Introduction au framework Django
Création d'une page perso

Myriam BEGEL

Cachan Réseau à Normale Sup'

Mardi 13 décembre 2016

1 Présentation

2 Création d'une page perso
- Créer un projet
- Création d'un modèle
- Découverte de l'interface d'administration
- Utiliser un gabarit
- Formulaire de contact
- Tester son code

Plan

1 Présentation

2 Création d'une page perso
- Créer un projet
- Création d'un modèle
- Découverte de l'interface d'administration
- Utiliser un gabarit
- Formulaire de contact
- Tester son code

Django

Django est un framework de développement web en Python.

The web framework for perfectionists with deadlines.

La documentation est parfaite, disponible pour 5 versions et en 8 langues.[1]

[1] Doc

Modèle MVC

Architecture Modèle - Vue - Contrôleur : classique

Modèle : représente une information, *un objet* généralement enregistré dans la base de données

Vue : s'occupe de la *visualisation des données*, d'actions de l'utilisateur comme la soumission de formulaire

Contrôleur : s'occupe des évènements de l'utilisateur, communique avec les modèles et transmet les données à la vue

Modèle MVC

Requête HTTP

Contrôleur

Réponse HTML

Demande

Données

Modèle

Vue

Modèle MVT

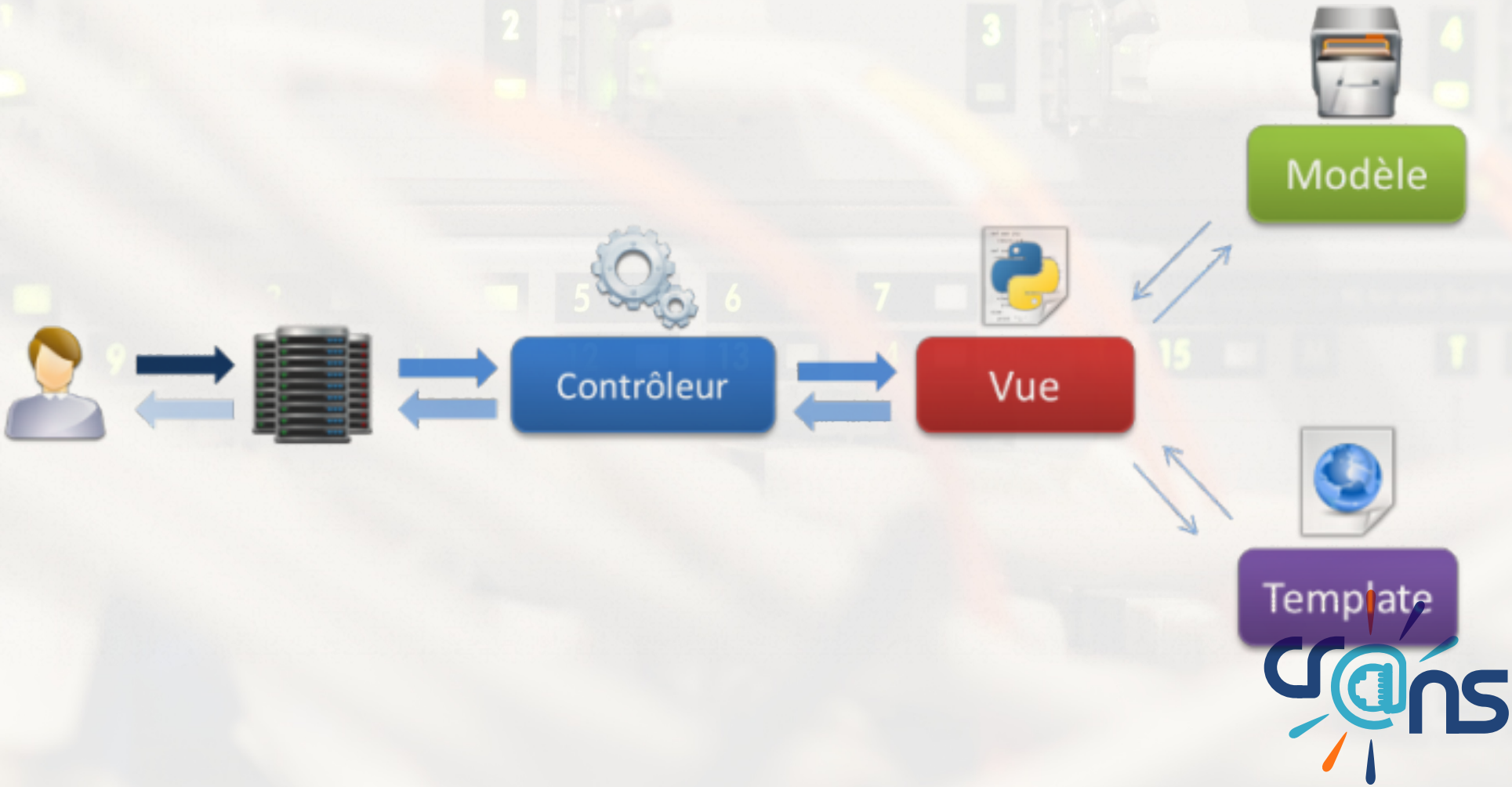
Architecture Modèle - Vue - Template : Django
S'occupe lui-même du Contrôleur

Modèle : représente une information, *un objet* généralement enregistré dans la base de données

Vue : s'occupe de la *visualisation des données*, d'actions de l'utilisateur comme la soumission de formulaire

Gabarit *(=template)* : *fichier HTML* avec des extensions : boucle for, conditionnelle if, variables ...

Modèle MVT



Modèle MVT

1. L'utilisateur envoie une requête
2. Django appelle la Vue correspondant à l'URL
3. La Vue récupère les données, le gabarit
4. et génère un rendu HTML pour l'utilisateur

Plan

1 Présentation

2 Création d'une page perso
- Créer un projet
- Création d'un modèle
- Découverte de l'interface d'administration
- Utiliser un gabarit
- Formulaire de contact
- Tester son code

Démarrer un projet

Projet Git : https://gitlab.crans.org/begel/perso

```
django-admin startproject perso
```

perso/
- manage.py
- perso/
  - __ init__ .py
  - settings.py
  - urls.py
  - wsgi.py

```
./manage.py runserver
```

Première application

```
django-admin startapp perso_app
```

perso/

- manage.py
- perso/
- perso_app/
  - __init__ .py
  - admin.py
  - apps.py
  - models.py
  - tests.py
  - views.py

Ajout de `perso_app` dans les `INSTALLED_APPS` de `settings.py`

Afficher un fichier HTML

Récupérer le gabarit et la feuille de style fournie.

Dans perso/urls.py, inclure les routes de perso_app/urls.py

Ecrire une route dans perso_app/urls.py et la vue correspondante.

Premier modèle : Publication

Un modèle est une classe python qui représentera un objet dans la base de données.
Les modèles étendent la classe `models.Model`
Les champs de l'objet que l'on veut enregistrer sont des extensions de `models.Field`. Il y en a pour tous les goûts.[2]

[2] Doc

Premier modèle : Publication

```
class Publication(models.Model):
    titre = models.CharField(max_length = 200)
    auteurs = models.CharField(max_length = 200)
    date = models.DateField()
    editeur = models.CharField(max_length = 200, blank =
     True)
    description = models.TextField()

    def __str__(self):
        return self.titre
```

Premier modèle : Publication

Pour créer la table dans la base de données :

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

A faire à chaque fois que l'on crée/change un modèle.
Pour "jouer" avec le modèle, on peut ouvrir un shell :

```
./manage.py shell
```

Administration des modèles

Django fournit une interface d'administration pour gérer les modèles.[3]

On peut créer, modifier, supprimer des instances.
On peut aussi personnaliser cette interface, graphiquement mais aussi au niveau des fonctionalités.

127.0.0.1:8000/admin

[3] Doc

Création d'un super-utilisateur

Pour accéder à l'interface d'admin, il faut des comptes.
On peut gérer cela dans l'interface mais il nous faut un premier utilisateur.

```
./manage.py createsuperuser
```

Administrer Publication

Dans admin.py, enregistrer le modèle Publication.

```
admin.site.register(Publication)
```

C'est ici qu'on enregistrera de nouvelles publications.

Un gabarit

Un gabarit est un fichier HTML avec des extensions.[4]

```
{% for publication in publications %}
{{ publication }}
{% endfor %}

{% if publication.editeur %}
{% else %}
{% endif %}

<a href="{% url 'accueil'%}"> Accueil </a>
```

[4] Doc

Gabarit pour les publications

A vos claviers !

Ecrire un gabarit, une vue et une règle de routage pour afficher la liste des publications.

Factoriser/diviser les gabarits

Généralement, dans un site web, il y a une en-tête, un pied de page, un menu communs à toute les pages.

On ne recopie pas le HTML complet !

Création de base.html : un HTML à trous.

```
{% block body %}{% endblock %}
```

Factoriser/diviser les gabarits

On étend le gabarit de base et on remplit les trous.

```
{% extends 'perso_app/base.html' %}
{% block body %}
    {# Mettre ici le code HTML #}
{% endblock %}
```

On peut aussi inclure un gabarit avec la balise `include`.

Un modèle et un formulaire

On peut décrire un formulaire par une classe avec des champs étendant `forms.Form` dans forms.py
Comme pour les modèles, il y une flopée de `forms.Field`
On peut aussi créer automatiquement un formulaire à partir d'un modèle.

```
class ContactForm(ModelForm):
    class Meta:
        model = Contact
        fields = ['nom', 'mail', 'raison', 'message']
```

Formulaire de contact

Ecrire le modèle correspondant
On pensera à actualiser urls.py, views.py et admin.py (et base.html)
Les gabarits savent générer automatiquement le HTML pour un formulaire.

```
class ContactForm(ModelForm):
    class Meta:
        model = Contact
        fields = ['nom', 'mail', 'raison', 'message']
```

Attention à ne pas oublier la protection CSRF[5]

[5] Doc

Traiter l'envoi d'un formulaire

Nouvelle règle de routage - nouvelle vue
On récupère les données et on enregiste le nouveau message.

Les tests c'est le bien

Pourquoi écrire des tests ?[6]

- Gagner du temps
- Vérifier plus rigoureusement, plus rapidement
- Gagner la confiance des autres
- Prévenir la casse

Que tester ? Les méthodes des modèles et les vues.
Ecrire les tests le plus tôt possible, parfois avant même d'écrire les fonctions.

[6] Doc

Ecrire un test pour Publication

On ajoute une méthode `recent` au modèle Publication et on la teste dans tests.py
Pour lancer les tests : `./manage.py test perso_ app`
Django lancera les fonctions commençant par 'test' dans les classes étendant 'TestCase'