichiers ou de répertoires sans en perdre l'historique. On a aussi un versioning sur les metadatas (genre les changements de permissions des fichiers.

Git : Enfin le voilà. Le versioning ultime. Créé par Linus Torvalds en 2005, il permet notamment au bordel qu'est Linux d'être maintenu par des développeurs du monde entier grâce à un système original de version : en gros, chaque ordinateur a une version du code source et il n'y a pas forcément un serveur central qui garde tout (et demande un compte à chaque fois. Bon, maintenant on est de retour au format minitel avec Github mais on va vous montrer comment s'en sortir). Il y a également un système de branche pour pouvoir gérer différentes versions du code en parallèle. Tout est fait sous forme de petits fichiers de versioning qui vont faire des copies des fichiers correspondant à la modification proposée. Bref, c'est trop bien et on a pas fait mieux.

C'est pas forcément utile de comprendre le fonctionnement interne de Git pour développer (la preuve, je n'ai pas franchement chercher au tréfond du bousin) mais c'est en revanche indispensable de comprendre comment l'utiliser avant de faire n'importe quoi. Du coup, on va voir ci-dessous comment utiliser Git et comment on l'utilise sur le site AE.

3.2 TLDR

Un système de versioning permet de faire de la merde dans votre code et de pouvoir revenir en arrière malgré tout. Ça permet aussi de coder à plusieurs. Git est le meilleur système de gestion de version (ou système de versioning) que vous pourrez trouver à l'heure actuelle. Utilisez-le.

CHAPITRE 4

Installer le projet

4.1 Dépendances du système

Certaines dépendances sont nécessaires niveau système :

- poetry
- libmysqlclient
- libssl
- libjpeg
- libxapian-dev
- zlib1g-dev
- python
- gettext
- graphviz
- mysql-client (pour migrer de l'ancien site)

4.1.1 Sur Ubuntu

```
sudo apt install libssl-dev libjpeg-dev zlib1g-dev python-dev libffi-dev python-dev␣
↪libgraphviz-dev pkg-config libxapian-dev gettext git
curl -sSL https://raw.githubusercontent.com/python-poetry/poetry/master/get-poetry.py |␣
↪python -

# To include mysql for importing old bdd
sudo apt install libmysqlclient-dev
```

4.1.2 Sur MacOS

Pour installer les dépendances, il est fortement recommandé d'installer le gestionnaire de paquets homebrew.

```
brew install git python xapian graphviz poetry

# Si vous aviez une version de python ne venant pas de homebrew
brew link --overwrite python


# Pour bien configurer gettext
brew link gettext # (suivez bien les instructions supplémentaires affichées)

# Pour installer poetry
pip3 install poetry
```

Note : Si vous rencontrez des erreurs lors de votre configuration, n'hésitez pas à vérifier l'état de votre installation homebrew avec `brew doctor`

4.1.3 Sur Windows avec `WSL`

Note : Comme certaines dépendances sont uniquement disponible dans un environnement Unix, il est obligatoire de passer par `WSL` pour installer le projet.

— **Prérequis :** vous devez exécuter Windows 10 versions 2004 et ultérieures (build 19041 & versions ultérieures) ou Windows 11.
— **Plus d'info :** docs.microsoft.com

```
# dans un shell Windows
wsl --install

# afficher la liste des distribution disponible avec WSL
wsl -l -o

# installer WSL avec une distro
wsl --install -d <nom_distro>
```

Note : Si vous rencontrez le code d'erreur `0x80370102`, regardez les réponses de ce post.

Une fois `WSL` installé, mettez à jour votre distro & installez les dépendances **(voir la partie installation sous Ubuntu)**.

Note : Comme *git* ne fonctionne pas de la même manière entre Windows & Unix, il est nécessaire de cloner le repository depuis Windows. (cf : stackoverflow.com)

Pour accéder au contenu d'un répertoire externe à `WSL`, il suffit simplement d'utiliser la commande suivante :

```
# oui c'est beau, simple et efficace
cd /mnt/<la_lettre_du_disque>/vos/fichiers/comme/dhab
```

Note : Une fois l'installation des dépendances terminée (juste en dessous), il vous suffira, pour commencer à dev, d'ouvrir votre plus bel IDE et d'avoir 2 consoles : 1 console `WSL` pour lancer le projet & 1 console pour utiliser `git`

4.2 Installer le projet

```
# Sait-on jamais
sudo apt update

# Les commandes git doivent se faire depuis le terminal de Windows si on utilise WSL !
git clone https://github.com/ae-utbm/sith3.git
cd Sith

# Création de l'environnement et installation des dépendances
poetry install

# Activation de l'environnement virtuel
poetry shell

# Prépare la base de donnée
python manage.py setup

# Installe les traductions
python manage.py compilemessages
```

Note : Pour éviter d'avoir à utiliser la commande poetry shell systématiquement, il est possible de consulter *Utiliser direnv*.

4.3 Configuration pour le développement

Lorsqu'on souhaite développer pour le site, il est nécessaire de passer le logiciel en mode debug dans les settings_custom. Il est aussi conseillé de définir l'URL du site sur localhost. Voici un script rapide pour le faire.

```
echo "DEBUG=True" > sith/settings_custom.py
echo 'SITH_URL = "localhost:8000"' >> sith/settings_custom.py
```

4.4 Démarrer le serveur de développement

Il faut toujours avoir préalablement activé l'environnement virtuel comme fait plus haut et se placer à la racine du projet. Il suffit ensuite d'utiliser cette commande :

```
python manage.py runserver
```

Note : Le serveur est alors accessible à l'adresse http://localhost:8000.

4.5 Générer la documentation

La documentation est automatiquement mise en ligne sur readthedocs à chaque envoi de code sur GitLab. Pour l'utiliser en local ou globalement pour la modifier, il existe une commande du site qui génère la documentation et lance un serveur la rendant accessible à l'adresse http://localhost:8080. Cette commande génère la documentation à chacune de ses modifications, inutile de relancer le serveur à chaque fois.

```
python manage.py documentation

# Il est possible de spécifier un port et une adresse d'écoute différente
python manage.py documentation adresse:port
```

4.6 Lancer les tests

Pour lancer les tests il suffit d'utiliser la commande intégrée à django.

```
# Lancer tous les tests
python manage.py test

# Lancer les tests de l'application core
python manage.py test core

# Lancer les tests de la classe UserRegistrationTest de core
python manage.py test core.tests.UserRegistrationTest

# Lancer une méthode en particulier de cette même classe
python manage.py test core.tests.UserRegistrationTest.test_register_user_form_ok
```

4.7 Vérifier les dépendances Javascript

Une commande a été écrite pour vérifier les éventuelles mises à jour à faire sur les librairies Javascript utilisées. N'oubliez pas de mettre à jour à la fois le fichier de la librairie, mais également sa version dans *sith/settings.py*.

```
# Vérifier les mises à jour
python manage.py check_front
```

CHAPITRE 5

La structure du projet

5.1 Le principe MVT

Django est un framework suivant le modèle MVT (Model-View-Template) aussi appelé MTV (Model-Template-View).

Fig. 1 – Diagramme MVT

On peut ainsi voir que la Vue gère la logique d'application, le modèle gère la structure de la base de données et communique avec elle et la vue effectue la logique de l'application. Décris comme ça, cela fait penser au modèle MVC (Model-View-Controller) mais évitons de nous complexifier les choses. Disons que c'est assez proche mais qu'il y a quelques différences (déjà au niveau du nommage).

On peut également représenter le tout sous une autre forme, plus simple à comprendre et visualiser en aplatissant le diagramme. Cela représente mieux ce qui se passe.

Fig. 2 – Diagramme MVT aplati

Cette représentation permet de se représenter les interactions sous formes de couches. Avec ça en tête, ce sera plus simple d'appréhender la manière dont est découpé le projet.

5.2 Le découpage en applications

/projet

sith/
: Application principale du projet.

accounting/
: Ajoute un système de comptabilité.

api/

Application où mettre les endpoints publiques d'API.

club/
: Contiens les modèles liés aux clubs associatifs et ajoute leur gestion.

com/
: Fournis des outils de communications aux clubs (weekmail, affiches…).

core/
: Application la plus importante. Contiens les principales surcouches
liées au projet comme la gestion des droits et les templates de base.

counter/
: Ajoute des comptoirs de vente pour les clubs et gère les ventes sur les lieux de vie.

data/
: Contiens les fichiers statiques ajoutées par les utilisateurs.
N'est pas suivit par Git.

doc/
: Contiens la documentation du projet.

eboutic/
: Ajoute le comptoir de vente en ligne. Permet d'acheter en carte bancaire.

election/
: Ajoute un système d'élection permettant d'élire les représentants étudiants.

forum/
: Ajoute un forum de discutions.

launderette/
: Permet la gestion des laveries.

locale/
: Contiens les fichiers de traduction.

matmat/
: Système de recherche de membres.

pedagogy/
: Contiens le guide des UVs.

rootplace/
: Ajoute des outils destinés aux administrateurs.

static/
: Contiens l'ensemble des fichiers statiques ajoutés par les développeurs.
Ce dossier est généré par le framework, il est surtout utile en production.
Ce dossier n'es pas suivit par Git.

stock/
: Système de gestion des stocks.

subscription/
: Ajoute la gestion des cotisations des membres.

trombi/
: Permet la génération du trombinoscope des élèves en fin de cursus.

.coveragec
: Configure l'outil permettant de calculer la couverture des tests sur le projet.

.gitignore
: Permet de définir quels fichiers sont suivis ou non par Git.

.gitlab-ci.yml
: Permet de configurer la pipeline automatique de GitLab.

.readthedocs.yml
: Permet de configurer la génération de documentation sur Readthedocs.

.db.sqlite3
: Base de données de développement par défaut. Est automatiquement généré lors de la configuration du projet en local. N'est pas suivis par Git.

LICENSE
: Licence du projet.

LICENSE.old
: Ancienne licence du projet.

manage.py
: Permet de lancer les commandes liées au framework Django.

migrate.py
: Contiens des scripts de migration à exécuter pour importer les données de l'ancien site.

README.rst
: Fichier de README. À lire pour avoir des informations sur le projet.

requirements.txt
: Contiens les dépendances Python du projet.

5.3 L'application principale

/sith

__init__.py
: Permet de définir le dossier comme un package Python.
Ce fichier est vide.

settings.py
: Contiens les paramètres par défaut du projet.
Ce fichier est versionné et fait partie intégrant de celui-ci.

settings_curtom.py
: Contiens les paramètres spécifiques à l'installation courante.
Ce fichier n'est pas versionné et surcharges les paramètres par défaut.

urls.py
: Contiens les routes d'URLs racines du projet.
On y inclus les autres fichiers d'URLs et leur namespace.

toolbar_debug.py
: Contiens la configuration de la barre de debug à gauche à destination du site de développement.

et_keys/
: Contiens la clef publique du système de paiement E-Transactions.

Avertissement : Ne pas mettre de configuration personnelle ni aucun mot de passe dans **settings.py**. Si il y a besoin de ce genre de chose, il faut le mettre dans **settings_custom.py** qui lui n'est pas versionné.

5.4 Le contenu d'une application

/app1

__init__.py
: Permet de définir le dossier comme un package Python.
Ce fichier est généralement vide.

models.py
: C'est là que les modèles sont définis. Ces classes définissent
les tables dans la base de donnée.

views.py
: C'est là où les vues sont définies.

admin.py
: C'est là que l'on déclare quels modèles doivent apparaître
dans l'interface du module d'administration de Django.

tests.py
: Ce fichier contiens les tests fonctionnels, unitaires
mais aussi d'intégrations qui sont lancés par la pipeline.

urls.py
: On y défini les URLs de l'application et on les lies aux vues.

migrations/
: Ce dossier sert à stocker les fichiers de migration de la base
de données générées par la commande *makemigrations*.

templates/
: Ce dossier ci contiens généralement des sous dossiers et sert
à accueillir les templates. Les sous dossiers servent de namespace.

CHAPITRE 6

Créer une nouvelle application Hello World

L'objectif de ce petit tutoriel est de prendre rapidement en main les vues, les urls et les modèles de Django. On créera une application nommée *hello* qui fournira une page affichant « Hello World », une autre page qui affichera en plus un numéro qui sera récupéré depuis l'URL ainsi qu'une page affichant un élément récupéré de la base de données, le tout au milieu d'une page typique du Sith AE.

6.1 Créer l'application

La première étape est de crée l'application. Cela se fait très simplement avec les outils fournis par le framework.

```
./manage.py startapp hello
```

Il faut ensuite activer l'application dans les paramètres du projet en l'ajoutant dans la liste des applications installées.

```
# sith/settings.py

# ...

INSTALLED_APPS = (
    ...
    "hello",
)
```

Enfin, on vas inclure les URLs de cette application dans le projet sous le préfixe */hello/*.

```
# sith/urls.py
urlpatterns = [
    ...
    url(r"^hello/", include("hello.urls", namespace="hello", app_name="hello")),
]
```

6.2 Un Hello World simple

Dans un premier temps, nous allons créer une vue qui vas charger un template en utilisant le système de vues basées sur les classes de Django.

```
# hello/views.py

from django.views.generic import TemplateView

# Toute la logique pour servir la vue et parser le template
# est directement héritée de TemplateView
class HelloView(TemplateView):
    template_name = "hello/hello.jinja" # On indique quel template utiliser
```

On vas ensuite créer le template.

```
{# hello/templates/hello/hello.jinja #}

{# On étend le template de base du Sith #}
{% extends "core/base.jinja" %}

{# On remplis la partie titre du template étendu #}
{# Il s'agit du titre qui sera affiché dans l'onglet du navigateur #}
{% block title %}
Hello World
{% endblock title %}

{# On remplis le contenu de la page #}
{% block content %}
<p>Hello World !</p>
{% endblock content %}
```

Enfin, on crée l'URL. On veut pouvoir appeler la page depuis https://localhost:8000/hello, le préfixe indiqué précédemment suffit donc.

```
# hello/urls.py
from django.conf.urls import url
from hello.views import HelloView

urlpatterns = [
   # Le préfixe étant retiré lors du passage du routeur d'URL
   # dans le fichier d'URL racine du projet, l'URL à matcher ici est donc vide
   url(r"^$", HelloView.as_view(), name="hello"),
]
```

Et voilà, c'est fini, il ne reste plus qu'à lancer le serveur et à se rendre sur la page.

6.3 Manipuler les arguments d'URL

Dans cette partie, on cherche à détecter les numéros passés dans l'URL pour les passer dans le template. On commence par ajouter cet URL modifiée.

```
# hello/urls.py
from django.conf.urls import url
from hello.views import HelloView

urlpatterns = [
   url(r"^$", HelloView.as_view(), name="hello"),
   # On utilise un regex pour matcher un numéro
   url(r"^(?P<hello_id>[0-9]+)$", HelloView.as_view(), name="hello"),
]
```

Cette deuxième URL vas donc appeler la classe crée tout à l'heure en lui passant une variable *hello_id* dans ses *kwargs*, nous allons la récupérer et la passer dans le contexte du template en allant modifier la vue.

```
# hello/views.py
from django.views.generic import TemplateView

class HelloView(TemplateView):
    template_name = "hello/hello.jinja"

    # C'est la méthode appelée juste avant de définir le type de requête effectué
    def dispatch(self, request, *args, **kwargs):

        # On récupère l'ID et on le met en attribut
        self.hello_id = kwargs.pop("hello_id", None)

        # On reprend le déroulement normal en appelant la méthode héritée
        return super(HelloView, self).dispatch(request, *args, **kwargs)

    # Cette méthode renvoie les variables qui seront dans le contexte du template
    def get_context_data(self, **kwargs):

        # On récupère ce qui était sensé être par défaut dans le contexte
        kwargs = super(HelloView, self).get_context_data(**kwargs)

        # On ajoute notre ID
        kwargs["hello_id"] = self.hello_id

        # On renvoie le contexte
        return kwargs
```

Enfin, on modifie le template en rajoutant une petite condition sur la présence ou non de cet ID pour qu'il s'affiche.

```
{# hello/templates/hello/hello.jinja #}
{% extends "core/base.jinja" %}

{% block title %}
Hello World
{% endblock title %}
```

(suite sur la page suivante)

(suite de la page précédente)

```
{% block content %}
<p>
    Hello World !
    {% if hello_id -%}
    {{ hello_id }}
    {%- endif -%}
</p>
{% endblock content %}
```

Note : Il est tout à fait possible d'utiliser les arguments GET passés dans l'URL. Dans ce cas, il n'est pas obligatoire de modifier l'URL et il est possible de récupérer l'argument dans le dictionnaire *request.GET*.

6.4 À l'assaut des modèles

Pour cette dernière partie, nous allons ajouter une entrée dans la base de donnée et l'afficher dans un template. Nous allons ainsi créer un modèle nommé *Article* qui contiendra une entrée de texte pour le titre et une autre pour le contenu.

Commençons par le modèle en lui même.

```
# hello/models.py
from django.db import models


class Article(models.Model):

    title = models.CharField("titre", max_length=100)
    content = models.TextField("contenu")
```

Continuons avec une vue qui sera en charge d'afficher l'ensemble des articles présent dans la base.

```
# hello/views.py

from django.views.generic import ListView

from hello.models import Article

...

# On hérite de ListView pour avoir plusieurs objets
class ArticlesListView(ListView):

    model = Article # On base la vue sur le modèle Article
    template_name = "hello/articles.jinja"
```

On n'oublie pas l'URL.

```
from hello.views import HelloView, ArticlesListView
```

(suite sur la page suivante)

(suite de la page précédente)

```
urlpatterns = [
    ...
    url(r"^articles$", ArticlesListView.as_view(), name="articles_list")
]
```

Et enfin le template.

```
{# hello/templates/hello/articles.jinja #}
{% extends "core/base.jinja" %}

{% block title %}
    Hello World Articles
{% endblock title %}

{% block content %}
    {# Par défaut une liste d'objets venant de ListView s'appelle object_list #}
    {% for article in object_list %}
        <h2>{{ article.title }}</h2>
        <p>{{ article.content }}</p>
    {% endfor %}
{% endblock content %}
```

Maintenant que toute la logique de récupération et d'affichage est terminée, la page est accessible à l'adresse https://localhost:8000/hello/articles.

Mais, j'ai une erreur ! Il se passe quoi ? ! Et bien c'est simple, nous avons crée le modèle mais il n'existe pas dans la base de données. Il est dans un premier temps important de créer un fichier de migrations qui contiens des instructions pour la génération de celle-ci. Ce sont les fichiers qui sont enregistrés dans le dossier migration. Pour les générer à partir des classes de modèles qu'on viens de manipuler il suffit d'une seule commande.

```
./manage.py makemigrations
```

Un fichier *hello/migrations/0001_initial.py* se crée automatiquement, vous pouvez même aller le voir.

Note : Il est tout à fait possible de modifier à la main les fichiers de migrations. C'est très intéressant si par exemple il faut appliquer des modifications sur les données d'un modèle existant après cette migration mais c'est bien au delà du sujet de ce tutoriel. Référez vous à la documentation pour ce genre de choses.

J'ai toujours une erreur ! Mais oui, c'est pas fini, faut pas aller trop vite. Maintenant il faut appliquer les modifications à la base de données.

```
./manage.py migrate
```

Et voilà, là il n'y a plus d'erreur. Tout fonctionne et on a une superbe page vide puisque aucun contenu pour cette table n'est dans la base. Nous allons en rajouter. Pour cela nous allons utiliser le fichier *core/management/commands/populate.py* qui contiens la commande qui initialise les données de la base de données de test. C'est un fichier très important qu'on viendra à modifier assez souvent. Nous allons y ajouter quelques articles.

```
# core/management/commands/populate.py
from hello.models import Article


...
```

(suite sur la page suivante)

(suite de la page précédente)

```
class Command(BaseCommand):

    ...

    def handle(self, *args, **options):

        ...

        Article(title="First hello", content="Bonjour tout le monde").save()
        Article(title="Tutorial", content="C'était un super tutoriel").save()
```

On regénère enfin les données de test en lançant la commande que l'on viens de modifier.

```
./manage.py setup
```

On reviens sur https://localhost:8000/hello/articles et cette fois-ci nos deux articles apparaissent correctement.

CHAPITRE 7

Ajouter une traduction

Le code du site est entièrement écrit en anglais, le texte affiché aux utilisateurs l'est également. La traduction en français se fait ultérieurement avec un fichier de traduction. Voici un petit guide rapide pour apprendre à s'en servir.

7.1 Dans le code du logiciel

Imaginons que nous souhaitons afficher « Hello » et le traduire en français. Voici comment signaler que ce mot doit être traduit.

Si le mot est dans le code Python :

```
from django.utils.translation import gettext_lazy as _

# ...

help_text=_("Hello")

# ...
```

Si le mot apparaît dans le template Jinja :

```
{# ... #}

{% trans %}Hello{% endtrans %}

{# ... #}
```

7.2 Générer le fichier django.po

La traduction se fait en trois étapes. Il faut d'abord générer un fichier de traductions, l'éditer et enfin le compiler au format binaire pour qu'il soit lu par le serveur.

```
./manage.py makemessages --locale=fr --ignore "env/*" -e py,jinja
```

7.3 Éditer le fichier django.po

```
# locale/fr/LC_MESSAGES/django.po

# ...
msgid "Hello"
msgstr "" # Ligne à modifier

# ...
```

Note : Si les commentaires suivants apparaissent, pensez à les supprimer. Ils peuvent gêner votre traduction.

```
#, fuzzy
#| msgid "Bonjour"
```

7.4 Générer le fichier django.mo

Il s'agit de la dernière étape. Un fichier binaire est généré à partir du fichier django.mo.

```
./manage.py compilemessages
```

Note : Pensez à redémarrer le serveur si les traductions ne s'affichent pas

CHAPITRE 8

Configurer son environnement de développement

Le projet n'est en aucun cas lié à un quelconque environnement de développement. Il est possible pour chacun de travailler avec les outils dont il a envie et d'utiliser l'éditeur de code avec lequel il est le plus à l'aise.

Pour donner une idée, Skia a écrit une énorme partie de projet avec l'éditeur *vim* sur du GNU/Linux alors que Sli a utilisé *Sublime Text* sur MacOS.

8.1 Configurer Black pour son éditeur

Tous les détails concernant l'installation de black sont ici : https://black.readthedocs.io/en/stable/editor_integration.html

Néanmoins, nous tenterons de vous faire ici un résumé pour deux éditeurs de textes populaires que sont VsCode et Sublime Text.

```
# Installation de black
pip install black
```

8.1.1 VsCode

Avertissement : Il faut installer black dans son environement virtuel pour cet éditeur

Black est directement pris en charge par l'extension pour le Python de VsCode, il suffit de rentrer la configuration suivante :

```
{
    "python.formatting.provider": "black",
    "editor.formatOnSave": true
}
```

8.1.2 Sublime Text

Il est tout d'abord nécessaire d'installer ce plugin : https://packagecontrol.io/packages/sublack.

Il suffit ensuite d'ajouter dans les settings du projet (ou directement dans les settings globales) :

```
{
    "sublack.black_on_save": true
}
```

Si vous utilisez le plugin anaconda, pensez à modifier les paramètres du linter pep8 pour éviter de recevoir des warnings dans le formatage de black comme ceci :

```
{
    "pep8_ignore": [
      "E203",
      "E266",
      "E501",
      "W503"
    ]
}
```

CHAPITRE 9

La gestion des droits

La gestion des droits dans l'association étant très complexe, le système de permissions intégré dans Django ne suffisait pas, il a donc fallu créer et intégrer le notre.

La gestion des permissions se fait directement sur un modèle, il existe trois niveaux de permission :
- Édition des propriétés de l'objet
- Édition de certaines valeurs l'objet
- Voir l'objet

9.1 Protéger un modèle

Lors de l'écriture d'un modèle, il est très simple de définir des permissions. Celle-ci peuvent être basées sur des groupes et/ou sur des fonctions personnalisées.

Nous allons présenter ici les deux techniques. Dans un premier temps nous allons protéger une classe Article par groupes.

```
from django.db import models
from django.conf import settings
from django.utils.translation import gettext_lazy as _

from core.views import *
from core.models import User, Group

# Utilisation de la protection par groupe
class Article(models.Model):

    title = models.CharField(_("title"), max_length=100)
    content = models.TextField(_("content"))

    # Ne peut pas être une liste
    # Groupe possédant l'objet
```

(suite sur la page suivante)

(suite de la page précédente)

```
    # Donne les droits d'édition des propriétés de l'objet
    owner_group = models.ForeignKey(
        Group, related_name="owned_articles", default=settings.SITH_GROUP_ROOT_ID
    )

    # Doit être une liste
    # Tous les groupes qui seront ajouté dans ce champ auront les droits d'édition de l
↪'objet
    edit_group = models.ManyToManyField(
        edit_groups = models.ManyToManyField(
            Group,
            related_name="editable_articles",
            verbose_name=_("edit groups"),
            blank=True,
        )
    )

    # Doit être une liste
    # Tous les groupes qui seront ajouté dans ce champ auront les droits de vue de l'objet
    view_groups = models.ManyToManyField(
        Group,
        related_name="viewable_articles",
        verbose_name=_("view groups"),
        blank=True,
    )
```

Voici maintenant comment faire en définissant des fonctions personnalisées. Cette deuxième solution permet, dans le cas où la première n'est pas assez puissante, de créer des permissions complexes et fines. Attention à bien les rendre lisibles et de bien documenter.

```
from django.db import models
from django.utils.translation import gettext_lazy as _

from core.views import *
from core.models import User, Group

# Utilisation de la protection par fonctions
class Article(models.Model):

    title = models.CharField(_("title"), max_length=100)
    content = models.TextField(_("content"))

    # Donne ou non les droits d'édition des propriétés de l'objet
    # Un utilisateur dans le bureau AE aura tous les droits sur cet objet
    def is_owned_by(self, user):
        return user.is_board_member

    # Donne ou non les droits d'édition de l'objet
    # L'objet ne sera modifiable que par un utilisateur cotisant
    def can_be_edited_by(self, user):
        return user.is_subscribed
```

(suite sur la page suivante)

(suite de la page précédente)

```
    # Donne ou non les droits de vue de l'objet
    # Ici, l'objet n'est visible que par un utilisateur connecté
    def can_be_viewed_by(self, user):
        return not user.is_anonymous
```

Note : Si un utilisateur est autorisé à un niveau plus élevé que celui auquel il est testé, le système le détectera automatiquement et les droits lui seront accordé. Par défaut, les seuls utilisateurs ayant des droits quelconques sont les *superuser* de Django et les membres du bureau AE qui sont définis comme *owner*.

9.2 Appliquer les permissions

9.2.1 Dans un template

Il existe trois fonctions de base sur lesquelles reposent les vérifications de permission. Elles sont disponibles dans le contexte par défaut du moteur de template et peuvent être utilisées à tout moment.

Tout d'abord, voici leur documentation et leur prototype.

core.views.**can_edit_prop**(*obj*, *user*)

Paramètres
- **obj** – Object to test for permission
- **user** – core.models.User to test permissions against

Renvoie if user is authorized to edit object properties

Type renvoyé bool

Example

```
if not can_edit_prop(self.object ,request.user):
    raise PermissionDenied
```

core.views.**can_edit**(*obj*, *user*)

Paramètres
- **obj** – Object to test for permission
- **user** – core.models.User to test permissions against

Renvoie if user is authorized to edit object

Type renvoyé bool

Example

```
if not can_edit(self.object ,request.user):
    raise PermissionDenied
```

core.views.**can_view**(*obj*, *user*)

Paramètres
- **obj** – Object to test for permission
- **user** – core.models.User to test permissions against

Renvoie if user is authorized to see object

Type renvoyé bool

Example

```
if not can_view(self.object ,request.user):
    raise PermissionDenied
```

Voici un exemple d'utilisation dans un template :

```
{# ... #}
{% if can_edit(club, user) %}
    <a href="{{ url('club:tools', club_id=club.id) }}">{{ club }}</a>
{% endif %}
```

9.2.2 Dans une vue

Généralement, les vérifications de droits dans les templates se limitent au liens à afficher puisqu'il ne faut pas mettre de logique autre que d'affichage à l'intérieur. C'est donc généralement au niveau des vues que cela a lieu.

Notre système s'appuie sur un système de mixin à hériter lors de la création d'une vue basée sur une classe. Ces mixins ne sont compatibles qu'avec les classes récupérant un objet ou une liste d'objet. Dans le cas d'un seul objet, une permission refusée est levée lorsque l'utilisateur n'as pas le droit de visionner la page. Dans le d'une liste d'objet, le mixin filtre les objets non autorisés et si aucun ne l'est l'utilisateur recevra une liste vide d'objet.

Voici un exemple d'utilisation en reprenant l'objet Article crée précédemment :

```
from django.views.generic import CreateView, ListView

from core.views import CanViewMixin, CanCreateMixin

from .models import Article

...

# Il est important de mettre le mixin avant la classe héritée de Django
# L'héritage multiple se fait de droite à gauche et les mixins ont besoin
# d'une classe de base pour fonctionner correctement.
class ArticlesListView(CanViewMixin, ListView):

    model = Article # On base la vue sur le modèle Article
    ...

# Même chose pour une vue de création de l'objet Article
class ArticlesCreateView(CanCreateMixin, CreateView):

    model = Article
```

Le système de permissions de propose plusieurs mixins différents, les voici dans leur intégralité.

class core.views.**CanCreateMixin**(***kwargs*)

This view is made to protect any child view that would create an object, and thus, that can not be protected by any of the following mixin

Raises PermissionDenied

class core.views.**CanEditPropMixin**(***kwargs*)

This view is made to protect any child view that would be showing some properties of an object that are restricted

to only the owner group of the given object. In other word, you can make a view with this view as parent, and it would be retricted to the users that are in the object's owner_group

Raises PermissionDenied

class core.views.**CanEditMixin**(***kwargs*)

This view makes exactly the same thing as its direct parent, but checks the group on the edit_groups field of the object

Raises PermissionDenied

class core.views.**CanViewMixin**(***kwargs*)

This view still makes exactly the same thing as its direct parent, but checks the group on the view_groups field of the object

Raises PermissionDenied

class core.views.**UserIsRootMixin**(***kwargs*)

This view check if the user is root

Raises PermissionDenied

class core.views.**FormerSubscriberMixin**(***kwargs*)

This view check if the user was at least an old subscriber

Raises PermissionDenied

class core.views.**UserIsLoggedMixin**(***kwargs*)

This view check if the user is logged

Raises PermissionDenied

CHAPITRE 10

Le système de groupes

Il existe deux types de groupes sur le site AE. L'un se base sur des groupes enregistrés en base de données pendant le développement, c'est le système de groupes réels. L'autre est plus dynamique et comprend tous les groupes générés pendant l'exécution et l'utilisation du programme. Cela correspond généralement aux groupes liés aux clubs. Ce sont les méta groupes.

10.1 La définition d'un groupe

Comme on peut l'observer, il existe une entrée de groupes dans la base de données. Cette classe implémente à la fois les groupes réels et les méta groupes.

Ce qui différencie ces deux types de groupes ce sont leur utilisation et leur manière d'être générés. La distinction est faite au travers de la propriété *is_meta*.

class core.models.**Group**(**args*, ***kwargs*)

Implement both RealGroups and Meta groups

Groups are sorted by their is_meta property

exception DoesNotExist

exception MultipleObjectsReturned

description

Description of the group

get_absolute_url()

This is needed for black magic powered UpdateView's children

is_meta

If False, this is a RealGroup

10.2 Les groupes réels

Pour simplifier l'utilisation de ces deux types de groupe, il a été crée une classe proxy (c'est à dire qu'elle ne correspond pas à une vraie table en base de donnée) qui encapsule leur utilisation. RealGroup peut être utilisé pour créer des groupes réels dans le code et pour faire une recherche sur ceux-ci (dans le cadre d'une vérification de permissions par exemple).

class core.models.**RealGroup**(**args*, ***kwargs*)

RealGroups are created by the developer. Most of the time they match a number in settings to be easily used for permissions.

exception DoesNotExist

exception MultipleObjectsReturned

objects = <core.models.RealGroupManager object>

Assign a manager in a way that MetaGroup.objects only return groups with is_meta=True

---

Note : N'oubliez pas de créer une variable dans les settings contenant le numéro du groupe pour facilement l'utiliser dans le code plus tard. Ces variables sont du type *SITH_GROUP_GROUPE_NAME_ID*.

10.3 Les méta groupes

Les méta groupes, comme expliqué précédemment, sont utilisés dans les contextes où il est nécessaire de créer un groupe *on runtime*. Les objets *MetaGroup*, bien que dynamiques, doivent tout de même s'enregistrer en base de donnée comme des vrais groupes afin de pouvoir être affectés dans les permissions d'autres objets, comme un forum ou une page de wiki par exemple. C'est principalement utilisé au travers des clubs qui génèrent automatiquement deux groupes à leur création :

— club-bureau : contient tous les membres d'un club **au dessus** du grade défini dans settings.SITH_MAXIMUM_FREE_ROLE.
— club-membres : contient tous les membres d'un club **en dessous** du grade défini dans settings.SITH_MAXIMUM_FREE_ROLE.

class core.models.**MetaGroup**(**args*, ***kwargs*)

MetaGroups are dynamically created groups. Generaly used with clubs where creating a club creates two groups :
— club-SITH_BOARD_SUFFIX
— club-SITH_MEMBER_SUFFIX

exception DoesNotExist

exception MultipleObjectsReturned

objects = <core.models.MetaGroupManager object>

Assign a manager in a way that MetaGroup.objects only return groups with is_meta=False

10.4 La liste des groupes réels

Les groupes réels existant par défaut dans le site sont les suivants :

Groupes gérés automatiquement par le site :
— **Public** -> tous les utilisateurs du site
— **Subscribers** -> tous les cotisants du site
— **Old subscribers** -> tous les anciens cotisants
Groupes gérés par les administrateurs (à appliquer à la main sur un utilisateur) :

— **Root** -> administrateur global du site
— **Accounting admin** -> les administrateurs de la comptabilité
— **Communication admin** -> les administrateurs de la communication
— **Counter admin** -> les administrateurs des comptoirs (foyer et autre)
— **SAS admin** -> les administrateurs du SAS
— **Forum admin** -> les administrateurs du forum
— **Pedagogy admin** -> les administrateurs de la pédagogie (guide des UVs)
— **Banned from buying alcohol** -> les utilisateurs interdits de vente d'alcool (non mineurs)
— **Banned from counters** -> les utilisateurs interdits d'utilisation des comptoirs
— **Banned to subscribe** -> les utilisateurs interdits de cotisation

CHAPITRE 11

Générer l'environnement avec populate

Lors de l'installation du site en local (via la commande *setup*), la commande **populate** est appelée.

Cette commande génère entièrement la base de données de développement. Elle se situe dans *core/management/commands/populate.py*.

Utilisations :

```
./manage.py setup # Génère la base de test
./manage.py setup --prod # Ne génère que le schéma de base et les données strictement␣
↪nécessaires au fonctionnement
```

CHAPITRE 12

Les données générées du site dev

Par défaut, la base de données du site de prod contient des données nécessaires au fonctionnement du site comme les groupes (voir *La liste des groupes réels*), un utilisateur root, les clubs de base et quelques autres instances indispensables. En plus de ces données par défaut, la base de données du site de dev contient des données de test (*fixtures*) pour remplir le site et le rendre exploitable.

Voici les clubs générés pour le site de dev :

- AE
  - Bibo'UT
  - Carte AE
  - Guy'UT
    - Woenzel'UT
  - Troll Penché
- BdF
- Laverie

Voici utilisateurs générés pour le site de dev :

Le mot de passe de tous les utilisateurs est **plop**.

- **root** -> Dans le groupe Root et cotisant
- **skia** -> responsable info AE et cotisant, barmen MDE
- **public** -> utilisateur non cotisant et sans groupe
- **subscriber** -> utilisateur cotisant et sans groupe
- **old_subscriber** -> utilisateur anciennement cotisant et sans groupe
- **counter** -> administrateur comptoir
- **comptable** -> administrateur comptabilité
- **guy** -> utilisateur non cotisant et sans groupe
- **rbatsbak** -> utilisateur non cotisant et sans groupe
- **sli** -> cotisant avec carte étudiante attachée au compte, barmen MDE
- **krophil** -> cotisant avec des plein d'écocups, barmen foyer
- **comunity** -> administrateur communication
- **tutu** -> administrateur pédagogie

CHAPITRE 13

Ajouter des fixtures

Les fixtures sont contenus dans *core/management/commands/populate.py* après la ligne 205 : `if not options["prod"]:`.

Pour ajouter une fixtures, il faut :

- importer la classe à instancier en début de fichier
- créer un objet avec les attributs nécessaires en fin de fichier
- enregistrer l'objet dans la base de données

```python
# Exemple pour ajouter un utilisateur

# Importation de la classe
import core.models import User

# [...]

# Création de l'objet
jesus = User(
    username="jc",
    last_name="Jesus",
    first_name="Christ",
    email="son@god.cloud",
    date_of_birth="2020-24-12",
    is_superuser=False,
    is_staff=True,
)
jesus.set_password("plop")
# Enregistrement dans la base de donnée
jesus.save()
```

CHAPITRE 14

Ajouter une nouvelle cotisation

Il arrive régulièrement que le type de cotisation proposé varie en prix et en durée au cours des années. Le projet étant pensé pour être utilisé par d'autres associations dans la mesure du possible, ces cotisations sont configurables directement dans les paramètres du projet.

14.1 Comprendre la configuration

Pour modifier les cotisations disponnibles, tout se gère dans la configuration avec la variable *SITH_SUBSCRIPTIONS*. Dans cet exemple, nous allons ajouter une nouvelle cotisation d'un mois.

```
from django.utils.translation import gettext_lazy as _

SITH_SUBSCRIPTIONS = {
    # Voici un échantillon de la véritable configuration à l'heure de l'écriture.
    # Celle-ci est donnée à titre d'exemple pour mieux comprendre comment cela fonctionne.
    "un-semestre": {"name": _("One semester"), "price": 15, "duration": 1},
    "deux-semestres": {"name": _("Two semesters"), "price": 28, "duration": 2},
    "cursus-tronc-commun": {
        "name": _("Common core cursus"),
        "price": 45,
        "duration": 4,
    },
    "cursus-branche": {"name": _("Branch cursus"), "price": 45, "duration": 6},
    "cursus-alternant": {"name": _("Alternating cursus"), "price": 30, "duration": 6},
    "membre-honoraire": {"name": _("Honorary member"), "price": 0, "duration": 666},
    "un-jour": {"name": _("One day"), "price": 0, "duration": 0.00555333},

    # On rajoute ici notre cotisation
    # Elle se nomme "Un mois"
    # Coûte 6€
    # Dure 1 mois (on résonne en semestre, ici c'est 1/6 de semestre)
```

(suite sur la page suivante)

(suite de la page précédente)

```
    "un-mois": {"name": _("One month"), "price": 6, "duration": 0.166}
}
```

14.2 Créer une migration

La modification de ce paramètre est étroitement lié à la génération de la base de données. Cette variable est utilisé dans l'objet *Subscription* pour générer les *subscription_type*. Le modifier requiers de générer une migration de basse de données.

```
./manage.py makemigrations subscription
```

Note : N'oubliez pas d'appliquer black sur le fichier de migration généré.

14.3 Rajouter la traduction pour la cotisation

Comme on peut l'observer, la cotisation a besoin d'un nom qui est internationalisé. Il est donc nécessaire de le traduire en français. Pour rajouter notre traduction de *« One month »* il faut se référer à cette partie de la documentation : *Ajouter une traduction*.

CHAPITRE 15

Modifier le weekmail

Le site est capable de générer des mails automatiques composé de l'agrégation d'articles composé par les administrateurs de clubs. Le contenu est inséré dans un template standardisé et contrôlé directement dans le code. Il arrive régulièrement que l'équipe communication souhaite modifier ce template. Que ce soit les couleurs, l'agencement ou encore la bannière ou le footer, voici tout ce qu'il y a à savoir sur le fonctionnement du weekmail en commençant par la classe qui le contrôle.

class com.models.**Weekmail**(**args*, ***kwargs*)

The weekmail class

Variables

- **title** – Title of the weekmail
- **intro** – Introduction of the weekmail
- **joke** – Joke of the week
- **protip** – Tip of the week
- **conclusion** – Conclusion of the weekmail
- **sent** – Track if the weekmail has been sent

exception DoesNotExist

exception MultipleObjectsReturned

get_banner()

Return an absolute link to the banner.

get_footer()

Return an absolute link to the footer.

render_html()

Renders an HTML version of the mail with images and fancy CSS.

render_text()

Renders a pure text version of the mail for readers without HTML support.

send()

Send the weekmail to all users with the receive weekmail option opt-in. Also send the weekmail to the mailing list in settings.SITH_COM_EMAIL.

15.1 Modifier la bannière et le footer

Comme on peut l'observer plus haut, ces éléments sont contrôlés par les méthodes *get_banner* et *get_footer* de la classe *Weekmail*. Les modifier est donc très simple, il suffit de modifier le contenu de la fonction et de rajouter les nouvelles images dans les statics.

Les images sont à ajouter dans dans **core/static/com/img** et sont à nommer selon le type (banner ou footer), le semestre (Automne ou Printemps) et l'année.

Exemple : *weekmail_bannerA18.jpg* pour la bannière de l'automne 2018.

```
# Sélectionner le fichier de bannière pour le weekmail de l'automne 2018

def get_banner(self):
    return "http://" + settings.SITH_URL + static("com/img/weekmail_bannerA18.jpg")
```

Note : Penser à prendre les images au format **jpg** et que les images soient le plus léger possible, c'est bien mieux pour l'utilisateur final.

Note : Pensez à laisser les anciennes images dans le dossier pour que les anciens weekmails ne soient pas affectés par les changements.

15.2 Modifier le template

Comme on peut le voir dans la documentation de la classe, il existe deux templates différents. Un des templates est en texte pur et sert pour le rendu dégradé des lecteurs de mails ne supportant pas le HTML et un autre fait un rendu en HTML.

Ces deux templates sont respectivement accessibles aux emplacements suivants :

— com/templates/com/weekmail_renderer_html.jinja
— com/templates/com/weekmail_renderer_text.jinja

Note : Pour le rendu HTML, pensez à utiliser le CSS et le javascript le plus simple possible pour que le rendu se fasse correctement dans les clients mails qui sont souvent capricieux.

Note : Le CSS est inclus statiquement pour que toute modification ultérieure de celui-ci n'affecte pas les versions précédemment envoyées.

Avertissement : Si vous souhaitez ajouter du contenu, n'oubliez pas de bien inclure ce contenu dans les deux templates.

CHAPITRE 16

Documentation de core

16.1 Classes liées aux utilisateurs

class core.models.**User**(**args*, ***kwargs*)

Defines the base user class, useable in every app

This is almost the same as the auth module AbstractUser since it inherits from it, but some fields are required, and the username is generated automatically with the name of the user (see generate_username()).

Added field : nick_name, date_of_birth Required fields : email, first_name, last_name, date_of_birth

CHAPITRE 17

Syntaxe markdown utilisée dans le site

Note : Si vous faites une mise à jour sur le parseur markdown, il est bon de documenter cette mise à jour dans la page de référence *doc/SYNTAX.md*. Mettre à jour ce fichier vas casser les tests si vous ne mettez pas à jour le fichier *doc/SYNTAX.html* qui lui correspond juste après. Pour mettre à jour ce fichier il suffit d'utiliser la commande

```
./manage.py markdown > doc/SYNTAX.html
```

Avertissement : Le rendu de cette aide est fait via Sphinx, il peut représenter quelques différences avec la réalité. Il est possible de télécharger juste en dessous les versions brutes.

- `Fichier d'aide en markdown`
- `Fichier d'aide rendu en HTML`

CHAPITRE 18

Commandes utiles

18.1 Appliquer le header de licence sur tout le projet

```
for f in $(find . -name "*.py" ! -path "*migration*" ! -path "./env/*" ! -path "./doc/*
↪"); do cat ./doc/header "$f" > /tmp/temp && mv /tmp/temp "$f"; done
```

18.2 Compter le nombre de lignes de code

```
sudo apt install cloc
cloc --exclude-dir=doc,env .
```

CHAPITRE 19

Utiliser direnv

Pour éviter d'avoir à sourcer l'environnement à chaque fois qu'on rentre dans le projet, il est possible d'utiliser l'utilitaire direnv.

```
# Installation de l'utilitaire

# Debian et Ubuntu
sudo apt install direnv
# Mac
brew install direnv


# Installation dans la config
# Si sur bash
echo 'eval "$(direnv hook bash)"' >> ~/.bashrc
# Si sur ZSH
echo 'eval "$(direnv hook zsh)"' >> ~/.zshrc

exit # On redémarre le terminal

# Une fois dans le dossier du projet site AE
direnv allow .
```

Une fois que cette configuration a été appliquée, aller dans le dossier du site applique automatiquement l'environnement virtuel, cela fait beaucoup moins de temps perdu pour tout le monde.

Direnv est un utilitaire très puissant et qui peut s'avérer pratique dans bien des situations, n'hésitez pas à aller vous renseigner plus en détail sur celui-ci.

CHAPITRE 20

Configurer pour la production

20.1 Configurer Sentry

Pour connecter l'application à une instance de sentry (ex : https://sentry.io) il est nécessaire de configurer la variable **SENTRY_DSN** dans le fichier *settings_custom.py*. Cette variable est composée d'un lien complet vers votre projet sentry.

20.2 Récupérer les statiques

Nous utilisons du SCSS dans le projet. En environnement de développement (DEBUG=True), le SCSS est compilé à chaque fois que le fichier est demandé. Pour la production, le projet considère que chacun des fichier est déjà compilé, et, pour ce faire, il est nécessaire d'utiliser les commandes suivantes dans l'ordre :

```
./manage.py collectstatic # Pour récupérer tous les fichiers statiques
./manage.py compilestatic # Pour compiler les fichiers SCSS qu'ils contiennent
```

Note : Le dossier où seront enregistrés ces fichiers statiques peut être changé en modifiant la variable *STATIC_ROOT* dans les paramètres.

CHAPITRE 21

Documentations complémentaires

21.1 Python et Django

- Apprendre Python
- Apprendre Django
- Documentation de Django
- Classy Class-Based Views

21.2 HTML/Jinja/JS/(S)CSS

- Cours sur le javascript
- Cours sur jQuery
- Cours sur le HTML et CSS
- Documentation sur les grilles CSS
- Guide pour le SASS
- Documentation de fontawesome
- Documentation pour Jinja2

21.3 Git

- Cours sur Git
- Livre sur Git

CHAPITRE 22

Documents téléchargeables

— Rapport de la TW de Skia sur la création du site
— Rapport sur la TO de Skia et LoJ
— Manuel du service E-transactions
— Guide de trésorerie

Index