



**SORBONNE
UNIVERSITÉ**

CRÉATEURS DE FUTURS
DEPUIS 1257

Data Gouv

Création d'un site Web avec le framework Django



Les membres du groupe:

- Pacôme TISSERAND
- Adonis-Ahmed HAOUILI
- Sonia MOUSSAOUI
- Adame BENADJAL
- Alexandre LY

Enseignant encadrant:

Monsieur Bertil FOLLIOU

Table des matières

Introduction	6
Qu'est-ce que Django ?	6
Pourquoi est-ce qu'on aurait besoin d'un framework ?	7
Présentation de notre projet	7
Diagramme de GANTT	10
Répartition des tâches pour chaque membre du groupe	11
Comment accéder à notre site ?	12
La Gestion des différents environnements de développement.	12
La gestion spécifique de linux :	13
La gestion spécifique de Mac OS X:	13
La gestion spécifique de Window :	13
La gestion spécifique de AWS:	16
La configuration interne de Django	17
Introduction	17
Mise en place	17
Choix de développement from scratch ou accompagné	17
Choix de base de donnée	18
Le chargement des données du gouvernement	19
Choix et recherche des données	19
Vérification et analyse des base en python	20
Bug d'importation avec le parsing de csv :	21
Importation des données	22
Modélisation de la base de donnée	22
Importation de donnée	23
Bug d'importation de donnée	24
Les bugs de taille de donnée	25
Les bugs de type de donnée	25
Parallélisation de l'importation	25
L'architecture de Django et le développement de l'application	26

Explication sur le fonctionnement de Django	26
Le mécanisme de routage de notre application	28
Les API RESTless	30
Les vues	31
Les vues liés à des forms	31
Création de module de form	31
L'ajout d'un module de captcha	32
Intégration de form	32
La mise en forme des form	33
Rendu du formulaire	34
Rendu final	35
Traitement de gros forms et problématique de fonctionnement d'ORM	35
Bug de gestion de délimiteur dans l'ORM Django	37
wrapper de vue	40
Moteur de templating	41
Tests Croisés	42
Groupe 1 – Jeu narratif avec pour but de rester en vie	42
Remarques :	42
Bugs / Problèmes :	42
Améliorations possibles :	43
Groupe 2 - Jeu Graphique avec génération procédurales avec un moteur de jeu.	44
Remarques :	44
Bugs / Problèmes :	44
Améliorations possibles :	45

Introduction

Dans le cadre de l'UE de projet encadré nous avons choisi de développer un site Web complet en utilisant un nouveau langage pour la plupart d'entre nous qui est Python et plus précisément à l'aide du Framework Django.

Qu'est-ce que Django ?

Django est un framework web libre et open-source écrit en Python. Un framework web est un ensemble de composants qui vous aide à développer des sites web plus rapidement et plus facilement.

Lorsque vous créez un site web, vous avez souvent besoin de la même chose : une manière de gérer l'authentification de vos utilisateurs (créer un compte, se connecter, se déconnecter), une partie dédiée à la gestion de votre site, des formulaires, une manière de mettre en ligne des fichiers, etc.

La bonne nouvelle, c'est que d'autres gens se sont aussi rendus compte de ce problème et ont décidé de s'allier avec des développeurs·ses pour le résoudre. Ensemble, ces personnes ont créé différents frameworks, dont Django, pour fournir un set de composants de base qui peuvent être utilisés lors de la création d'un site web.

Les frameworks existent pour vous éviter de réinventer la roue à chaque fois. Ils vous aident aussi à alléger la charge de travail liée à la création d'un site web.

Pourquoi est-ce qu'on aurait besoin d'un framework ?

Pour comprendre ce à quoi peut bien servir Django, nous avons besoin de nous intéresser aux multiples rôles des serveurs. Par exemple, la première chose qu'a besoin de savoir un serveur, c'est que vous aimeriez qu'il vous affiche une page web.

Imaginez une boîte aux lettres (un port) dont l'arrivée de lettres (une requête) serait surveillée. C'est le travail qu'effectue le serveur. Le serveur web lit la lettre qu'il a reçue et en réponse, retourne une page web. Généralement, lorsque vous voulez envoyer quelque chose, vous avez besoin de contenu. Django est quelque chose qui va vous aider à créer ce contenu.

Présentation de notre projet

Pour cette UE de projet encadré, notre groupe était composé de cinq personnes. Tisserant Pacome, Haouili Adonis, Sonia Moussaoui, Ly Alexandre et Benadjal Adame.

Afin de choisir le sujet de notre projet, chacun d'entre nous a proposé un projet concret allant d'une application mobile Android à l'exécutable sur Windows et Linux ainsi qu'un site web en utilisant le Framework Django.

Après une longue réflexion entre nous et en exposant chacun à tour de rôle nos arguments, nous avons opté pour le projet Django qui semblait être le projet le plus intéressant pour nous car il nous permet d'apprendre une nouvelle technologie ainsi que de consolider des connaissances que l'on a pu apprendre au premier semestre de notre L2 ATIC durant l'UE développement web et donc d'aller beaucoup plus en profondeur dans le monde du web.

Concrètement, le but de notre projet a été d'employer des données fournies au grand public du site data.gouv.fr et plus précisément des données liées à l'enseignement.

Nous avons donc télécharger quatre fichiers CSV (comma-separated values) contenant ces données conséquentes (des fichiers qui allaient jusqu'à 2.7GO) qui vont être par la suite le coeur de notre projet car nous allons effectuer des traitements, des affichages et des mises en pages de ces derniers.

En plus de Django, nous utiliserons Bootstrap pour des templates beaucoup plus propre. Pour la base de donnée nous avons choisi MySQL pour stocker toute cette masse de donnée.

Notre projet c'est fait évidemment en plusieurs étapes que l'on va détailler par la suite en commençant bien sûr par la conception et la réflexion, fixer les différentes tâches à effectuer et en mettant en place un diagramme de GANTT pour s'organiser entre nous, se répartir les tâches de manière équitable afin de mener à bien ce projet.

Les principales tâches fixées après avoir nourri une réflexion sur la conception de ce site web sont :

→ **Configuration de Django:**

- ◆ Notre groupe possède 2 utilisateurs Linux, un sur Windows et deux utilisateurs sur Mac.
- ◆ Il a fallu mettre en place l'environnement de développement sur chacune de nos machines pour que pouvoir travailler efficacement.

→ **Intégration bases de données:**

- ◆ Nous avons commencé par télécharger les fichiers CSV (comma-separated values) sur data.gouv.fr et il fallait donc intégrer toutes les informations qu'ils contenaient dans notre base de donnée.
 - Un premier fichier qui contient l'effectif régional d'étudiants inscrits.
 - Un deuxième fichier qui contient la liste des écoles doctorales.

-
- Un troisième fichier qui contient la liste des bénéficiaires d'une prime d'excellence scientifique.
 - Un dernier fichier qui contient les effectifs totaux d'étudiants par université pour toute la France.
- ◆ Mise en place de scripts Python qui vont effectuer l'importation des données depuis ces fichiers CSV.
 - ◆ Intégrer les données dans nos bases MySQL via l'ORM (object relational mapping) Django qui contient les modèles adéquats.
 - ◆ Gérer les différents bugs rencontrés lors de l'importation.
 - ◆ Créer un script qui va effectuer les 4 importations d'un coup en utilisant des fonctions de parallélisation pour répartir et optimiser l'importation.

→ **Création de templates:**

- ◆ Mise en page de la page d'accueil.
- ◆ Page de recherche par établissement.
- ◆ Des templates pour chacune de nos tables.

Diagramme de GANTT

Datagouv			13 févr. 2018
Tâches			2
Nom	Date de début	Date de fin	
Réflexion & conception	22/01/18	20/02/18	
Configuration Django.(Gestion des déps).	23/01/18	06/02/18	
<i>Tâche réalisé par Pacôme Tisserand sur 2 Mac, 2 linux, et Windows.</i>			
Rédaction du Rapport.	23/01/18	02/04/18	
création template	06/02/18	30/03/18	
<i>Création de template avec Jinja2 via le moteur de template par défaut de Django.</i>			
<i>Sera également utilisé la library CSS bootstrap.</i>			
<i>Et éventuellement des library JS de rendu de graph.</i>			
intégration bdd	06/02/18	19/03/18	
<i>Intégration de base de donnée se fait sous la forme de script python qui importent des CSV pour les mettre dans la base de donnée MySQL via l'ORM django qui contient les modèles adéquats.</i>			
page de recherche par établissement	06/02/18	06/03/18	
aff établissement - stat élève	07/03/18	30/03/18	
liaison établissement prime élève	13/02/18	30/03/18	
amélioration menu recherche	12/02/18	30/03/18	
ajout carte	01/03/18	20/03/18	
Préparation soutenance	19/03/18	30/03/18	
<i>Préparation à l'oral, Réalisation du PowerPoint de la présentation. & Répartition du temps.</i>			
Tests croisés	19/03/18	19/03/18	
<i>Cette tâche regroupe tout les membres du groupe ATIC.</i>			
Tests	19/03/18	30/03/18	
<i>Vérification des fonctionnalités du site, et dernières mises à jour.</i>			
Rédaction du Rapport final et remise.	26/03/18	26/03/18	

Répartition des tâches pour chaque membre du groupe

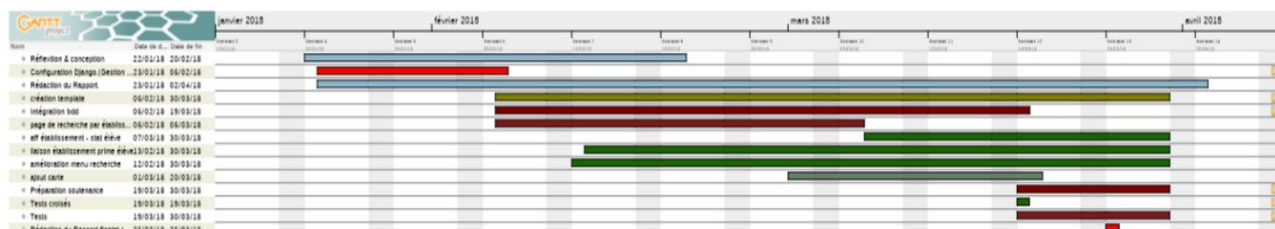


Diagramme des Ressources

5



Comment accéder à notre site ?

Notre site est accessible par deux moyens différents :

- **Directement sur le Web** : Nous avons hébergé notre site sur Amazon Web Services (AWS) accessible directement via le lien suivant :

<http://ec2-18-219-255-214.us-east-2.compute.amazonaws.com/>

A noter que cette version n'est pas forcément la plus à jour, se référer à la seconde méthode pour avoir la toute dernière version.

- **En installation local** : il existe un git que nous avons créés pour construire le projet, avec les instructions d'installation présentes directement dans le *README.md*.

Lien : <https://github.com/darcosion/datagouv.git>

La Gestion des différents environnements de développement.

Pour ce projet, comme dis en introduction, nous avons des machines très différentes les unes des autres, à savoir deux Ubuntu (16.04 LTS & 12.04 LTS), deux Mac OS X (Sierra 10.12.6), et enfin un Windows Seven.

Suivant les OS et leurs version, nous avons anticipés que ça n'allait pas être la même version de Django qui serait lancé.

Par conséquent, nous avons utilisé un outil professionnel de gestion de versioning des dépendances nommé [pyenv](#), il permet de définir grâce à un fichier *requirement.txt* et un *selfcheck.txt* un environnement unifié pour l'ensemble des personnes du groupe qui par conséquent utiliseront forcément la même version de chaque package, évitant ainsi d'avoir des bugs dus à des différences de code en fonction des versions.

La gestion spécifique de linux :

Sous linux, nous avons souhaités prioriser l'installation de la base de donnée car c'était le système sur lequel nous avions plus de stabilité au niveau de MySQL et plus de possibilité de debug.

Au niveau de l'installation, c'est l'installation classique qui est décrite dans le README.md.

La gestion spécifique de Mac OS X:

Sur Mac OS X, une particularité de complexification a été ajoutée, le gestionnaire de package APT que nous utilisons n'était pas présent, nous avons donc dû nous rabattre sur brew, qui est un installateur/compilateur de source code très similaire à composer pour le PHP mais permettant de résoudre toutes les dépendances sur Mac.

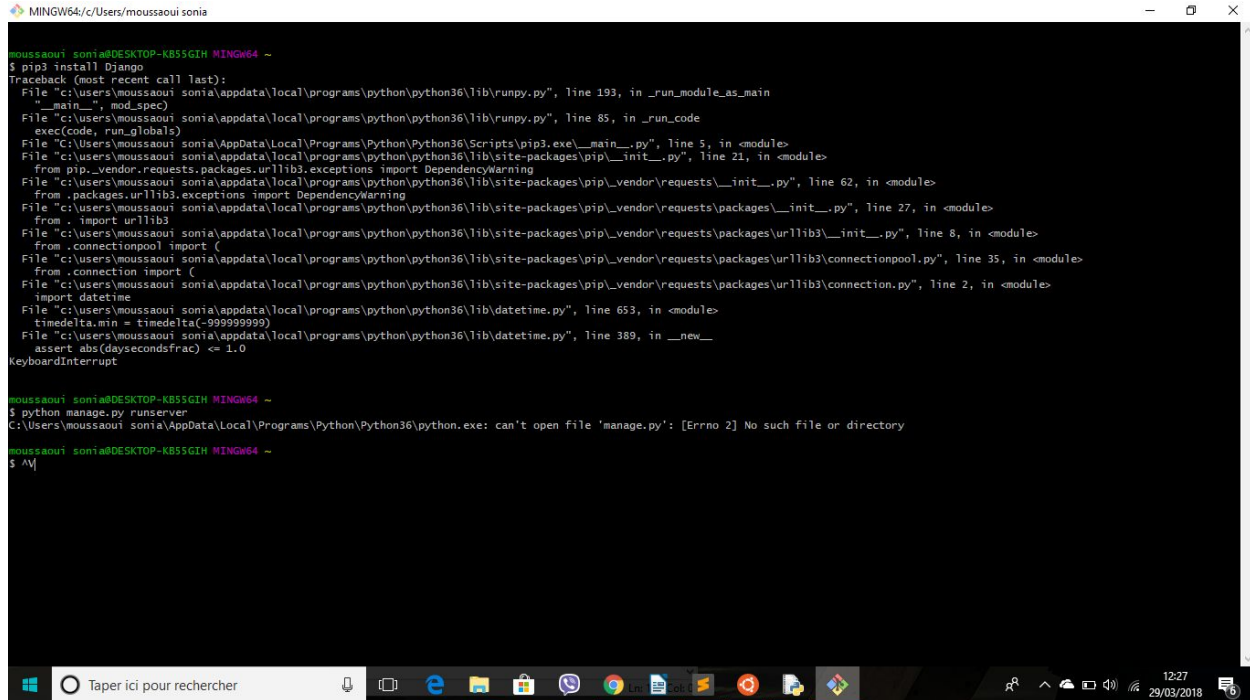
La gestion spécifique de Window :

Sous windows, il n'existait pas de solution toute faite pour notre problématique, du fait que python était historiquement développé sous linux, l'intégration de Django dans des Ubuntu et des Mac ayant un noyau linux était plutôt trivial comparé à l'installation sous windows qui possède un noyau... windows.

Donc sous windows, nous avons d'abord optés pour une utilisation pure de Django avec python, puisque python est supporté et installable sous windows, et il intègre pip et easy_install pour gérer ses dépendances si on le lui spécifie dans son installateur.

Nous avons commencés par faire cela, mais s'est posé le soucis de la manipulation de python hors console IDLE car Django nécessite de se lancer à l'invite de commande.

En parallèle, nous avons donc installés git pour windows, avec un logiciel nommé git-scm qui nous donnait une invite de commande git.



```
MINGW64/c:/Users/moussaoui sonia
moussaoui sonia@DESKTOP-K855GIH MINGW64 ~
$ pip3 install Django
Traceback (most recent call last):
  File "c:/users/moussaoui sonia/appdata/local/programs/python/python36/lib/runpy.py", line 193, in _run_module_as_main
    "__main__", mod_spec)
  File "c:/users/moussaoui sonia/appdata/local/programs/python/python36/lib/runpy.py", line 85, in _run_code
    exec(code, run_globals)
  File "C:/Users/moussaoui sonia/AppData/Local/Programs/Python/Python36/Scripts/pip3.exe__main__.py", line 5, in <module>
  File "c:/users/moussaoui sonia/appdata/local/programs/python/python36/lib/site-packages/pip__init__.py", line 21, in <module>
    from pip._vendor.requests.packages.urllib3.exceptions import DependencyWarning
  File "c:/users/moussaoui sonia/appdata/local/programs/python/python36/lib/site-packages/pip/_vendor/requests/__init__.py", line 62, in <module>
    from .packages.urllib3.exceptions import DependencyWarning
  File "c:/users/moussaoui sonia/appdata/local/programs/python/python36/lib/site-packages/pip/_vendor/requests/packages/urllib3/__init__.py", line 27, in <module>
    from . import urllib3
  File "c:/users/moussaoui sonia/appdata/local/programs/python/python36/lib/site-packages/pip/_vendor/requests/packages/urllib3/__init__.py", line 8, in <module>
    from .connectionpool import (
  File "c:/users/moussaoui sonia/appdata/local/programs/python/python36/lib/site-packages/pip/_vendor/requests/packages/urllib3/connectionpool.py", line 35, in <module>
    from .connection import (
  File "c:/users/moussaoui sonia/appdata/local/programs/python/python36/lib/site-packages/pip/_vendor/requests/packages/urllib3/connection.py", line 2, in <module>
    import datetime
  File "c:/users/moussaoui sonia/appdata/local/programs/python/python36/lib/datetime.py", line 653, in <module>
    timedelta.min = timedelta(-999999999)
  File "c:/users/moussaoui sonia/appdata/local/programs/python/python36/lib/datetime.py", line 389, in __new__
    assert abs(daysecondsfrac) <= 1.0
KeyboardInterrupt

moussaoui sonia@DESKTOP-K855GIH MINGW64 ~
$ python manage.py runserver
C:/Users/moussaoui sonia/AppData/Local/Programs/Python/Python36/python.exe: can't open file 'manage.py': [Errno 2] No such file or directory

moussaoui sonia@DESKTOP-K855GIH MINGW64 ~
$ ^M
```

Mais on s'est aperçu que git-scm ne gérait pas bien du tout sa mémoire sous windows, et que nous avions [des bugs](#) très étranges avec, accompagnés d'une lenteur très inhabituelle. Nous n'avons pas souhaités investiguer car nous avions une solution de repli qui consistait à utiliser une VM linux sous windows d'un genre un peu particulier.

En effet, microsoft propose dans le microsoft Store une [VM ubuntu spécialement conçue pour windows](#) utilisant les fonctions du système de virtualisation Hyper-V dont les performances nous intéressaient.

Nous l'avons donc installés et au début, la VM ne présentait aucun soucis, elle était identique en utilisation à une de nos machines sous Ubuntu.

Mais nous avons rencontrés deux soucis très bloquants avec, outre les prétendu bugs de gestion de passerelle réseau indiqués par *Felix Forlini* qui ne nous sont pas apparus.

Le premier bug consistait au fait que des modifications apportés au code de Django accessible en dehors de la VM n'étaient pas pris en compte dans la VM, ce qui signifiait qu'à chaque fois qu'on faisait une modification, nous étions obligés de relancer la VM pour qu'elle prenne en compte la modification.

Ce bug est du à une propriété du système de gestion de fichier NTFS qui se nomme les NTFS extended file attributs qui lient littéralement le contenu d'un gros fichier NTFS contenant l'arborescence de la VM avec l'ext3/ext4 virtuel crée par Hyper-V lorsque l'on lance la VM Ubuntu.

Et comme nous utilisons git dans la VM, nous avons souvent un bug phénoménalement bizarre lorsque nous récupérons sur le réseau une version plus récente du projet.

Ce qui se passait en réalité, c'était que tout d'abord, nous éditions dans windows un certain nombre de fichier de développement de Django qui se mettaient donc à jour dans le NTFS.

Puis, dans la VM, afin d'ajouter nos modifications, nous faisons un git pull qui allaient également modifier ces fichiers dans l'ext3 d'Ubuntu.

Ensuite, nous fermions la VM et là, **la VM tentais de mettre à jour les fichiers sur le NTFS, mais ne pouvait pas toujours car les fichiers mis à jour en raison de nous ajouts sur le NTFS entraient en collision, et du coup, windows résolvait cela silencieusement en décidant que la version la plus à jour était celle du NTFS et il supprimait purement et simplement la version de la VM.**

Le pire étant que vu que nous ne modifions pas le .git lui même, git de son coté, affirmait que les modifications avaient été chargés et que le repo était à jour alors qu'en réalité, une partie des ajouts distants avaient été détruit par windows.

Ce bug que l'on pourrait qualifier de fonctionnalité a été très bien documenté par Microsoft ici :

<https://blogs.msdn.microsoft.com/commandline/2016/11/17/do-not-change-linux-files-using-windows-apps-and-tools/>

Leur recommandation consiste à hoster les fichiers sous windows via `/mnt/file_in_windows`, ce que nous avons fait, malheureusement trop tard.

Entre temps, nous avons effectués une autre mise en place en utilisant mon serveur distant AWS et un ssh + VNC, mais Amazon et Eduroam ont des règle de sécurité très contradictoires, par conséquence, cette voie n'a pas été plus approfondie en raison des blocages induits.

Pour préciser, nous avons mis en place un serveur VNC sur la machine AWS mais il utilisait le port 5001, bloqué par Eduroam, et il n'était pas possible de mettre un port primaire (entre 1 et 1000) sur AWS en raison de ses règles de sécurité.

La gestion spécifique de AWS:

Pour diffuser notre site sur le web et faciliter les test croisés, nous avons plus tard mis en place une VM AWS.

L'installation du projet DataGouv s'est déroulé de la même façon car la VM était sous Ubuntu.

Cependant, AWS a de solide règles de sécurité, et il a fallu jouer avec son firewall pour ouvrir les ports adéquats pour le site (port 80 et 443).

```
root@DESKTOP-KB55GIH:/mnt/c/datagouv
root@DESKTOP-KB55GIH:~# cd
root@DESKTOP-KB55GIH:~# cd home/soniahocine/datagouv/
-bash: cd: home/soniahocine/datagouv/: No such file or directory
root@DESKTOP-KB55GIH:~# cd ..
root@DESKTOP-KB55GIH:~# cd home/soniahocine/datagouv/
root@DESKTOP-KB55GIH:/home/soniahocine/datagouv# source bin/activate
(datagouv) root@DESKTOP-KB55GIH:/home/soniahocine/datagouv# cd datagouv/
(datagouv) root@DESKTOP-KB55GIH:/home/soniahocine/datagouv/datagouv# service mysql start
* Starting MySQL database server mysqld
No directory, logging in with HOME=/
^C
(datagouv) root@DESKTOP-KB55GIH:/home/soniahocine/datagouv/datagouv# python3 manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).
March 28, 2018 - 16:52:29
Django version 2.0.1, using settings 'datagouv.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
^C
(datagouv) root@DESKTOP-KB55GIH:/home/soniahocine/datagouv/datagouv# git status
On branch master
Your branch and 'origin/master' have diverged,
and have 1 and 5 different commits each, respectively.
(use "git pull" to merge the remote branch into yours)
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   home/views.py
        modified:   template/recherche.html

no changes added to commit (use "git add" and/or "git commit -a")
(datagouv) root@DESKTOP-KB55GIH:/home/soniahocine/datagouv/datagouv# git pull
remote: Counting objects: 137, done.
remote: Compressing objects: 100% (64/64), done.
remote: Total 137 (delta 110), reused 99 (delta 72), pack-reused 0
Receiving objects: 100% (137/137), 23.67 KiB | 0 bytes/s, done.
Resolving deltas: 100% (110/110), completed with 11 local objects.
From http://github.com/darcosion/datagouv
  3d11709..78551e4  master    -> origin/master
error: Your local changes to the following files would be overwritten by merge:
       datagouv/home/views.py
       datagouv/template/recherche.html
Please, commit your changes or stash them before you can merge.
Aborting
(datagouv) root@DESKTOP-KB55GIH:/home/soniahocine/datagouv/datagouv# git add home/forms.py
(datagouv) root@DESKTOP-KB55GIH:/home/soniahocine/datagouv/datagouv# git commit -m "l'ajout des formulaire 28 "
On branch master
Your branch and 'origin/master' have diverged,
and have 1 and 26 different commits each, respectively.
(use "git pull" to merge the remote branch into yours)
```

La configuration interne de Django

Introduction

Django est un framework très modulaire, sur lequel beaucoup de développeur et de projets se sont basés, au point que certains projets populaire furent intégrés au framework de base afin d'en faciliter l'utilisation, soit sous la forme de MiddleWare, soit sous la forme d'application, comme la maintenant très célèbre application PanelAdmin qui permet d'administrer la base de donnée de Django directement depuis une interface web sur le site.

Mise en place

Choix de développement from scratch ou accompagné

Dans notre cas, nous nous sommes d'abord interrogés sur la pertinence de reprendre des applications existantes contenant des fonctionnalités qui nous intéressent, comme [django-csvimport](#), [plotly](#), ou encore [Django-graphos](#).

Puis, nous avons décidé de reprendre tout de la base, estimant plus intéressant en terme d'expérience d'apprendre à créer une application Django de A à Z.

Nous avons donc créé datagouv de cette façon :

```
mkdir datagouv & cd datagouv
```

```
pyenv -m * & source bin/activate #mise en place du pyenv
```

```
*installation des dépendances de Django*
```

```
django startproject datagouv
```

```
cd datagouv
```

```
python3 manage.py makeapp home #création de l'application que l'on souhaite développer.
```

Choix de base de donnée

Django a relativement bien documenté 4 type de base de donnée, à savoir les base de données issues de PostgreSQL, MySQL, SQLite et Oracle.

De base, si aucune base de donnée n'est spécifié dans le fichier, Django va créer automatiquement une base de donnée SQLite.

Mais nous avons exclue cette techno de SGDB pour une raison simple : nous souhaitons traiter de gros volumes de donnée, et SQLite, comme son nom l'indique n'est pas adapté à de la base de donnée très volumétrique mais plutôt à de la petite base de donnée.

Nous avons donc expérimentés plusieurs technologies et regardé leurs perfs sur le net.

De base, nous avons exclus les technologies NoSQL car nous estimons que l'ORM Django pouvait pallier assez simplement aux problématiques de manque de maniabilité des base de donnée relationnelles grâce à son mécanisme de migration automatisés.

Après avoir effectués quelques tests avec MariaDB et postgresQL, nous nous sommes finalement tournés vers MySQL en raison de sa stabilité sur nos différentes machines.

Et nous avons donc modifiés le fichier *datagouv/datagouv/datagouv/settings.py* en conséquence pour y spécifier notre base de donnée :

```
76 # Database
77 # https://docs.djangoproject.com/en/2.0/ref/settings/#databases
78
79 DATABASES = {
80     'default' : {
81         'NAME' : "datagouv",
82         'ENGINE' : 'django.db.backends.mysql',
83         'USER' : "root",
84         'PASSWORD' : "motdepas",
85     }
86 }
87
88
```

Le chargement des données du gouvernement

Choix et recherche des données

Concernant les données du gouvernement, nous visions spécifiquement l'affichage de donnée du ministère de l'enseignement supérieur et de la recherche, et nous avons par la suite restreint notre choix sur 4 grosses base de donnée qui nous semblaient intéressantes.

La première est une base de donnée des effectifs régionaux d'étudiants inscrits dans les établissements et les formations de l'enseignement supérieur sur deux ans :

<https://www.data.gouv.fr/fr/datasets/effectifs-d-etudiants-inscrits-dans-les-etablissements-et-les-formations-de-l-enseignement-superieur/>

La seconde est une liste des écoles doctorales accréditées en France :

<https://www.data.gouv.fr/fr/datasets/liste-des-ecoles-doctorales-accreditees/>

La troisième est une liste des bénéficiaires de la prime d'excellence scientifique durant une période donnée :

<https://www.data.gouv.fr/fr/datasets/les-beneficiaires-de-la-prime-d-excellence-scientifique-mesr/>

Enfin, la dernière est une liste des effectifs étudiants inscrits dans tous les organismes sous la tutelle du ministère de l'enseignement supérieur de 2006 à maintenant :

<https://www.data.gouv.fr/fr/datasets/effectifs-d-etudiants-inscrits-dans-les-etablissements-publics-sous-tutelle-du-ministere-en-charge-d/>

Cette base de données était particulièrement problématique car elle avait environ 2 700 000 entrées et 78 colonnes.

Vérification et analyse des bases en python

Via le module CSV de python, nous avons pu construire un script d'analyse des bases nommé `test_csv.py` qui s'utilise de cette façon :

```
$ python3 test_csv.py --help
```

```
usage: test_csv.py [-h] [-f F] [-c]
```

Ajout de fichier à tester

optional arguments:

`-h, --help` show this help message and exit

`-f F` Envoie de fichiers à tester

-c *Afficher uniquement les colonnes*

Ce fichier utilise le module argparse pour traiter des argument proprement à l'invite de commande et propose deux fonctionnalités, l'affichage du nombre de colonne avec leur noms, et le nombre total de ligne du fichier.

```
(datagouv) @:~/Documents/django-site/datagouv$ python3 test_csv.py -c -f fr-esr-pes-pedr-beneficiaires
il y a 19 colonnes
colonnes :
annee
code_sexe
sexe
code_secteur_cnu
secteur_disciplinaire
code_groupe_corps
groupe_de_corps
code_uai
etablissement
code_academie
academie
code_region
region
code_pres
pres
code_idex
idex
beneficiaires
geo_localisation
```

A noter que sur les gros fichiers comme la dernière base énoncé, il est recommandé en analyse préliminaire d'utiliser -c pour éviter le script tourne 3 heures en comptant toute les entrées si le CSV est très gros.

Bug d'importation avec le parsing de csv :

Le système d'importation nous a posé quelques soucis.

Pour commencer, l'importation utilise des streams afin de ne pas surcharger la mémoire de la machine avec les données, nous avons donc utilisés un premier reader et en second, nous utilisons la fonction split() pour séparer dans un tableau les différents éléments d'une entrée CSV. Mais la base de donnée pouvait contenir des listes imbriqués, de cette façon : donnée1;(donnéeex;donnéey;donnéez);donnée3.

Et la fonction split ne prenait pas en compte les parenthèses malheureusement, mais la fonction de stream CSV permettait de résoudre ce soucis, nous avons donc opérés de la sorte :

```
20 with open(args.f[0]) as csvfile:
21     # le delimiter permet de définir les entrée
22     # le quotechar sépare les colonnes
23     reader = csv.reader(csvfile, delimiter='\\n')
24     if(args.c == True):
25         for row in reader:
26             trow = csv.reader(row, delimiter=';')
27             print("il y a {} colonnes".format(len(list(trow)[0])))
28             trow = csv.reader(row, delimiter=';')
29             print("colonnes : \\n")
30             for t in list(trow)[0]:
31                 print(t)
32             break
33
```

Ensuite, une autre particularité du parseur nous avait posé soucis, et elle avait à voir avec le typage faible de python.

A un certain moment, nous avons du caster un stream en array python, via la fonction list(), ce qui a le malheureux effet de charger tout le contenu du stream dans la mémoire vive.

Sur les petits CSV, ça ne posais pas trop de soucis, mais sur la base de 3 millions d'entrée...

Importation des données

Modélisation de la base de donnée

Pour importer les données, nous avons créés une mécanique assez complexe mais très efficace.

En premier lieu, dans *datagouv/datagouv/home/models.py*, nous avons créés des modèles correspondant chacun à un des fichiers csv que l'on importait.

Le modèle ainsi créé va se générer dans la base de donnée SQL via le mécanisme de migration de Django avec les commandes :

```
python3 manage.py makemigrations
```

python3 manage.py migrate

```
55 class EffectifRegional(models.Model):
56     rentree=models.PositiveIntegerField(null=True)
57     rentree_universitaire=models.CharField(max_length=100,null=True) #Il y a du un
58     niveau_geographique=models.CharField(max_length=100,null=True)
59     geo_nom=models.CharField(max_length=100, null=True)
60     regroupement=models.CharField(max_length=100, null=True)
61     rgp_ formations_ou_etablissements=models.CharField(max_length=200, null=True)
62     secteur=models.CharField(max_length=20, null=True)
63     secteur_de_l_etablissement=models.CharField(max_length=100, null=True)
64     sexe=models.PositiveIntegerField(null=True)
65     sexe_de_l_etudiant=models.CharField(max_length=20, null=True)
66     effectif=models.FloatField(null=True)
67     a_des_effectifs_dut=models.CharField(max_length=20, null=True)
68     effectif_dut=models.CharField(max_length=20, null=True) #vide à revoir
69     a_des_effectifs_ing=models.CharField(max_length=20, null=True)
70     effectif_ing=models.CharField(max_length=20, null=True) #vide à revoir
71     diffusable=models.CharField(max_length=20,null=True)
72     donnees_diffusables=models.CharField(max_length=120,null=True)
73     secret=models.CharField(max_length=20,null=True)
74     donnees_soumises_au_secret_stat=models.CharField(max_length=120,null=True)
75     niveau_geo=models.CharField(max_length=30,null=True)
76     geo_id=models.CharField(max_length=10,null=True)
77     reg_id=models.CharField(max_length=40,null=True)
78     aca_id=models.CharField(max_length=20,null=True)
79     dep_id=models.CharField(max_length=20,null=True)
80     uucr_i=models.CharField(max_length=20,null=True)
81
82
```

De cette façon, nous disposons d'une base de donnée vide qui pouvait être modifiée à la volée si un élément du modèle ne nous satisfaisait pas.

Importation de donnée

Ensuite, nous avons créés des scripts d'importation de CSV qui cette fois-ci allait ajouter les données dans la base de donnée via l'ORM de Django.


```
11 with open('../fr-esr-pes-pedr-beneficiaires.csv') as csvfile:
12     # le delimiter permet de définir les entrée
13     # le quotechar sépare les colonnes
14     reader = csv.reader(csvfile, delimiter='\\n')
15     for row in list(reader)[1:]:
16         #print(compt)
17         trow = list(csv.reader(row, delimiter=';'))
18         trow[0][8]=trow[0][8].replace('\\x92','"')
19         if(trow[0][15] == ''):
20             trow[0][15] = None
21         if(trow[0][5] == ''):
22             trow[0][5] = None
23         t = BenefPrimeExcellence(compt
24             ,trow[0][0]
25             ,trow[0][1]
26             ,trow[0][2]
27             ,trow[0][3]
28             ,trow[0][4]
29             ,trow[0][5]
30             ,trow[0][6]
31             ,trow[0][7]
32             ,trow[0][8]
33             ,trow[0][9]
34             ,trow[0][10]
35             ,trow[0][11]
36             ,trow[0][12]
37             ,trow[0][13]
38             ,trow[0][14]
39             ,trow[0][15]
40             ,trow[0][16]
41             ,trow[0][17]
42             ,trow[0][18]
43         )
44         try:
45             t.save()
```

De cette façon, à chaque fois qu'un élément ne rentrais pas dans les les colonnes que nous avons définis pour la base de donnée, il était possible de régler le soucis de deux façons :

- Soit en modifiant le contenu dans le script d'import pour qu'il soit adapté à la base de donnée.
- Soit en modifiant les attributs des modèles dans la base de donnée pour qu'il soit plus adaptés au contenu du CSV.

Bug d'importation de donnée

Suivant les bugs que nous avons rencontrés, nous les avons résolus de plusieurs manière différentes.

Les bugs de taille de donnée

Sans doutes les plus triviales, certaines données texte dans le csv dépassent la taille limite attribués dans la base de donnée, il suffisait simplement d'augmenter la taille d'un attribut de modèle pour régler le soucis, de cette façon :

```
<<<libelle = models.CharField(max_length=100)
```

```
>>>libelle = models.CharField(max_length=200)
```

On a eu ce cas de figure dans pleins de situations différentes, avec des gens au nom de famille particulièrement long, du genre "VON BUELTZINGSLOEWEN", ou bien "CHAMPEIL-DESPLATS, THULLIER".

Fait intéressant, on a aussi eu des surprises sur des données qu'on pensait figés, comme la civilité sensée être "madame", "monsieur" ou "mademoiselle", ou on a retrouvés par exemple "Madame,Madame" ou "Madame,Monsieur".

Le genre également, où on retrouvais "M" pour masculin, "F" pour féminin et très éloquemment "non définit".

Les bugs de type de donnée

Nous avons constatés également des bugs au niveau des types de donnée, le plus trivial était les champs où nous attentions des nombres et qui étaient remplacés par "None" lorsqu'ils étaient vide. Là dessus nous fûmes forcés de les modifier dans le script d'importation.

Mais également au niveau du département où on s'attendait logiquement à une suite de numéro tels que 75000 pour Paris île-de-France. Et où nous avons eu la surprise de constater que la corse possédait les numéros 2A000 pour la haute-corse et 2B000 pour la basse-corse.

Parallélisation de l'importation

Pour rendre la plus simple possible l'importation de donnée, nous avons fait le choix de paralléliser le chargement de donnée dans la mesure où nous savions que les données sont séparés proprement dans différents modèles sans relations.

Pour cela, nous avons utilisés un module python nommé "multiprocessing" qui est très générique, possédant une fonction permettant de créer des pools de calcule permettant de répartir la charge de travail de manière transparente pour l'utilisateur.

```
1 from multiprocessing.dummy import Pool as ThreadPool
2 import importlib
3
4 modules=['home.conv_beneficiaire', 'home.conv_ecole_doctorante',
5      |'home.conv_eff_etudiant', 'home.effectif_regional']
6
7
8 pool = ThreadPool(4)
9
10 results = pool.map(importlib.import_module, modules)
11
12 pool.close()
13 pool.join()
```

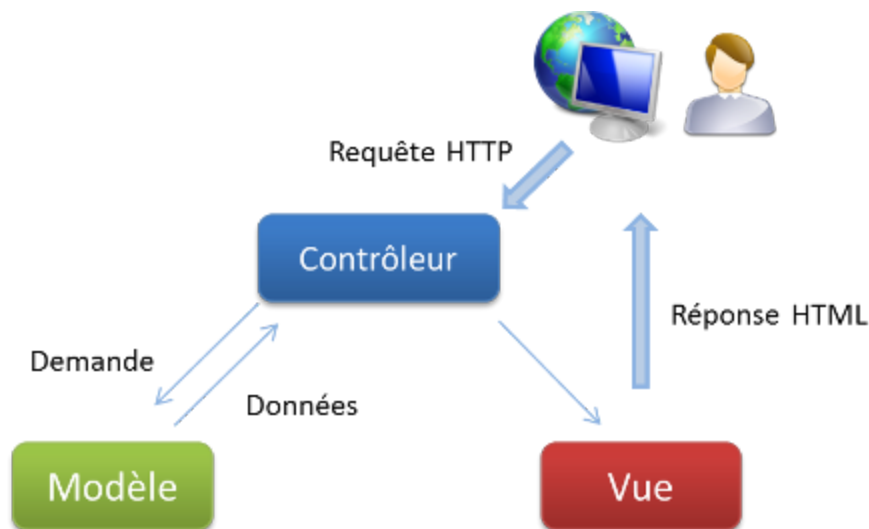
Le code est extrêmement trivial. Ce paralléliseur se contente d'importer les 4 scripts d'importation en parallèle et la fonction map va forcer l'utilisation de 4 processus pythons différents pour l'importation, ce qui permet au système de les répartir proprement dans l'optique de parallélisation qui était voulue.

L'architecture de Django et le développement de l'application

Explication sur le fonctionnement de Django

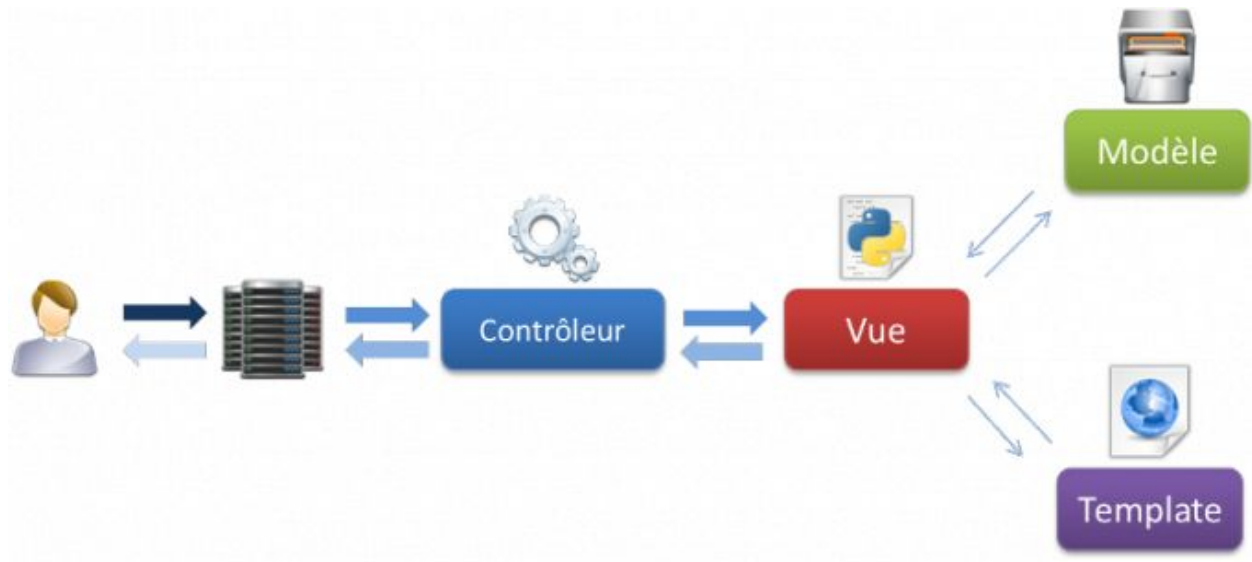
Django fonctionne avec une architecture MVT orienté objet.

C'est un dérivé d'un autre modèle d'interface utilisateur nommé le modèle MVC (Modèle Vue Contrôleur), que l'on représente de cette façon :



Les modèles correspondent aux données dans la base de donnée, la vue correspond au rendu visuel que va faire le framework et le contrôleur administre toute la phase de traitement des requêtes utilisateur.

Le modèle MVT (Modèle Vue Template) quand à lui fonctionne de cette façon :



Le contrôleur est un mécanisme de routage d'url présent dans de multiples fichiers *urls.py*, les vues sont des fonctions de traitement de donnée appelant les modèles et envoyant les données aux templates qui eux vont s'occuper de faire le rendu final pour l'utilisateur via un autre langage nommé Jinja2.

Le mécanisme de routage de notre application

Le fichier `datagouv/datagouv/home/urls.py` contient notre routage d'url est orchestré de cette façon :

```
urlpatterns = [

#Pages principales

url(r'^$', views.description, name='home'),

url(r'description/$', views.description, name='description'),

url(r'recherche/$', views.recherche, name='recherche'),
```

#ici, on a les routes principales qui nous donnent accès aux différentes pages principales
#du sites notamment recherche qui est la page de base pour toutes les pages de recherche

#Menu Ecole_Doctorante:

url(r'recherche/ecole/\$', views.recherche_ecole, name='recherche_ecole'),

*url(r'recherche/ecole/(?P<idparcours>\d+)/(?P<idparcoursf>\d+)/\$', views.recherche_ecole,
name='recherche_ecole'),*

[divers routes d'autres menus]

#Pour chaque menu, nous avons une route qui redirige vers l'affichage du modèle

#concerné

#Page de recherche spécifique :

url(r'^fregion/(?P<regionid>\d+)\\$', views.rendufregion, name='idregion'),

[divers routes de pages de recherche spécifiques]

#Ces routes nous servent à traiter les affichage d'une entrée spécifique

#on peut noter l'arrivée de paramètres d'url à partir d'ici, c'est une composant du modèle

RESTless dont nous reparlerons plus tard.

#Ecole Sous menu

url(r'^recherche/ecole/libelle/\$', views.recherche_ecolefiltre, name='recherche_ecolelibelle'),

*url(r'^recherche/ecole/libelle/(?P<idparcours>\d+)/(?P<idparcoursf>\d+)/\$',
views.recherche_ecolefiltre, name='recherche_ecolelibelle'),*

```
url(r'^recherche/ecole/univ/$', views.recherche_ecolefiltre, name='recherche_ecoleuniv'),
```

```
url(r'^recherche/ecole/univ/(?P<idparcours>\d+)/(?P<idparcoursf>\d+)/$',  
views.recherche_ecolefiltre, name='recherche_ecoleuniv'),
```

[divers route de sous-menu]

#ici, les différentes routes de sous-menu servent principalement à lister en fonction

#d'autres éléments

#form ecole

```
url(r'^recherche/ecole/ecolepost/$', views.recherche_post_ecolefiltre,  
name='recherche_ecolefiltrepost'),
```

[divers routes de post formulaires]

#enfin, les formulaires servent à faire des recherches beaucoup plus affinés.

#Formulaire

```
url(r'contact/$', views.contact, name='contact'),
```

Les API RESTless

Les API RESTless sont un type d'application web ayant une arborescence précise permettant de spécifier tous nos besoins de manière clair dans l'url.

Dans notre cas, notre application étant une application de recherche, le besoin de base se représente de cette façon :

GET /recherche/ecole

Et peut s'affiner de cette façon :

GET /recherche/ecole/libelle/

Ou encore avec des paramètres d'url :

GET /recherche/ecole/100/125/

L'idée des API RESTless est d'appliquer une norme nommé HATEOAS pour "Hypermedia As The Engine Of Application State" décrivant le bon fonctionne d'une application en fonction de la façon dont sont formé les URL.

Les vues

Nous avons créés un certain nombre de vue répondant à différents besoins que nous avons, allant du besoin le plus simple, comme par exemple une vue d'affichage pur et dur :

def description(request):

return render(request, "description.html", locals())

Vers des besoins de plus en plus complexes comme par exemple l'affichage de donnée :

```
18 def recherche_ecole(request, idparcours=0, idparcoursf=25):
19     list_ecoledocto = EcoleDoctorante.objects.all()[int(idparcours):int(idparcoursf)]
20     return rendumenu(request, "recherche.html", locals())
21
```

Où on peut retrouver un modèle de base de donnée utilisant l'ORM Django pour créer des requêtes SQL transparentes pour le développeur.

Les vues liés à des forms

Django possède dans son framework un système de génération automatique de form qui présente plusieurs avantages, à savoir de pouvoir générer du code stable pour la génération de form quelque soit le moteur de templating, de pouvoir générer à la volée des

forms basés sur des modèles, de pouvoir construire un formulaire comme si c'était un objet, avec toutes les possibilités que ça offre (héritage, design pattern, refactoring, etc...).

Création de module de form

Nous avons décidés d'exploiter cette fonctionnalités pour créer des formulaires de recherche et avons aboutis à un premier jet de test dans la page contact, permettant à chacun de laisser un commentaire sur le site :

```
5 class ContactForm(forms.Form):
6     nom = forms.CharField(max_length=100, label="Votre nom")
7     message = forms.CharField(max_length=500, widget=forms.Textarea)
8     captcha = CaptchaField()
```

Un objet de type formulaire possède des attributs que l'on considère comme des entrées du formulaire. Il est possible d'être réellement très exhaustif sur les attributs en spécifiant son type, la façon dont il est affiché, comment il fonctionne, comment il traite les données, etc...

L'ajout d'un module de captcha

Comme ce site est propulsé sur le web, il était nécessaire, pour ne pas dire indispensable d'empêcher des bots d'écrire absolument n'importe quoi, c'est pourquoi nous avons du installer un nouveau module Django nommé [django-simple-captcha](#), servant à spécifier un champ de captcha dans les formulaires dans lesquels il est inséré.

Il est à noter que python possède également une protection CSRF dont nous reparlerons plus tard, mais nous estimons insuffisante face aux bots web.

Pour installer ce module, il a simplement été nécessaire de faire un pip-install dango-simple-captcha pour que le module soit installé dans les libs python, puis de modifier *settings.py* pour l'ajouter au projet, via la liste INSTALLED_APP dans laquelle on a rajouté "catpcha".

Intégration de form

Nous avons ensuite ajouté le formulaire dans la vue de contact de cette façon :

```
198 def contact(request):
199     list_com = ContactCommentaire.objects.all()
200     if(request.method == 'POST'):
201         contactform = ContactForm(request.POST)
202         if(contactform.is_valid()):
203             newcom = ContactCommentaire(nom=contactform.cleaned_data['nom'],
204             commentaire=contactform.cleaned_data['message'])
205             newcom.save()
206         else:
207             erreur = contactform.errors
208     else:
209         contactform = ContactForm()
210     return render(request, "contact.html", locals())
211
```

On peut noter que la vue traite deux cas. Le cas simple où elle affiche le formulaire (c'est le "else" à la fin) et le cas un peu plus complexe où l'utilisateur lui renvoie son form, et où la vue doit à la fois vérifier si il est valide et si oui, ajouter un nouveau commentaire sur la page avant de l'afficher, ou bien s'il est invalide et elle doit afficher l'erreur effectué par l'utilisateur dans son remplissage.

Notez que Django intègre des fonctionnalités de sécurité rendant les formulaires très résistants aux attaques, notamment un mécanisme de "sanitization" des données du formulaire dont on peut voir le résultat dans les données de "cleaned_data".

Un autre élément intéressant est la prise en charge des erreurs de formulaire par Django, qui va se charger d'afficher clairement les soucis dans le traitement du formulaire lorsque l'utilisateur le complète mal, via la variable "errors". Chose amusante, les erreurs de formulaire sont traduites dans la langue spécifiée dans le *settings.py*.

La mise en forme des form

Un formulaire créé avec Django est un objet python qui va être transformé en code HTML par le moteur de templating en fonction des réglages et indications présents dans le formulaire. Mais l'objet de type Form en lui même n'est pas un objet graphique, en réalité, il

contient des widgets (un pour chacun des attributs qui constituent une entrée du formulaire) et ces widgets sont éditables et designables à volonté.

```
libelle = forms.CharField(label="Libellé",required=False, widget=forms.TextInput(attrs={'class':  
'form-control'}))
```

Ici, par exemple nous avons une entrée d'un formulaire qui attend une chaîne de caractère et qui possède comme widget très logiquement un `TextInput`.

On peut noter que l'on a édité le widget pour lui ajouter un indicateur pour le CSS en plus dans son rendu HTML.

Rendu du formulaire

Au final, il ne reste plus que le mécanisme de rendu du template à expliquer.

Ici, nous allons nous contenter de décrire le fonctionnement de l'affichage du formulaire sans entrer dans les détails du moteur de templating de Django, ce qui viendra plus tard dans le rapport.

Donc voici comment le formulaire est traité dans `contact.html` une fois que la vue `contact` l'a créé et envoyé l'objet python au moteur de templating.

```
141 <section id="contact" class="content-section text-center">  
142     <div class="contact-section">  
143         <div class="container">  
144             <h2>Commentez</h2>  
145             <p>Comment avez-vous trouvé notre site ?</p>  
146             <div class="row">  
147                 <div class="col-md-8 col-md-offset-2">  
148                     <form class="form-horizontal" method="POST">  
149                         {% csrf_token %}  
150                         <table>  
151                             {{ contactform }}  
152                         </table>  
153                         <button type="submit" class="btn btn-default">Envoyer un message</button>  
154                     </form>  
155                 </div>  
156             </div>  
157             <p>{{ erreur }}</p>  
158         </div>  
159     </div>  
160 </section>  
161 <br><br><br>
```


On retrouve donc le “`{{ contactform }}`” qui est la variable contenant le formulaire et qui va être traduit en HTML5 par le moteur de templating.

On notera également la présence d’un “`{% csrf_token %}`” qui est un mécanisme de protection des formulaires de python qui permet d’ajouter un identifiant unique à chaque formulaire. Cela permet d’éviter de recevoir plusieurs fois le même formulaire et de l’exécuter accidentellement plusieurs fois.

En rendu HTML, la protection CSRF est transformé en `<input name="csrfmiddlewaretoken" value="o4tqNYeCO0J7ckhPJ8kdUap4AKPijzNojvx8voU1jRcv9K1NX1a7yyDdLTJGD95" type="hidden">`, où la valeur est un hash sha256 garantissant l’unicité du formulaire.


Rendu final

Commentez

Comment avez-vous trouvé notre site ?

Votre nom :

Message :

Captcha : 

Traitement de gros forms et problématique de fonctionnement d'ORM

Le formulaire de contact étant le plus simple, pour la recherche, nous avons été obligés de créer des formulaire avec jusqu'à 12 entrées possibles différentes à potentiellement traiter simultanément, autant dire une sacré recherche d'efficacité dans le traitement des requêtes générées par l'ORM Django.

L'idée est de créer un objet QuerySet (c'est le retour d'une requêtes dans l'ORM de Django), puis d'y appliquer une cascade de filtre en fonction des paramètres demandés dans le formulaire.

```
22 def recherche_ecoledoctorante post(request, idparcours=0, idparcours=25):
23     if(request.method == 'POST'):
24         home_Ecoledoctorante = home_ecoledoctorante(request.POST)
25         if(home_Ecoledoctorante.is_valid()):
26             list_ecoledocto = False
27             if(home_Ecoledoctorante.cleaned_data['libelle'] != ""):
28                 list_ecoledocto = EcoleDoctorante.objects.filter(libelle__contains=home_Ecoledoctorante.cleaned_data['libelle'])
29             if(home_Ecoledoctorante.cleaned_data['ville'] != ""):
30                 if(list_ecoledocto):
31                     list_ecoledocto.filter(ville__contains=home_Ecoledoctorante.cleaned_data['ville'])
32                 else:
33                     list_ecoledocto = EcoleDoctorante.objects.filter(ville__contains=home_Ecoledoctorante.cleaned_data['ville'])
34             if(home_Ecoledoctorante.cleaned_data['libelle_region_avant2016'] != ""):
35                 if(list_ecoledocto):
36                     list_ecoledocto.filter(libelle_region_avant2016__contains=home_Ecoledoctorante.cleaned_data['libelle_region_avant2016'])
37                 else:
38                     list_ecoledocto = EcoleDoctorante.objects.filter(libelle_region_avant2016__contains=home_Ecoledoctorante.cleaned_data['libelle_region_avant2016'])
39             if(home_Ecoledoctorante.cleaned_data['nom_du_directeur'] != ""):
40                 if(list_ecoledocto):
41                     list_ecoledocto.filter(nom_du_directeur__contains=home_Ecoledoctorante.cleaned_data['nom_du_directeur'])
42                 else:
43                     list_ecoledocto = EcoleDoctorante.objects.filter(nom_du_directeur__contains=home_Ecoledoctorante.cleaned_data['nom_du_directeur'])
44             if(home_Ecoledoctorante.cleaned_data['mail'] != ""):
45                 if(list_ecoledocto):
46                     list_ecoledocto.filter(mail__contains=home_Ecoledoctorante.cleaned_data['mail'])
47                 else:
48                     list_ecoledocto = EcoleDoctorante.objects.filter(mail__contains=home_Ecoledoctorante.cleaned_data['mail'])
49             if(home_Ecoledoctorante.cleaned_data['site_web'] != ""):
50                 if(list_ecoledocto):
51                     list_ecoledocto.filter(site_web__contains=home_Ecoledoctorante.cleaned_data['site_web'])
52                 else:
53                     list_ecoledocto = EcoleDoctorante.objects.filter(site_web__contains=home_Ecoledoctorante.cleaned_data['site_web'])
54             if(list_ecoledocto):
55                 list_ecoledocto = list_ecoledocto[int(idparcours):int(idparcours+25)]
56             else:
57                 erreur = home_Ecoledoctorante.errors
58     return rendumenu(request, "recherche.html", locals())
59
```

Ici, on peut voir un traitement de grosse requête filtrée. Si on suppose que tous les éléments du formulaire ont été remplis, alors la liste d'école doctorante sera filtrée en fonction de 6 éléments.

On peut noter la présence de délimiteurs tout à la fin (idparcours et idparcours+25), nous avons d'abord songé à délimiter chaque requête une à une, ce qui a généré un bug très complexe que je vais m'atteler à décrire dans la section suivante.

Bug de gestion de délimiteur dans l'ORM Django

Un ORM est avant tout une machine à remplacer le code SQL par son pendant relatif dans la gestion des objets dans le langage duquel il découle.

Cela signifie par exemple que pour l'ORM de Django, le but du jeu est de faire faire à l'utilisateur 100% de ses requêtes via `Models.objects` plutôt qu'en SQL, afin de lui permettre de ne travailler qu'avec un seul langage (Python en l'occurrence) et éventuellement de rendre ses migrations d'un SGDB à un autre transparentes.

Mais dans certains cas, le développement d'un ORM est abordé de manière trivial et cela amène à des problèmes logiques étranges.

En l'occurrence, dans notre cas, nous voulions faire des choses du genre :

```
list_ecoledocto =  
EcoleDoctorante.objects.filter(libelle__contains=home_Ecoledoctorante.cleaned_data['libelle'])[int(  
idparcours):int(idparcoursf)]
```

```
list_ecoledocto.filter(nom_du_directeur__contains=home_Ecoledoctorante.cleaned_data['nom_du_  
directeur'])[int(idparcours):int(idparcoursf)]
```

Et Django nous indiquait que *list_ecoledocto* était un `ConstantQuerySet`, et que de ce fait, on ne pouvait plus appliquer de fonction de transformation dessus.

Pourtant, ça semble logique de filtrer une table, appliquer une limite dessus, puis filtrer le contenu de la limite...

Mais en fait, pas tant que cela si on commence à regarder de plus près le fonctionnement interne de l'ORM de Django.

En réalité, l'ORM transcrit chaque transformation sous la forme d'une partie d'opération SQL ou d'une opération SQL complète.

De cette façon, *list_ecoledocto =*
EcoleDoctorante.objects.filter(libelle__contains=home_Ecoledoctorante.cleaned_data['libelle'])

peut se traduire par

```
SELECT * FROM home_EcoleDoctorante WHERE libelle LIKE ` %data% `;
```

Et si ensuite, on fait :

```
list_ecoledocto.filter(nom_du_directeur__contains=home_Ecoledoctorante.cleaned_data['nom_du_
directeur'])
```

La requête devient :

```
SELECT * FROM home_EcoleDoctorante WHERE libelle LIKE ` %data% ` AND nom_du_directeur
LIKE ` %data1% ` ;
```

Et l'ORM de Django à intelligemment lié les deux requêtes en utilisant le mot clé "AND".

Mais si on dis :

```
list_ecoledocto =
EcoleDoctorante.objects.filter(libelle__contains=home_Ecoledoctorante.cleaned_data['libelle'])[int(
idparcours):int(idparcoursf)]
```

```
list_ecoledocto.filter(nom_du_directeur__contains=home_Ecoledoctorante.cleaned_data['nom_du_
directeur'])[int(idparcours):int(idparcoursf)]
```

Là, la première partie de la requête se traduit par :

```
SELECT * FROM home_EcoleDoctorante WHERE libelle LIKE ` %data% ` LIMIT 0, 25;
```

Et la seconde souhaite ajouter "AND nom_du_directeur LIKE ` %data1% ` LIMIT 0, 25".

Or, la syntaxe SQL exige un unique LIMIT situé en fin de requête, alors Django ne sait pas que faire : doit il appliquer uniquement le premier LIMIT ? Ou le second ? Ou créer une sous requête pour les appliquer tout les deux ? Ou encore faire une jointure avec deux fois la même table en éliminant les doublons ?

Donc pour éviter cette ambiguïté dans l'intention de code, Django préfère déclarer un QuerySet sur lequel on a appliqué un LIMIT comme constant.

Et c'est fondamentalement très intéressant car cela montre les limites de transcription des ORM qui souhaitent créer des modèles orientés objet pour cacher le code SQL mais qui ne peuvent pas dépasser les limites de la syntaxe SQL.

wrapper de vue

Avant de passer au plus parlant, c'est à dire le système de templating, un petit mot sur une subtilité de Django qui sont les wrapper de vue.

Comme Django utilise des vues présentées la plupart du temps comme des fonctions (même si de manière plus avancée, il est possible de déclarer les vues comme des objets), il est également possible d'effectuer des cascades de vue en appelant une autre vue dans une vue plutôt qu'en envoyant directement sa sortie au moteur de templating.

C'est ce que nous avons fait en utilisant un wrapper qui était censé avoir plusieurs utilités et qui sert maintenant uniquement à gérer le parcours de liste de donnée :

```
259 # wrapper d'ajout de form dans le menu
260 def rendumenu(request, uri, localvar):
261     if 'idparcours' in localvar:
262         if(int(localvar['idparcours']) >= 25):
263             prepcours = int(localvar['idparcours'])-25
264             prepcoursf = int(localvar['idparcoursf'])-25
265             print("retour (" + localvar['idparcours'] + ") " + "(" + str(prepcours) + ")")
266             localvar['idparcours']=int(localvar['idparcours'])+25
267             localvar['idparcoursf']=int(localvar['idparcoursf'])+25
268
269     locals().update(localvar)
270
271     return render(request, uri, locals())
```

La fonction `locals()` de python sert à récupérer toutes les données locales à un bloc donné et les retourne dans un objet de type dictionnaire (clé -valeur).

C'est la raison pour laquelle ce wrapper effectue une opération assez risquée qui consiste à prendre le dictionnaire des variables locales à la fonction `rendumenu` et y applique les variables locales d'une vue précédente afin de fusionner la sortie de `rendumenu` avec la fonction de traitement de donnée précédente qui l'a appelé avant de passer la main au moteur de templating.

Moteur de templating

Django intègre un moteur de templating basé sur Jinja2 qui est un langage de rendu dérivé de python.

Il a la particularité de proposer un système de structuration par bloc permettant de définir et redéfinir clairement chaque composant du site.

```
16     <body>
17         {% block header %}
18         {% include "header.html" %}
19         {% endblock %}
20     {% block content %}
21     <h1>Datagouv</h1>
22     {% endblock content %}
23     {% block footer %}
24     {% include "footer.html" %}
25     {% endblock %}
26
27     </body>
28
```

De cette façon, si un bloc "content" est créé dans base.html, qu'on crée une page page1.html qui extend de base.html et qu'on écrit dans cette page

```
{% block content %}
```

```
<h1>Le site datagouv</h1>
```

```
{% endblock content %}
```

Tout le reste du contenu de la page ne sera pas modifié.

Cela permet d'éviter énormément de réécriture de balise que l'on peut retrouver dans d'autres langages, comme PHP ou Ruby, de plus, il existe beaucoup de mécanismes de refactoring avec Jinja2, l'extends qui équivaut à une relation d'héritage entre un rendu et un autre, l'inclusion qui permet d'inclure un ensemble de bloc dans un autre, des mécanismes

de mise en forme de donnée passés par les vues avec toutes les méthodologies qu'on retrouverait dans un langage normal (boucle, condition, référence, variable, etc...).

Tests Croisés

1. Groupe 1 – Jeu narratif avec pour but de rester en vie

Un jeu narratif développé en langage Java. L'utilisation est seulement par le biais d'un terminal avec l'utilisation d'un fichier compiler en Java.

Remarques :

- Un jeu linéaire avec une seule trame d'histoire, lorsqu'on sélectionne les mêmes choix, on se retrouve dans la même histoire.
- Plusieurs choix sont à disposition, cependant un des «choix» inutile : L'option « fuir », elle ne fait rien.
- Incarnation d'un des personnages prédéfinis, avec des attaques différentes cependant le nombre de dégâts reste identiques. Pas de « hasard ».

Bugs / Problèmes :

- Sélection des choix avec des chiffres compris entre 1 & 4. Cependant lorsque cela n'est plus un chiffre, cela affiche un « Exception in thread ».

Test effectué : « é » « a » lors d'un choix.

- Lors du lancement du jeu, un choix est possible entre « jouer » et « quitter ». Lors de la sélection de « quitter », le programme envoie « YOU DIED ! » mais après quelque saut de ligne, le jeu envoie des lignes de narrations. Alors que j'ai choisi de quitter.
- Lorsque l'on joue au jeu, à un moment de la sélection, une phrase narrative défile et une phrase avec « BUG!!! » apparaît.

Améliorations possibles :

- Jeu développé très probablement avec des « switch cases » ou bien des « if ».
Conseil : Possibilités de développer un jeu à choix multiple avec des possibilités multiples avec l'utilisation de nœuds prédéfini, et faire ça en arbre « n-ères » afin de rendre les choix et l'histoire de manière plus dynamique.
- Plusieurs personnages sont prédéfinis avec des noms imposés. Conseil : Langage Java étant un langage orienté objet, possibilités de créer une classe personnage et de faire de l'héritage afin de créer différentes classes. Tel que « voleur », « sorcier » avec différents types d'attaques prédéfini au niveau de l'héritage.
- Seulement un monstre apparaît au cours de l'histoire. Même principe que dans la précédente amélioration. Créer une classe " Monstre " avec différents héritages.
- Face au différents bugs des caractères spéciaux, ou caractère simple. Spécifier davantage au niveau de la condition.
- Difficulté de quitter le jeu, rajouter une fonction `System.exit(« exit code »);`
- Pour rendre le jeu un peu plus aléatoire, utilisez la fonction `Math.random()`.
- Possibilités de mettre une interface graphique pour la sélection.

2. Groupe 2 - Jeu Graphique avec génération procédurales avec un moteur de jeu.

Objectif du projet :

Création d'un jeu dans un monde généré en procédural, avec l'utilisation d'un moteur 3D appelé "Godot" ; avec une utilisation de "GD script " dont l'utilisation fonctionne avec le langage python.

Remarques :

- Interface graphique très intéressante, animation par le moteur de recherche sont belles.
- Mouvement du personnage un peu rapide
- Une barre de vie est manquante pour le joueur, ne sachant pas trop quand est-ce qu'il meurt.
- Lors de la mort du joueur, la fenêtre est réduite, ça reste figé.
- Lorsque le joueur trouve la sortie, la fin apparaît, suivi d'une image qui peut paraître effrayant, ainsi toucher moins de public.
- Fonctionne sous linux, cependant nécessite le changement des droits par chmod. A essayer sur d'autres systèmes d'exploitations.

Bugs / Problèmes :

- La taille de l'ombre n'est pas réinitialisée de bonne taille lors d'un saut. Elle reste grande lors de la retombée du personnage.
- Lors d'une attaque du personnage, le personnage est bloqué. Ce qui fait que l'attaque est bloquante sur le mouvement.

-
- La vision du personnage n'est pas donnée, ce qui ne permet pas au joueur de définir l'endroit où il souhaite viser afin de taper avec son arme.
 - Le masque de collision (hitbox), n'est pas très bien défini sur les monstres. Lorsqu'on voit le personnage taper sur le monstre, par moment cela n'affecte pas le monstre.
 - Un problème majeur, lié à l'utilisation de la collision avec un monstre et un mur juste derrière. Lorsque la collision est effectuée entre le personnage et le monstre, le personnage réagit par un mouvement arrière (knock back), cependant celui-ci s'il passe derrière un mur, le personnage se retrouve bloqué.
 - Lors de la collision avec n'importe quel obstacle, il apparaît une fraction de seconde un carré noir sur le jeu.
 - Au niveau de la carte du jeu, à un endroit en particulier, un escalier est dessiné. Le personnage peut s'y rendre, cependant celui-ci dans ce cas en hauteur, il peut continuer comme s'il n'était pas monté. La notion d'hauteur n'est pas gérée.

Améliorations possibles :

- Création d'un menu avec définition des touches de jeux, ou bien apparition des différentes touches pour jouer. Le joueur ne devine pas comment ça se joue.
- Diminution de la vitesse de déplacement du personnage, ce qui permettrait un meilleur gain de durée de vie sur le jeu.
- Pour les problèmes liés à l'ombre / l'attaque bloquant / hitbox / collision, voir sur le moteur de jeu si c'est possible de mieux gérer.
- Enlever les étages, afin de ne pas les gérer. Ce qui éviterait les incohérences.
- Une fin avec une meilleure ambiance, et afficher un crédit afin de pouvoir remercier chaque participant dans ce projet, et les différents acteurs de votre projet.
- Les monstres ne sont pas mobiles, une sous-partie du projet montre un personnage (non joueur) qui poursuit un joueur. Je pense qu'il pourrait être judicieux de l'appliquer sur certains types de monstres voir tous les monstres afin d'avoir un petit peu de challenge.