



Formation Django

Votre formateur: Bastien

Merci à Véronique HELMER de @Pythagore FD

Plan de cours



1. Introduction et Installation

- Django
- Notre première application
- Le module d'administration

2. Un peu de théorie

- Static Rendering
- Les bases de données
- Multipage Rendering
- ORM

3. TP: Faunatrack

- Modèles et Vues
- Formulaires et Validations
- Authentification et permissions
- Signaux et logiques métiers
- Ecrire des test
- Mise en production
- Bonus ?



Django



Présentation

Framework Python Open Source développé en 2003

Caractéristiques

- Stabilité
- Modulaire (Orienté objet et nombreux package disponible)
- Facilité de prise en main
- Support *natif* Authentification, Permissions, Base de donnée, Cache, Internationalisation...
- ORM
- Architecture MVT

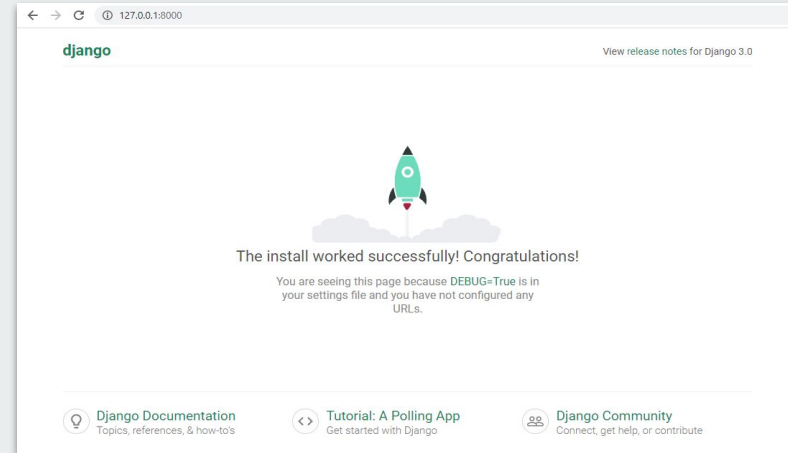
Installation

Créer un environnement virtuel

- `mkdir django_app`
- `cd django_app`
- `python3 -m venv venv`
- `source venv/bin/activate`

Lancer django

- `pip install --upgrade django django-extensions`
- `django-admin startproject pythagore`
- `cd pythagore`
- `python manage.py runserver`



pip freeze

```
asgiref==3.7.2
Django==5.0.1
django-extensions==3.2.3
sqlparse==0.4.4
typing_extensions==4.9.0
```

Notre première application “FaunaTrack”

➡ `python manage.py startapp faunatrack`

Commençons par créer une “vue” simple qui renvoie une réponse HTTP.

On importe le module `HttpResponse` et on crée une fonction dans `faunatrack/views.py`

Il faut ensuite ajouter cette vue dans notre array `'urlpatterns'`.

Une vue peut être une fonction ou une classe.

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'django_extensions', # Ajout de django-extensions  
    'faunatrack', # Ajout de faunatrack, notre première application  
]
```

```
from django.http import HttpResponse  
from django.shortcuts import render  
  
# Create your views here.  
  
def hello_world(request):  
    return HttpResponse(["Hello, World!"])
```

Notre premier modèle “Espèce” et les migrations

- ⇒ `python manage.py makemigrations`
- ⇒ `python manage.py migrate`
- ⇒ `python manage.py showmigrations`
- ⇒ `python manage.py migrate faunatrack 001` Revenir à une migration précédente (ici à la migration 001)
- ⇒ `python manage.py migrate faunatrack zero` Revenir à la migration initiale

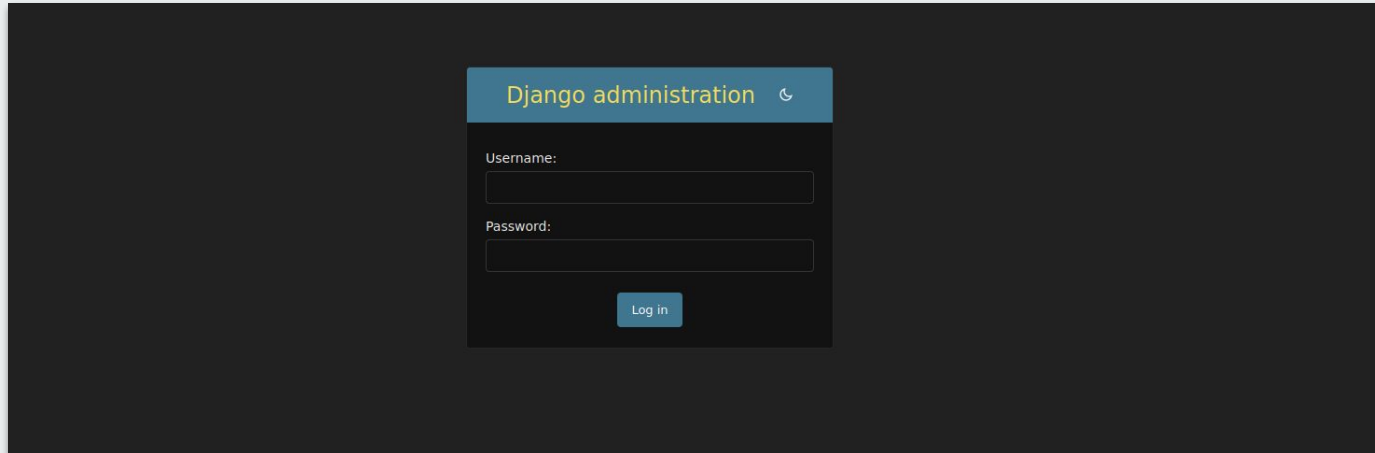
```
class Espece(models.Model):  
    nom_commun = models.CharField(max_length=250)
```

Le module “Admin”

⇒ `python manage.py createsuperuser`

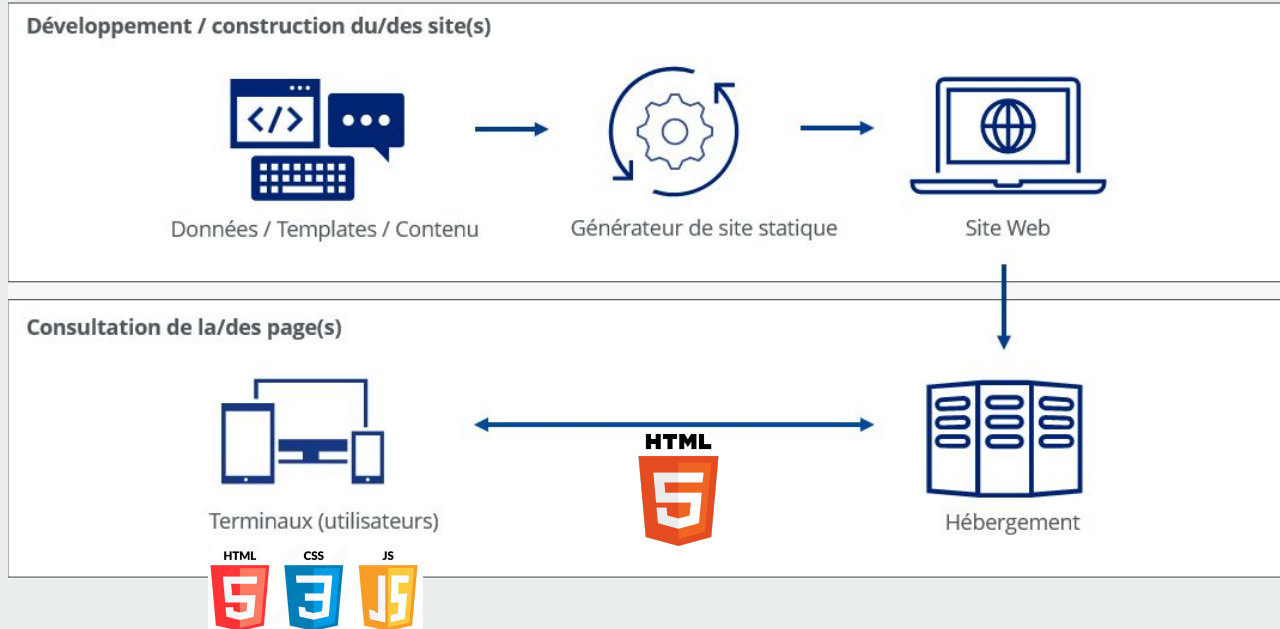
⇒ `/admin`

⇒ `python manage.py collectstatic` `STATIC_ROOT = BASE_DIR / 'static'`



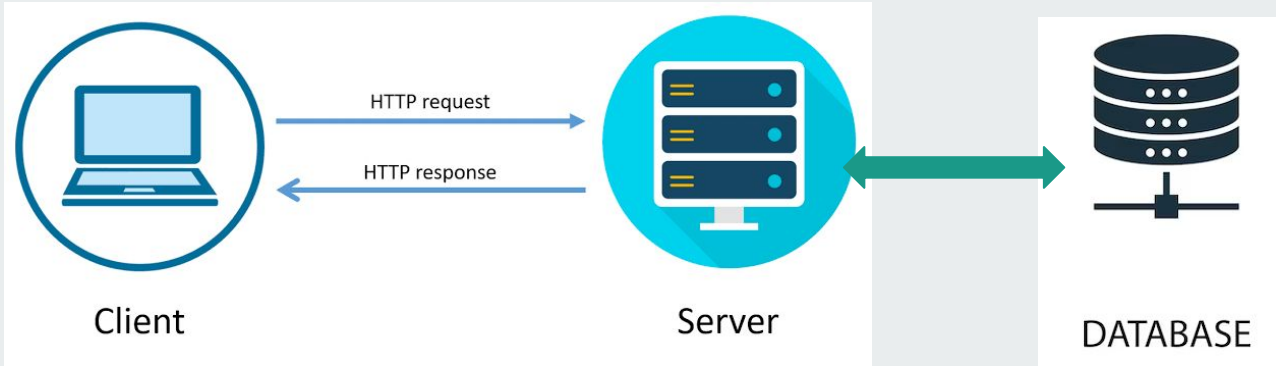
Un peu de théorie....

Static rendering



Un peu de théorie....

Base de données

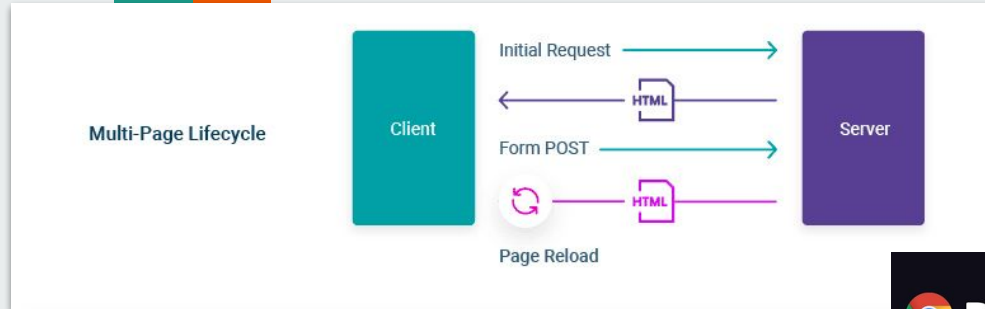


HTTP Status Codes

1XX	INFORMATIONAL
2XX	SUCCESS
3XX	REDIRECTION
4XX	CLIENT ERROR
5XX	SERVER ERROR

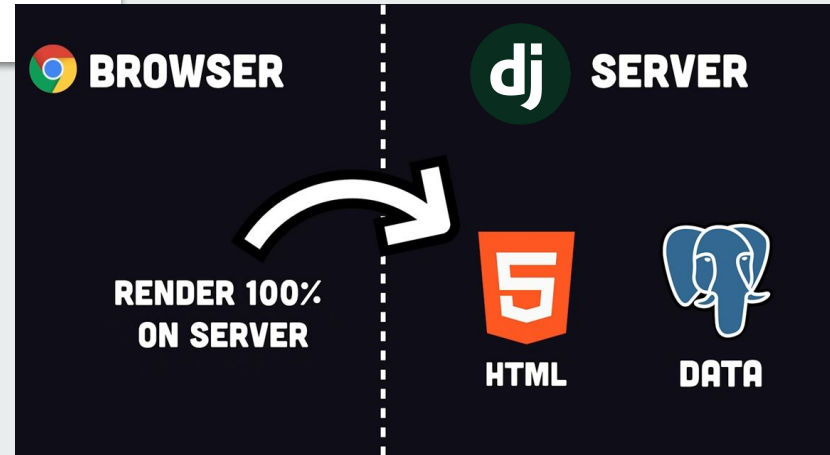
HTTP Verb	CRUD
POST	Create
GET	Read
PUT	Update/Replace
PATCH	Update/Modify
DELETE	Delete

Un peu de théorie....




Multipage rendering

Il existe énormément de façons différentes de faire du "rendering" web



Un peu de théorie....

Object Related Mapping (ORM)

 Guitare			
Type	Nom	Marque	nombre_de_corde
Cordes	Guitare	Fender	6

```
orchestre = Orchestre.objects.create(
    instruments = [
        Guitare.objects.create(nom="Guitare", type="Cordes", marque="Fender", nombre_de_corde=6),
        Piano.objects.create(nom="Piano", type="Touche", marque="Yamaha", nombre_de_touche=88),
    ]
)
orchestre.play() # Guitare Musique! Piano Musique!
```

```
class IntrumentDeMusique(models.Model):
    nom = models.CharField(max_length=250)
    type = models.CharField(max_length=250)
    marque = models.CharField(max_length=250)

    def play(self):
        return f"{self.nom} Musique!"

class Guitare(IntrumentDeMusique):
    nombre_de_corde = models.IntegerField()

class Piano(IntrumentDeMusique):
    nombre_de_touche = models.IntegerField()

class Orchestre(models.Model):
    instruments = models.ManyToManyField(IntrumentDeMusique)

    def play(self):
        for instrument in self.instruments.all():
            print(instrument.play())
```



TP : FaunaTrack

1. Analyser les pré-requis et créer nos modèles
2. Créer des vues et des templates
3. Mettre en place la validation de données
4. Améliorer nos formulaires - Les widgets
5. Ajouter une page de login et gérer les permissions
6. Appliquer des logiques métier et les tester
7. Module Admin et Import/Export (de fichier !)
8. Le module Email
9. Django Rest Framework
10. Mise en production

Field Name	Description
BigAutoField	A 64-bit integer that automatically increments. Suitable for IDs that need larger values.
BooleanField	A true/false field. The default form widget for this field is a <code>CheckboxInput</code> .
CharField	A field to store text-based values. Requires a <code>'max_length'</code> parameter.
DateField	Represents a date, without a time, represented in Python by a <code>'datetime.date'</code> instance.
DateTimeField	Used for date and time, represented in Python by a <code>'datetime.datetime'</code> instance.
DecimalField	A fixed-precision decimal number, represented in Python by a <code>'Decimal'</code> instance. Requires <code>'max_digits'</code> and <code>'decimal_places'</code> parameters.
DurationField	A field for storing periods of time, stored in Python as a <code>'datetime.timedelta'</code> instance.
EmailField	A <code>'CharField'</code> that checks that the value is a valid email address.
FileField	A file-upload field.
ImageField	Inherits all attributes and methods from <code>'FileField'</code> , but also validates that the uploaded object is a valid image.
IntegerField	An integer field. Values from -2147483648 to 2147483647 are safe in all databases supported by Django.
SlugField	A slug is a short label for something, containing only letters, numbers, underscores, or hyphens. They're generally used in URLs.
TextField	A large text field. The default form widget for this field is a <code>'Textarea'</code> .
TimeField	A time, represented in Python by a <code>'datetime.time'</code> instance.
URLField	A <code>'CharField'</code> for a URL, validated by <code>'URLValidator'</code> .
UUIDField	A field for storing universally unique identifiers. Uses Python's UUID class.

1. Analyser les pré-requis et créer nos modèles

Pré-requis :

<https://cloud.caliamis.net/remote.php/webdav/support/FaunaTrack%20-%20Pr%C3%A9Requis.pdf>

Documentations :

<https://docs.djangoproject.com/en/5.0/ref/models/fields/>
<https://docs.djangoproject.com/en/5.0/ref/models/fields/#arguments>

En cas d'ajout de **FileField** ou **ImageField** il faut configurer **settings.py** pour lui donner le chemin de nos fichiers statiques

```
STATICFILES_DIRS = [BASE_DIR / "faunatrack" / "static"]
```

Pour **ImageField**, il faut également installer la librairie pillow

```
python -m pip install Pillow
```

1. Analyser les pré-requis et créer nos modèles




Relationship Type	Description
ForeignKey	A one-to-many relationship. Requires two positional arguments: the class to which the model is related and the <code>'on_delete'</code> option.
ManyToManyField	Represents a many-to-many relationship. This field allows for the creation of a relationship that indicates that each instance of a model can be associated with multiple instances of another model, and vice versa.
OneToOneField	A one-to-one relationship. Essentially a <code>'ForeignKey'</code> with <code>'unique=True'</code> . This field is useful for creating a two-way link between two models, where each instance of a model is related to one and only one instance of another model.

<code>'on_delete'</code> Option	Description
CASCADE	Automatically deletes the object containing the ForeignKey when the referenced object is deleted.
PROTECT	Prevents deletion of the referenced object by raising a <code>'ProtectedError'</code> , a subclass of <code>'django.db.IntegrityError'</code> .
SET_NULL	Sets the ForeignKey null; requires <code>'null=True'</code> to be set on the ForeignKey.
SET_DEFAULT	Sets the ForeignKey to its default value; requires a default to be set on the ForeignKey.
SET()	Sets the ForeignKey to a passed value or a callable that returns a value.
DO_NOTHING	Takes no action. If your database backend enforces referential integrity, this will cause an IntegrityError unless you manually add a SQL <code>'ON DELETE'</code> constraint to the database field.
RESTRICT	Prevents the object from being deleted if there are any referenced objects still linked. This is similar to <code>'PROTECT'</code> but defers the checking of the constraint to the database level.

page 160 => ModelForm

2. Créer des vues et des templates

Page 171 => View avec ModelForm

- 
- Télécharger le fichier **base.html** et l'ajouter à notre vue **name="home"** qui pointe sur /
 - Ajouter des vues sous forme de classe à **faunatrack/views.py**
 - Ajouter des formulaires dans **faunatrack/forms.py**
 - Créer des urls pour ces vues dans **faunatrack/urls.py**
 - Ajouter **faunatrack/urls.py** dans notre fichier url de projet
 - Créer des templates dans **faunatrack/templates**
 - Ajouter les templates à notre **settings.py**

Ressources :

Utiliser un template base.html fournis avec **tailwind css** via un CDN:

<https://cloud.caliamis.net/apps/files/?dir=/support&openfile=47107>

Documentation tailwind:

<https://tailwindcss.com/docs/container>

Composant pré fait:

<https://tailwindui.com/components/preview>

3. Mettre en place la validation de données



- Se référer au pré-requis et définir la liste des données à valider
- Ajouter des validateurs à nos **models.Forms** dans **faunatrack/forms.py**
- Ajouter des validateurs sur les modèles si besoin
- Afficher les erreurs dans nos formulaires

4. Améliorer nos formulaires - Les widgets

page 137 à 146 du support

- Ajouter un widget **DateInput** pour les champs date
- Ajouter des **“help_text”** sur nos champs
- Surcharger **ModelForm** pour ajouter un **attribut “class”** à nos widget

5.1 Ajouter une page de login



- Ajouter des templates pour se connecter
- Ne pas oublier d'ajouter les templates "pythagore" dans **settings.py**
- Créer un **templatetags** pour ajouter des class à nos widgets directement dans le template
- Modifier **base.html** pour afficher un bouton se connecter et conditionner son affichage

5.2 Conditionner l'accès à nos vues

page 183 à 190 du support

- Les vues ne doivent pas être accessible sans être connecté
- On peut conditionner l'accès à notre vue avec **UserPassesTestMixin** ou via le modèle **Group** de l'interface Admin
- Rediriger l'utilisateur vers la page de login si il n'est pas connecté dans notre vue "**home**"

6. Appliquer des logiques métiers



- Utiliser l'héritage au niveau des permissions.
- Utiliser les signaux sur nos modèles
- Surcharger des fonctions pour appliquer des logiques métiers.
- Notifier nos utilisateurs avec le module message
- Ecrire des tests pour valider notre logique

7. Le module “Admin” et l’import export

- Connaître les attributs utiles : **list_display**, **list_filter**, **search_fields**, **ordering**, **list_editable**, **actions**
- Ajouter un modèle Inline
- Installer django import-export (pip install django-import-export)
- Définir des ressources et un ImportExportAdmin

TP : FaunaTrack

8. Module Email

def send_my_email(request):

```
    subject = 'Salut de Django'
    message = 'Ceci est un message test envoyé par Django.'
    email_from = 'your-email@example.com'
    recipient_list = ['recipient1@example.com', 'recipient2@example.com']
    send_mail(subject, message, email_from, recipient_list)
```

```
    return HttpResponse("Email envoyé !")
```

```
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.example.com'
EMAIL_PORT = 587
EMAIL_USE_TLS = True
EMAIL_HOST_USER = 'your-email@example.com'
EMAIL_HOST_PASSWORD = 'your-email-password'
```

- `EmailMessage => Email HTML`

9. Django Rest Framework

- Installer django rest framework (pip install djangorestframework)
- Mettre en place une vue simple et un serializer
- Utiliser les serializers pour modifier la réponse de nos vues
- Utiliser le router de DRF pour définir des ViewSets

TP : FaunaTrack

9.1 Django Rest Framework - Authentication



- Nous avons suivi cette documentation:
<https://www.django-rest-framework.org/api-guide/authentication/>
- Comprendre la différence entre BasicAuthentication / SessionAuthentication / TokenAuthentication
- Découvrir les JWT (ou JSON Web Tokens) et leur utilité
- Utiliser les packages comme dj-rest-auth et django-rest-framework-simplejwt pour ne pas réinventer la roue
- Appliquer des permissions à nos vues API

10. Mise en production

- Comprendre l'importance de **.env** et de l'utilisation des variables dans nos settings
- Ajouter un **requirements.txt** pour installer facilement nos dépendances
- Se connecter à une base de données distantes
- Utiliser Docker et docker compose pour monter facilement des conteneurs
- Utiliser gunicorn et un script bash pour appliquer automatiquement nos migrations et la collection des dossiers statiques
- Utiliser NGINX pour délivrer nos contenus statiques et gérer les redirections

Conclusion



Merci à tous !

J'espère que vous avez apprécié cette formation. Voici quelques liens utiles:

- GitHub Formateur <https://github.com/orgs/lance-kawa/repositories>
- Documentations Django <https://docs.djangoproject.com/fr/5.0/>
- Django ORM Query <https://docs.djangoproject.com/fr/5.0/topics/db/queries/>
- Django ORM API <https://docs.djangoproject.com/fr/5.0/ref/models/queriesets>
- Tailwind Docs <https://tailwindcss.com/docs/container>
- Lien du projet final : https://github.com/lance-kawa/django_formation
- Lien du README avec commandes django et docker : https://github.com/lance-kawa/django_formation/blob/master/README.md