

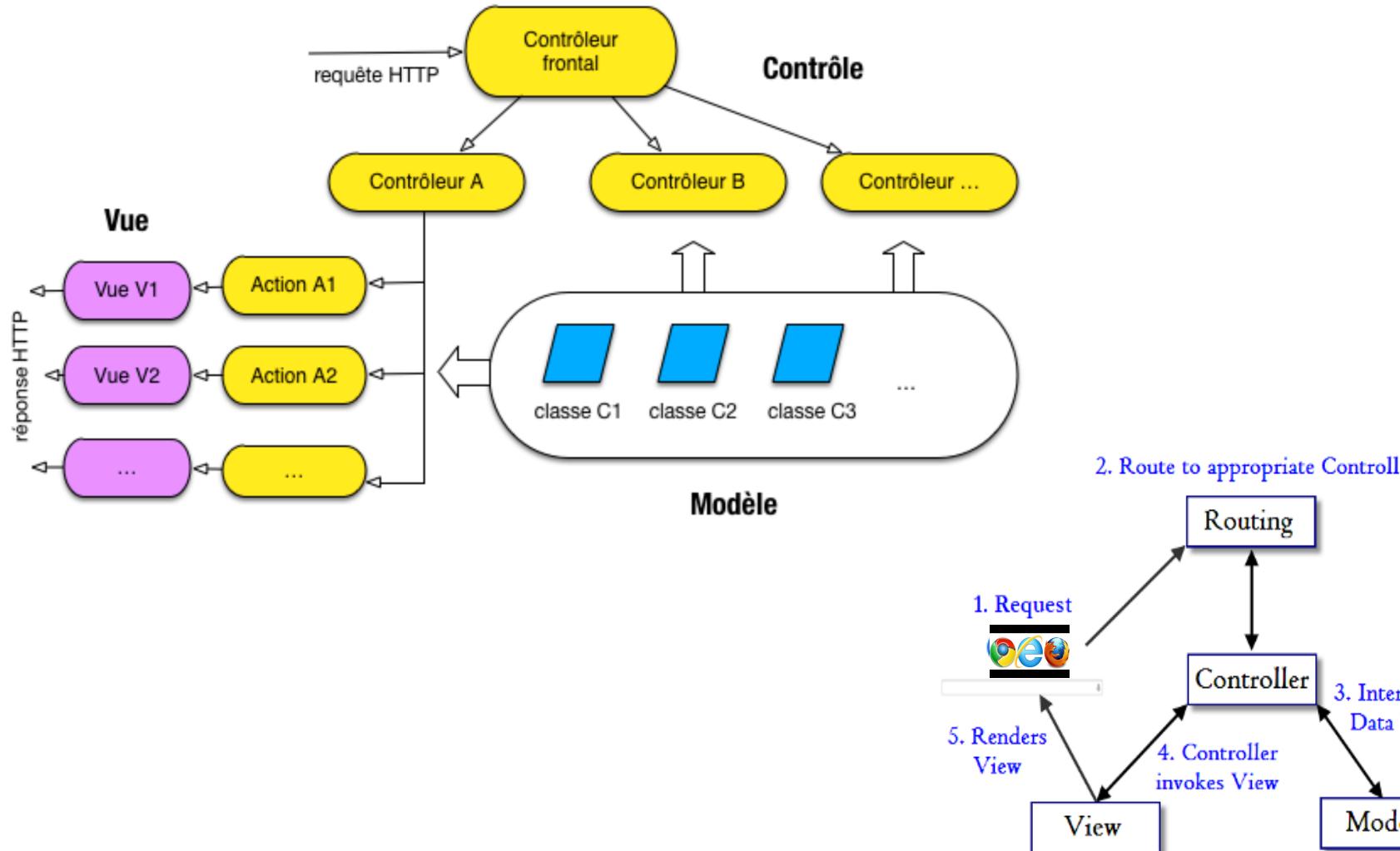
Introduction au Framework Django

L3 MIAGE CL

***Jean-François Pradat-Peyre,
Université Paris Nanterre - UFR SEGMI***

2020-2021

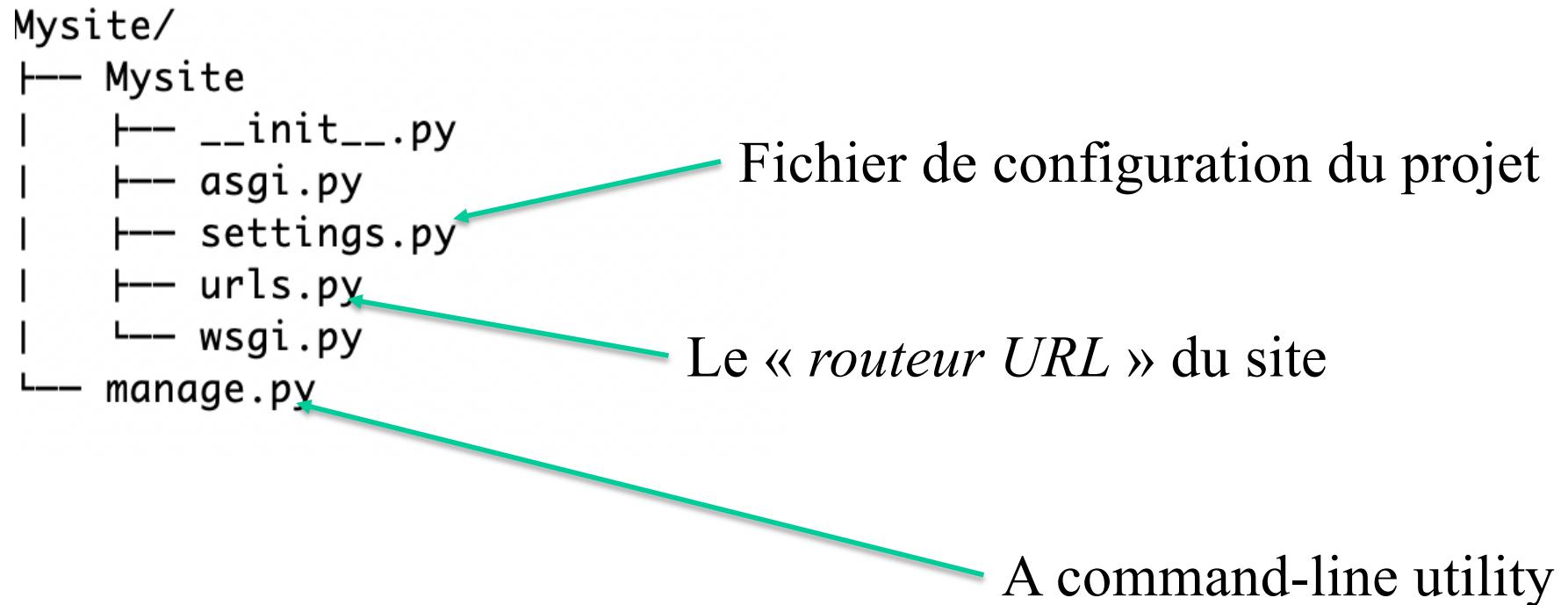
Architecture MVC (Web)



Django en quelques mots

- ❖ Django est un Framework Python qui implémente le modèle MVC en python
- ❖ Il s'installe dans un environnement virtuel (ou non) par la commande
 - `python -m pip install Django`
- ❖ On peut vérifier la version avec la commande
 - `pip list`
 - en octobre 2020, la version installée pour ce cours est la 3.1.2
- ❖ En Django on crée des projets (web) qui contiendront des applications
- ❖ Créer un projet de nom Mysite:
 - `django-admin startproject Mysite`
- ❖ Créer des applications avec
 - `python manage.py startapp myAPP`

Création d'un projet : arborescence



Django URL Dispatcher (routage d'URL)

- ❖ Le client envoie une requête PUT ou GET du protocole HTTP
- ❖ Cette requête est sous forme d'URL avec éventuellement des paramètres
 - Certaines données sont dans l'entête (et non visible dans l'URL)
- ❖ Côté Django, le serveur reçoit la requête et cherche à associer à la requête une action (fonction python) à l'aide d'une views ; pour cela il utilise un fichier urls.py (en pur Python, déterminé par la variable ROOT_URLCONF)
- ❖ Django charge le module urls.py et cherche la variable urlpatterns qui doit être une séquence d'instances de django.urls.path() et/ou de django.urls.re_path()
- ❖ Django parcours séquentiellement chaque pattern d'URL et s'arrête à la première instance qui associe l'URL demandée avec un pattern
- ❖ Django appelle alors la fonction de la views correspondante.

Exemple de routes avec Django (*fichier urls.py*)

```
from django.urls import path
from . import views
urlpatterns = [
    path('articles/2003/', views.special_case_2003),
    path('articles/<int:year>', views.year_archive),
    path('articles/<int:year>/<int:month>', views.month_archive),
    path('articles/<int:year>/<int:month>/<slug:slug>', views.article_detail),
]
```

Les « views » sont des actions sous Django et sont implémentées par des fonctions Python avec au moins le paramètre **request**

- A request to **/articles/2005/03/** would match the third entry in the list. Django would call the function **views.month_archive(request, year=2005, month=3)**.
- **/articles/2003/** would match the first pattern in the list, not the second one, because the patterns are tested in order, and the first one is the first test to pass. Feel free to exploit the ordering to insert special cases like this. Here, Django would call the function **views.special_case_2003(request)**
- **/articles/2003** would not match any of these patterns, because each pattern requires that the URL end with a slash.
- **/articles/2003/03/building-a-django-site/** would match the final pattern. Django would call the function **views.article_detail(request, year=2003, month=3, slug="building-a-django-site")**.

Expressions régulières en Python

On utilise des symboles qui ont une signification:

```
. ^ $ * + ? { } [ ] \ | ( )
```

- . Le point correspond à n'importe quel caractère.
- ^ Indique un commencement de segment mais signifie aussi "contraire de"
- \$ Fin de segment
- [xy] Une liste de segment possible. Exemple [abc] équivaut à : a, b ou c
- (x|y) Indique un choix multiple type (ps|ump) équivaut à "ps" OU "UMP"
- \d le segment est composé uniquement de chiffre, ce qui équivaut à [0-9].
- \D le segment n'est pas composé de chiffre, ce qui équivaut à [^0-9].
- \s Un espace, ce qui équivaut à [\t\n\r\f\v].
- \S Pas d'espace, ce qui équivaut à [^ \t\n\r\f\v].
- \w Présence alphanumérique, ce qui équivaut à [a-zA-Z0-9_].
- \W Pas de présence alphanumérique [^a-zA-Z0-9_].
- \ Est un caractère d'échappement

Il est possible de d'imposer le nombre d'occurrences avec la syntaxe suivante:

- A{2} : on attend à ce que la lettre A (en majuscule) se répète 2 fois consécutives.
- BA{1,9} : on attend à ce que le segment BA se répète de 1 à 9 fois consécutives.
- BRA{,10} : on attend à ce que le segment BRA ne soit pas présent du tout ou présent jusqu'à 10 fois consécutives.
- V0{1,} : on attend à ce que le segment V0 soit présent au moins une fois.

Exemple de routes avec Django avec expressions régulières

```
from django.urls import path, re_path

from . import views

urlpatterns = [
    path('articles/2003/', views.special_case_2003),
    re_path(r'^articles/(\?P<year>[0-9]{4})/$', views.year_archive),
    re_path(r'^articles/(\?P<year>[0-9]{4})/(\?P<month>[0-9]{2})/$', views.month_archive),
    re_path(r'^articles/(\?P<year>[0-9]{4})/(\?P<month>[0-9]{2})/(\?P<slug>[\w-]+)/$', views.article_detail),
]
```

```
from django.urls import include, re_path

urlpatterns = [
    re_path(r'^index/$', views.index, name='index'),
    re_path(r'^bio/(\?P<username>\w+)/$', views.bio, name='bio'),
    re_path(r'^weblog/', include('blog.urls')),
    ...
]
```

Démarrer le serveur de développement d'un projet

- ❖ Dans le répertoire Mysite (où se trouve le fichier manage.py)
 - Lancer la commande '**python manage.py runserver**'
 - Un serveur HTTP local (sur le port 8000 par défaut est démarré)
 - Il est possible d'utiliser un autre port en ajoutant le numéro de port choisi à la fin
 - Lancer la commande '**python manage.py runserver NUM_PORT**' (> 1024)
- ❖ On observe (sur le terminal) :

```
Performing system checks...
```

```
System check identified no issues (0 silenced).
```

```
You have unapplied migrations; your app may not work properly until they are applied.  
Run 'python manage.py migrate' to apply them.
```

```
October 13, 2020 - 15:50:53
```

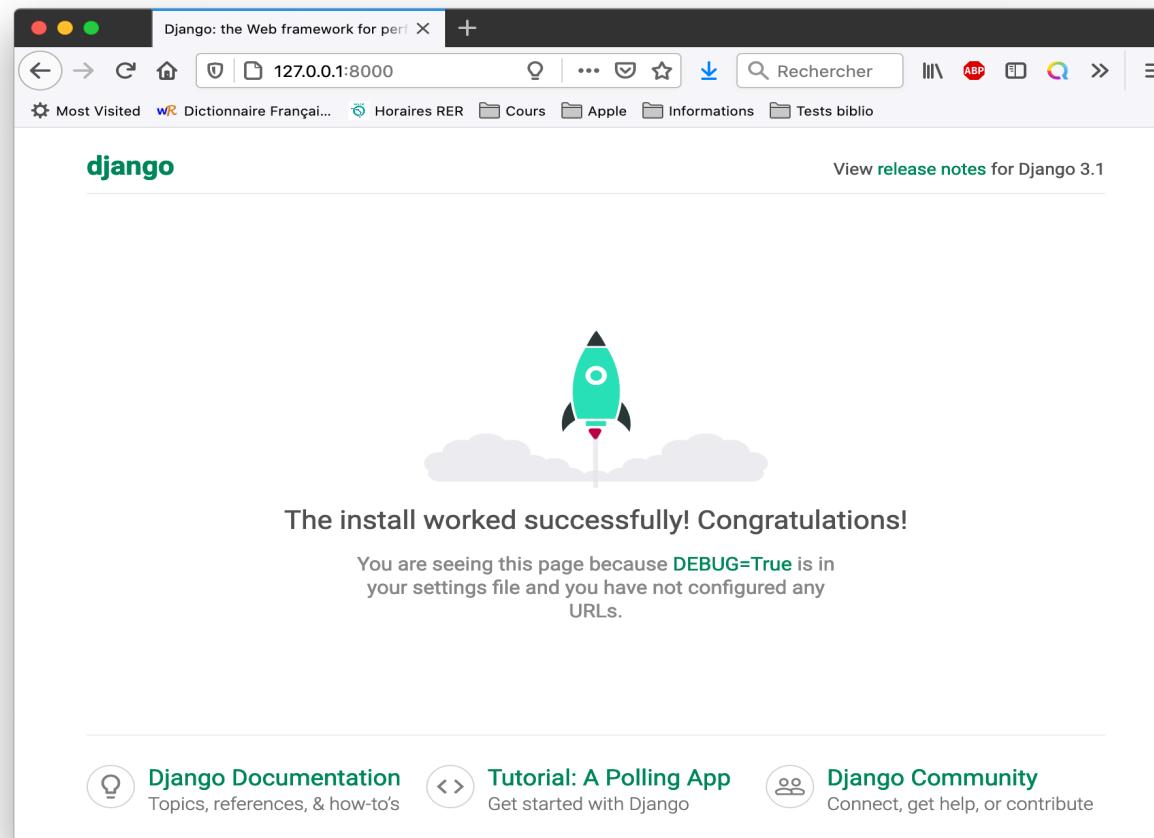
```
Django version 3.1, using settings 'mysite.settings'
```

```
Starting development server at http://127.0.0.1:8000/
```

```
Quit the server with CONTROL-C.
```

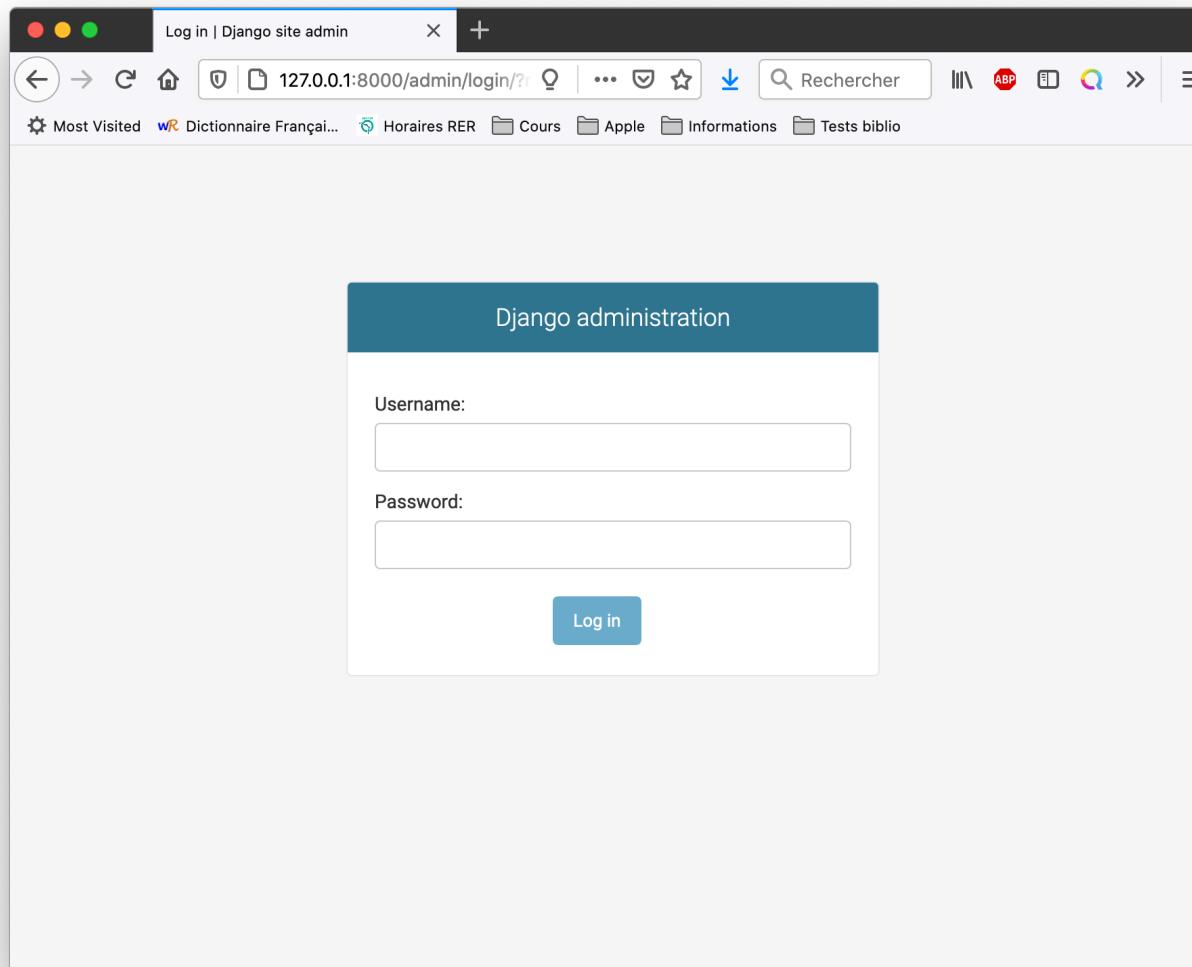
Pages de base (après installation standard)

- ❖ Une fois le serveur démarré, vous devez voir la page suivante dans un navigateur à l'URL `http://127.0.0.1:8000/`



Pages de base (après installation standard)

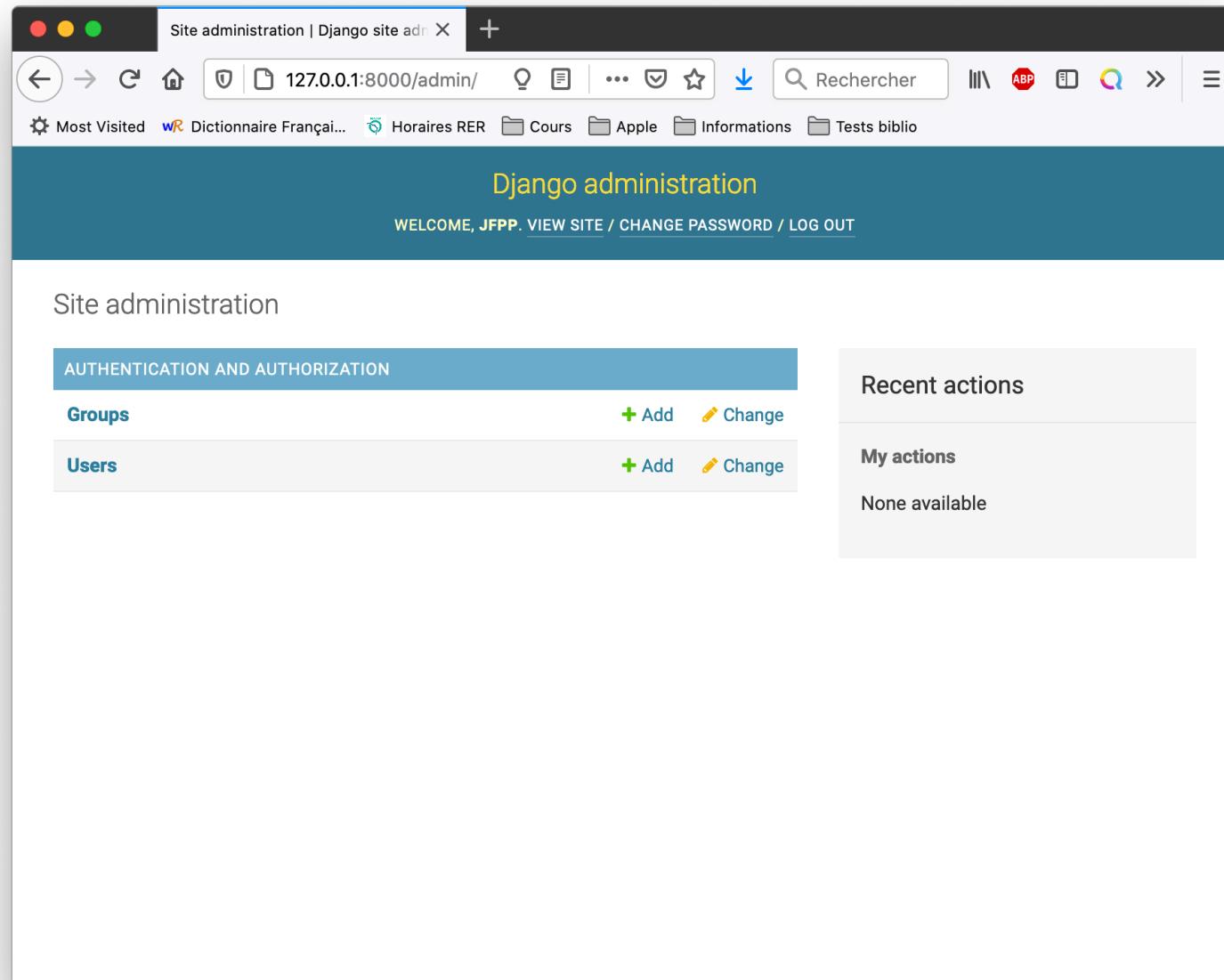
- ❖ Il existe aussi une page admin accessible à l'URL <http://127.0.0.1:8000/admin/>



Pages de base (après installation standard)

- ❖ Pour se connecter dans la page admin il faut que la base de données (par défaut sqlite3) soit en phase avec votre projet : il faut faire une « migration »
- ❖ Pour cela faire ‘python manage.py migrate’ (après avoir arrêté le serveur)
- ❖ Vous pouvez alors créer un « Super utilisateur » qui vous permettra de vous connecter à la page admin
- ❖ Taper la commande ‘python manage.py createsuperuser’
 - Vous indiquez un login puis un mail puis un mot de passe
 - Ces données sont stockées dans la base de données locale (par défaut sqlite3)
 - Vous pouvez (après avoir relancé le serveur local) accéder aux données de la page admin après authentification

Accès à la page admin de base <http://127.0.0.1:8000/admin/>



Créer une première application dans le projet

- ❖ Un projet est constitué de plusieurs application : gestion des clients, gestion des stocks, gestion technique des bases, etc.
- ❖ Chaque application aura son propre répertoire, sa propre arborescence et pourra être partagée entre différents projets
- ❖ Une application se crée par ‘python manage.py startapp myFisrtApp’



The screenshot shows a terminal window titled 'Mysite — -bash — 124x25'. The command 'tree myFisrtApp/' is run, displaying the directory structure:

```
(Projet1) imac-jeff:Mysite jfpp$ tree myFisrtApp/
myFisrtApp/
├── __init__.py
├── admin.py
├── apps.py
└── migrations
    └── __init__.py
├── models.py
└── tests.py
└── views.py

1 directory, 7 files
(Projet1) imac-jeff:Mysite jfpp$
```

Peupler l'application

- ❖ Créer des vues (views) avec des fonctions python qui seront appelées par le contrôleur selon le routage URL défini

```
Users > jfpp > PROG_PY > Mysite > myFisrtApp > views.py > ...
```

```
1  from django.http import HttpResponse
2  import datetime
3
4  def index(request):
5      now = datetime.datetime.now()
6      html = "<html><body>Ma première application donne l'heure : Il est %s.</body></html>" % now
7      return HttpResponse(html)
```

- ❖ Créer des liens vers ces vues (dans le fichier de routage urls.py)

```
from django.contrib import admin
from django.urls import path
from myFisrtApp import views

urlpatterns = [
    path('', views.index, name='home'),
    path('admin/', admin.site.urls),
]
```

Résultat : la racine pointe sur la vue créée et admin sur admin

The image displays two screenshots of a web browser window, likely Safari, running on macOS. Both screenshots show the URL `127.0.0.1:8000/` in the address bar.

Screenshot 1 (Top): This screenshot shows the output of a custom view. The page content reads: "Ma première application donne l'heure : Il est 2020-10-14 12:08:18.384507."

Screenshot 2 (Bottom): This screenshot shows the Django Admin interface. The title bar says "Site administration | Django site admin". The main header reads "Django administration" and "WELCOME, JFPP. VIEW SITE / CHANGE PASSWORD / LOG OUT". The left sidebar is titled "Site administration" and contains sections for "AUTHENTICATION AND AUTHORIZATION" (Groups, Users) and "RECENT ACTIONS" (Recent actions, My actions). The "Recent actions" section indicates "None available".

Introduire la persistance des données : les modèles et les BD

- ❖ Objectifs : utiliser des données en BD pour fournir un service à travers une application
- ❖ Deux solutions sont possibles en Django :
 1. Créer des données dans les modèles python de l'app (fichier models.py)
 - ✓ Les migrer (en deux étapes) dans la base de données
 - ▶ python manage.py makemigrations myFirstApp : prépare la migration
 - ▶ python manage.py migrate : réalise la migration
 - ✓ Les modèles peuvent être importés dans l'interface d'administration du site
 - ✓ Inconvénients : les données sont pilotés par le code applicatif
 - ✓ Avantage : assez grande simplicité de mise en œuvre
 2. Créer les données (et le schéma de données) en base
 - ✓ Récupérer la représentation en python : **inspectdb NOM_DE_TABLE**
 - ✓ Introduire les modèles dans les modèles de l'app (fichier models.py)
 - ✓ Avantage : les données sont pilotés par la base de données
 - ✓ Inconvénients : plus grande complexité de mise en œuvre (à relativiser)

Créer des données dans les modèles python de l'app : exemple

Users > jfpp > PROG_PY > Mysite > myFisrtApp >  models.py > ...

```
1  from django.db import models
2
3  # Create your models here.
4
5  class Statut(models.Model):
6      nom = models.CharField(max_length=10)
7      nomL = models.CharField(max_length=50)
8      code = models.PositiveIntegerField(unique=True)
9
10     def __str__(self):
11         return self.nom+'/'+ self.nomL+'/' Code =' +str(self.code)
12
```

Users > jfpp > PROG_PY > Mysite > myFisrtApp >  admin.py

```
1  from django.contrib import admin
2
3  from .models import Statut
4
5  admin.site.register(Statut)
6
```

Suite exemple

- ❖ Tester : `python manage.py check`
- ❖ Préparer : `python manage.py makemigrations myFisrtApp`
- ❖ Migrer : `python manage.py migrate`
- ❖ Si OK : on peut voir (site page admin : la table `myFisrtApp_statut`)

The screenshot shows the Django administration interface for the 'Status' model. The left sidebar lists 'Groups', 'Users', and 'MYFISRTAPP'. Under 'MYFISRTAPP', 'Status' is highlighted and has a yellow background. A green '+ Add' button is next to it. The main content area is titled 'Select statut to change' and contains a table with two rows:

<input type="checkbox"/> STATUT	MCF/Maître de Conférences/ Code =301
1 statut	

At the top right of the content area is a 'ADD STATUT +' button.

Seconde solution : piloter les données en Base

- ❖ Pour ce faire il faut avoir installer une BD (par défaut SQLite3)
- ❖ Nous choisissons MySql
 - Installer MySql-client (le serveur sera installé avec MAMP ou LAMP)
 - ✓ Sous windows voir : https://www.youtube.com/watch?v=kEnD_KN7P-k
 - ✓ Sous Mac : brew install mysql ; brew install mysql-client
 - ✓ Sous Linux : apt-get install mysql-client mysql-common
 - Autre solution : rester avec SQLite3 ou installer **PostgreSQL**
- ❖ Installer alors (sous venv) l'ORM mysql-client : pip install mysqlclient
- ❖ pip list :

```
[Projet1] imac-jeff:Mysite jfpp$ pip list
Package      Version
-----
asgiref      3.2.10
Django        3.1.2
mysqlclient   2.0.1
pip           20.2.3
pytz          2020.1
setuptools    47.1.0
sqlparse       0.4.1
```

- ❖ Ajouter dans le fichier Mysite/settings.py le driver de la base choisie (ici MySql) qui devient la BD par défaut (la base et les détails de connexion doivent être corrects)

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'TESTDB',
        'USER': 'test',
        'PASSWORD': 'test123',
        'HOST': '127.0.0.1',
        'PORT': '3306',
    },
    'sqlite': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

ORM Django : Translation Structure BD et Classe Python

La structure d'une table INDIVIDU (*MySQL*)

Name	Type	Collation	Attributes	Null	Default	Comments	Extra
ID	int(11)			No	None		AUTO_INCREMENT
NOM	varchar(20)	utf8_general_ci		No	None		
PRENOM	varchar(20)	utf8_general_ci		No	None		
mail	varchar(50)	utf8_general_ci		No	None		

La déclaration de la classe Individu (*dans models.py*)

Non nécessaire
Géré par l'ORM

```
class Individu(models.Model):
    # id = models.AutoField(db_column='ID', primary_key=True) # Field name made lowercase.
    nom = models.CharField(db_column='NOM', max_length=20) # Field name made lowercase.
    prenom = models.CharField(db_column='PRENOM', max_length=20) # Field name made lowercase.
    mail = models.CharField(max_length=50)
```

*Peut être construit automatiquement par
python manage.py inspectdb individu*