Python Django

ISITECH 2025-2026 ESI 4D

C'est parti! →

Formation Django

Jour 1 : Fondamentaux et Architecture MVC/MVT

Développeurs 4ème année

Programme du Jour 1 - Matin

Partie 1

- Introduction et philosophie Django
- Architecture MVT vs MVC
- Installation et environnement

Partie 2

- Création de projet
- Structure d'application
- Système de routage
- Pattern URL Dispatcher

Introduction à Django

Philosophie et Contexte

Qu’est-ce que Django ?

Framework Web Python

- Créé en 2003, open source depuis 2005
- "The web framework for perfectionists with deadlines"
- Batteries included
- Don’t Repeat Yourself (DRY)
- Convention over Configuration

Utilisé par

- Instagram
- Pinterest
- Mozilla
- The Washington Post
- Disqus
- Bitbucket

Philosophie Django : Les Principes

Django vs FastAPI : Positionnement

FastAPI

- Microframework moderne
- Async natif
- API REST focus
- Flexible et minimaliste
- Type hints Python 3.6+
- Performance élevée
- Documentation automatique

Cas d'usage : APIs modernes, microservices

Django

- Framework complet
- Sync par défaut
- Applications web complètes
- Opinionated et structuré
- ORM puissant
- Interface admin intégrée
- Écosystème mature

Cas d'usage : Applications web, backoffice, CMS

Les "Batteries Included" de Django

Core

- ORM complet
- Migrations de BDD
- System de templates
- Routage URL
- Middleware système
- Gestion des formulaires

Sécurité

- Protection CSRF
- Protection XSS
- SQL Injection prevention
- Clickjacking protection
- SSL/HTTPS support
- Validation de données

Fonctionnalités

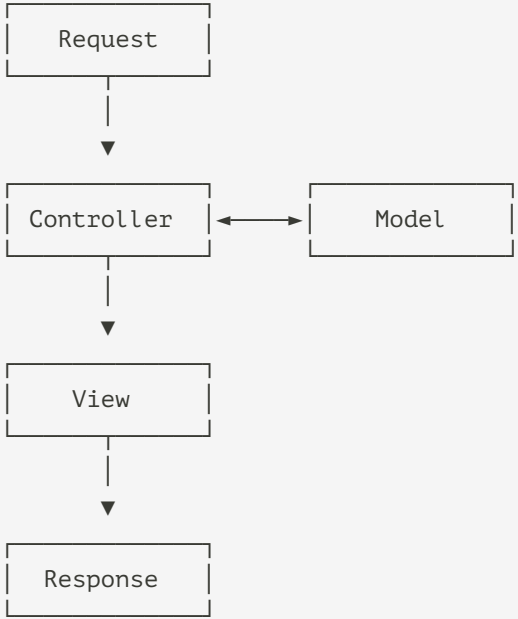
- Interface admin
- Authentification
- Sessions
- Cache framework
- Envoi d’emails
- Internationalisation
- Tests intégrés
- CLI puissant

Architecture MVT vs MVC

Comprendre le Pattern de Django

Le Pattern MVC Classique

Model-View-Controller



Rôles

Model : Données et logique métier

View : Présentation (templates)

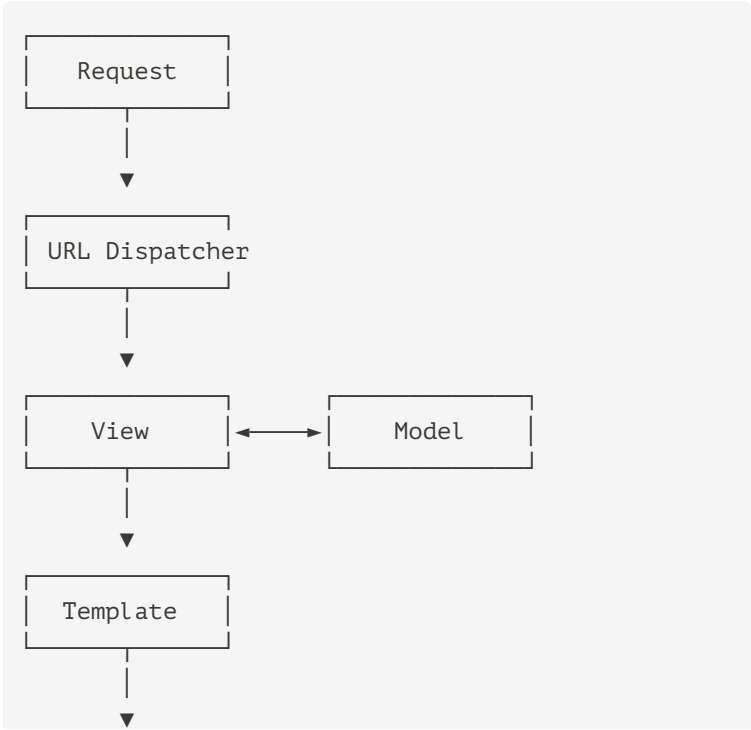
Controller : Logique de traitement

Flux :

1. Request arrive au Controller
2. Controller interroge le Model
3. Controller choisit la View
4. View génère le HTML
5. Response renvoyée

Le Pattern MVT de Django

Model-View-Template



Rôles dans Django

Model : Données et logique métier

View : Logique de traitement

Template : Présentation

URL Dispatcher : Routage

Différence clé : View Django = Controller MVC

Template Django = View MVC

Correspondance MVC et MVT

Concept	Framework MVC	Django MVT	Rôle
Données	**Model**	**Model**	Logique métier, BDD
Traitement	**Controller**	**View**	Logique application
Présentation	**View**	**Template**	Rendu HTML
Routage	Router	**URL Dispatcher**	Mapping URL

Django appelle "View" ce que les autres frameworks appellent "Controller", et appelle "Template" ce que les autres appellent "View".

Pattern MVT en Action : Exemple

Installation et Configuration

Mise en Place de l'Environnement

Prérequis et Installation

Prérequis

```
# Python 3.10+ recommandé
python --version

# pip à jour
pip install --upgrade pip

# Virtualenv
pip install virtualenv
```

Installation Django

```
# Créer environnement virtuel
python -m venv django_env

# Activer (Linux/Mac)
source django_env/bin/activate

# Activer (Windows)
django_env\Scripts\activate
```

Base de Données

Django supporte :

- **PostgreSQL** (production)
- **MySQL / MariaDB**
- **SQLite** (défaut, dev)
- **Oracle**

```
# PostgreSQL
pip install psycopg2-binary

# MySQL
pip install mysqlclient
```

Outils

- IDE : PyCharm, VS Code

Structure d’un Projet Django

```
myproject/                  # Racine du projet
├── manage.py               # CLI Django
├── myproject/              # Package Python du projet
│   ├── __init__.py
│   ├── settings.py         # Configuration
│   ├── urls.py             # URL dispatcher racine
│   ├── asgi.py             # Point d'entrée ASGI
│   └── wsgi.py             # Point d'entrée WSGI
└── myapp/                  # Une application
    ├── __init__.py
    ├── admin.py            # Config admin
    ├── apps.py             # Config app
    ├── migrations/         # Migrations BDD
    │   └── __init__.py
    ├── models.py           # Modèles
    ├── tests.py            # Tests
    ├── urls.py             # URLs de l'app
    └── views.py            # Vues
```

Projet vs Application

Projet Django

- **Conteneur global** du site web
- Configuration centralisée
- Gestion des applications
- URLs racine
- Settings communs

Un projet = un site web

Exemple : Site e-commerce

Application Django

- **Module fonctionnel** réutilisable
- Modèles spécifiques
- Vues spécifiques
- URLs spécifiques
- Réutilisable

Une app = une fonctionnalité

Exemples :

- App "products"
- App "cart"
- App "blog"
- App "users"

Créer un Projet Django

```
# Créer un nouveau projet
django-admin startproject myproject

# Structure créée
myproject/
├── manage.py
└── myproject/
    ├── __init__.py
    ├── settings.py
    ├── urls.py
    ├── asgi.py
    └── wsgi.py

# Se placer dans le projet
cd myproject

# Lancer le serveur
python manage.py runserver

# http://127.0.0.1:8000/
```

Le Fichier settings.py

```
# myproject/settings.py (extraits)

# Sécurité
SECRET_KEY = 'django-insecure-...'  # À changer
DEBUG = True  # False en production
ALLOWED_HOSTS = []  # Domaines autorisés

# Applications installées
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    # Vos apps ici
]

# Middleware
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
```

Configuration Base de Données

```python
# myproject/settings.py

# SQLite (par défaut)
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}

# PostgreSQL (production)
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'mydatabase',
        'USER': 'myuser',
        'PASSWORD': 'mypassword',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}
```

Créer une Application

```
# Créer une nouvelle application
python manage.py startapp books

# Structure créée
books/
├── __init__.py
├── admin.py
├── apps.py
├── migrations/
│   └── __init__.py
├── models.py
├── tests.py
└── views.py
```

```
# Enregistrer l'app dans settings.py
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
```

Système de Routage

URL Dispatcher et Pattern URL

Le Routage dans Django

Flux de Routage

```
1. Requête HTTP arrive
   ↓
2. URL Dispatcher analyse
   ↓
3. Recherche dans urlpatterns
   ↓
4. Match → Appel View
   ↓
5. View retourne HttpResponse
   ↓
6. Réponse au client
```

Concepts Clés

URLconf : Configuration des URLs

urlpatterns : Liste des routes

path() : Route simple

re_path() : Route regex

include() : Inclure URLs d'une app

name : Nommer une route

URL Dispatcher Pattern

URLs Racine du Projet

```python
# myproject/urls.py

from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    # Admin Django
    path('admin/', admin.site.urls),

    # Inclure les URLs de l'app books
    path('books/', include('books.urls')),

    # Autres apps
    path('api/', include('api.urls')),
    path('blog/', include('blog.urls')),
]
```

Le fichier urls.py racine délègue à chaque application

- `/admin/` → Interface admin
- `/books/` → URLs de books

URLs d’une Application

```python
# books/urls.py

from django.urls import path
from . import views

app_name = 'books'  # Namespace

urlpatterns = [
    # /books/
    path('', views.book_list, name='book_list'),

    # /books/5/
    path('<int:pk>/', views.book_detail, name='book_detail'),

    # /books/new/
    path('new/', views.book_create, name='book_create'),

    # /books/5/edit/
    path('<int:pk>/edit/', views.book_update, name='book_update'),

    # /books/5/delete/
    path('<int:pk>/delete/', views.book_delete, name='book_delete'),
```

Les Converters de Path

Converter	Description	Exemple	Capture
`str`	Chaîne (défaut)	`<str:name>`	Tout sauf `/`
`int`	Entier positif	`<int:id>`	0, 1, 42
`slug`	Slug	`<slug:post_slug>`	`my-post`
`uuid`	UUID	`<uuid:user_id>`	`075194d3-...`
`path`	Avec `/`	`<path:file_path>`	`docs/file.pdf`

```
# Exemples
urlpatterns = [
    path('user/<int:user_id>/', views.user_profile),
    path('post/<slug:slug>/', views.post_detail),
    path('files/<path:file_path>/', views.serve_file),
```

Pattern URL Dispatcher

Le Pattern

Objectif : Découpler URL de la logique

Principe : Mapper patterns vers handlers

Avantages :

- Séparation des préoccupations
- URLs lisibles et SEO-friendly
- Changement d'URL facile
- Reverse URL generation
- Testabilité

Implémentation Django

```
# Configuration déclarative
urlpatterns = [
    path('books/', views.book_list),
]

# Reverse lookup
from django.urls import reverse
url = reverse('books:book_detail',
              kwargs={'pk': 5})
# Génère: /books/5/
```

Pattern similaire : Front Controller

Nommer les Routes et Reverse Lookup

```python
# books/urls.py
from django.urls import path
from . import views

app_name = 'books'

urlpatterns = [
    path('', views.book_list, name='book_list'),
    path('<int:pk>/', views.book_detail, name='book_detail'),
]
```

```python
# Dans une view
from django.urls import reverse
from django.shortcuts import redirect

def some_view(request):
    # Générer une URL
    url = reverse('books:book_list')  # /books/

    # Avec paramètres
    url = reverse('books:book_detail', kwargs={'pk': 5})
```

Inclure les URLs d'une App

```python
# myproject/urls.py
from django.urls import path, include

urlpatterns = [
    path('books/', include('books.urls')),
]
```

```python
# books/urls.py
from django.urls import path
from . import views

app_name = 'books'

urlpatterns = [
    path('', views.book_list, name='list'),
    path('<int:pk>/', views.book_detail, name='detail'),
]
```

`include()` permet de modulariser, réutiliser, et changer le préfixe facilement

Exercice Pratique : Premier Routage

Objectif : Créer un projet avec routage

1. Créer projet `library`
2. Créer app `books`
3. Enregistrer dans `INSTALLED_APPS`
4. Créer views dans `books/views.py` :

```
from django.http import HttpResponse

def home(request):
    return HttpResponse("Bienvenue")

def book_list(request):
    return HttpResponse("Liste des livres")
```

5. Créer `books/urls.py` et définir routes
6. Inclure dans urls.py racine
7. Tester avec `runserver`

Pause - 15 minutes

On se retrouve pour continuer avec les modèles et l'ORM

Suite : Modèles et ORM Django

Formation Django

Modèles et ORM Django

Développeurs 4ème année

Modèles Django

L'ORM et le Pattern Active Record

Qu’est-ce qu’un Modèle Django ?

Définition

Un modèle Django est une classe Python qui représente une table de base de données

Responsabilités :

- Définir la structure des données
- Définir les relations entre tables
- Contenir la logique métier
- Valider les données
- Fournir l’API de requêtes

Exemple Simple

```python
from django.db import models

class Book(models.Model):
    title = models.CharField(max_length=200)
    author = models.CharField(max_length=100)
    published_date = models.DateField()
    price = models.DecimalField(
        max_digits=6,
        decimal_places=2
    )

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

Génère automatiquement la table SQL correspondante

Pattern Active Record

Le Pattern

Active Record : Un objet encapsule à la fois les données ET la logique de persistance

Principe :

- Une classe = une table
- Une instance = une ligne
- Les méthodes de l'objet savent se sauvegarder, se charger, se supprimer

Avantages :

- Simple et intuitif
- Moins de code

Django Implémentation

```python
# Créer et sauvegarder
book = Book(
    title="Python Guide",
    author="John Doe",
    published_date="2024-01-01",
    price=29.99
)
book.save()  # INSERT en BDD

# Modifier
book.price = 24.99
book.save()  # UPDATE

# Supprimer
book.delete()  # DELETE

# L'objet connaît la BDD
```

Types de Champs Django

Type	SQL	Utilisation	Exemple
`CharField`	VARCHAR	Texte court	`CharField(max_length=100)`
`TextField`	TEXT	Texte long	`TextField()`
`IntegerField`	INTEGER	Nombre entier	`IntegerField()`
`DecimalField`	DECIMAL	Prix, montants	`DecimalField(max_digits=6, decimal_places=2)`
`FloatField`	FLOAT	Nombre flottant	`FloatField()`
`BooleanField`	BOOLEAN	Vrai/Faux	`BooleanField(default=False)`

Options des Champs

Options Communes

```python
class Book(models.Model):
    # Obligatoire ou optionnel
    title = models.CharField(
        max_length=200
    )
    subtitle = models.CharField(
        max_length=200,
        blank=True,  # Formulaire
        null=True    # BDD
    )

    # Valeur par défaut
    status = models.CharField(
        max_length=20,
        default='draft'
    )

    # Unique
    isbn = models.CharField(
        max_length=13,
```

Autres Options

```python
class Book(models.Model):
    # Choices
    CATEGORY_CHOICES = [
        ('FIC', 'Fiction'),
        ('SCI', 'Science'),
        ('HIS', 'History'),
    ]
    category = models.CharField(
        max_length=3,
        choices=CATEGORY_CHOICES
    )

    # Index et aide
    title = models.CharField(
        max_length=200,
        db_index=True,
        help_text="Titre du livre"
    )

    # Nom de colonne custom
```

Méthodes Spéciales des Modèles

```python
from django.db import models

class Book(models.Model):
    title = models.CharField(max_length=200)
    author = models.CharField(max_length=100)
    published_date = models.DateField()
    price = models.DecimalField(max_digits=6, decimal_places=2)

    def __str__(self):
        """Représentation string de l'objet"""
        return self.title

    def __repr__(self):
        """Représentation détaillée pour debug"""
        return f"Book(title='{self.title}', author='{self.author}')"

    class Meta:
        """Métadonnées du modèle"""
        db_table = 'library_books'  # Nom de table custom
        ordering = ['-published_date']  # Ordre par défaut
        verbose_name = 'livre'
        verbose_name_plural = 'livres'
        unique_together = [['title', 'author']]  # Contrainte
```

Relations entre Modèles

ForeignKey, ManyToMany, OneToOne

ForeignKey : Relation Un-à-Plusieurs

Définition

```python
class Author(models.Model):
    name = models.CharField(max_length=100)
    birth_date = models.DateField()

    def __str__(self):
        return self.name

class Book(models.Model):
    title = models.CharField(max_length=200)
    author = models.ForeignKey(
        Author,
        on_delete=models.CASCADE,
        related_name='books'
    )
    published_date = models.DateField()

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

Utilisation

```python
# Créer un auteur
author = Author.objects.create(
    name="J.K. Rowling",
    birth_date="1965-07-31"
)

# Créer un livre lié
book = Book.objects.create(
    title="Harry Potter",
    author=author,
    published_date="1997-06-26"
)

# Accès direct
print(book.author.name)
# "J.K. Rowling"

# Relation inverse (related_name)
books = author.books.all()
# Tous les livres de l'auteur
```

Options de on_delete

Option	Comportement	Cas d'usage
`CASCADE`	Supprime les objets liés	Dépendance forte (Commande → Lignes)
`PROTECT`	Empêche la suppression	Protection (Catégorie → Produits)
`SET_NULL`	Met à NULL	Relation optionnelle
`SET_DEFAULT`	Met valeur par défaut	Fallback défini
`SET()`	Fonction custom	Logique complexe
`DO_NOTHING`	Ne fait rien	Gestion manuelle (rare)

```
class Book(models.Model):
    author = models.ForeignKey(
```

ManyToManyField : Plusieurs-à-Plusieurs

Définition

```python
class Author(models.Model):
    name = models.CharField(max_length=100)

    def __str__(self):
        return self.name

class Book(models.Model):
    title = models.CharField(max_length=200)
    authors = models.ManyToManyField(
        Author,
        related_name='books'
    )

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

Un livre peut avoir plusieurs auteurs

Un auteur peut avoir plusieurs livres

Utilisation

```python
# Créer des auteurs
author1 = Author.objects.create(
    name="Author One"
)
author2 = Author.objects.create(
    name="Author Two"
)

# Créer un livre
book = Book.objects.create(
    title="Collaborative Book"
)

# Ajouter des auteurs
book.authors.add(author1, author2)

# Accès
for author in book.authors.all():
    print(author.name)
```

ManyToMany avec Table Intermédiaire

```python
class Author(models.Model):
    name = models.CharField(max_length=100)

class Book(models.Model):
    title = models.CharField(max_length=200)
    authors = models.ManyToManyField(
        Author,
        through='BookAuthor',
        related_name='books'
    )

class BookAuthor(models.Model):
    """Table intermédiaire avec données supplémentaires"""
    book = models.ForeignKey(Book, on_delete=models.CASCADE)
    author = models.ForeignKey(Author, on_delete=models.CASCADE)
    role = models.CharField(max_length=50)  # "main", "co-author"
    order = models.IntegerField()  # Ordre d'apparition

    class Meta:
        unique_together = [['book', 'author']]
        ordering = ['order']
```

OneToOneField : Relation Un-à-Un

Définition

```python
from django.contrib.auth.models import User

class UserProfile(models.Model):
    """Extension du modèle User"""
    user = models.OneToOneField(
        User,
        on_delete=models.CASCADE,
        related_name='profile'
    )
    bio = models.TextField(blank=True)
    birth_date = models.DateField(
        null=True,
        blank=True
    )
    avatar = models.ImageField(
        upload_to='avatars/',
        null=True,
        blank=True
    )
```

Utilisation

```python
# Créer un utilisateur
user = User.objects.create_user(
    username='john',
    email='john@example.com'
)

# Créer son profil
profile = UserProfile.objects.create(
    user=user,
    bio="Développeur Python",
    birth_date="1990-01-01"
)

# Accès direct
print(user.profile.bio)
# "Développeur Python"

# Inverse
print(profile.user.username)
# "john"
```

L'API de Requêtes Django

QuerySets et Managers

Manager et QuerySets

Manager

Le **Manager** est l'interface pour faire des requêtes

Par défaut : `objects`

```python
class Book(models.Model):
    title = models.CharField(max_length=200)
    # ...

    # Manager par défaut
    objects = models.Manager()
```

Utilisé pour :

- Créer des objets
- Récupérer des objets
- Filtrer, trier, agréger

QuerySet

Le **QuerySet** représente une collection d'objets

Lazy : La requête SQL n'est exécutée que quand nécessaire

```python
# Crée un QuerySet (pas de SQL)
qs = Book.objects.filter(price__lt=30)

# Toujours pas de SQL
qs = qs.filter(title__icontains='django')

# ICI le SQL est exécuté
for book in qs:
    print(book.title)
```

Chaînable : On peut enchaîner les méthodes

```python
books = Book.objects.filter(
```

Opérations CRUD de Base

Create - Créer

```python
# Méthode 1 : create()
book = Book.objects.create(
    title="Django Guide",
    author="John Doe",
    price=29.99
)

# Méthode 2 : save()
book = Book(
    title="Django Guide",
    author="John Doe",
    price=29.99
)
book.save()

# Bulk create (optimisé)
books = [
    Book(title="Book 1", price=10),
    Book(title="Book 2", price=20),
]
```

Read - Lire

```python
# Récupérer tous
all_books = Book.objects.all()

# Récupérer un seul (erreur si 0 ou >1)
book = Book.objects.get(id=1)
book = Book.objects.get(isbn='123456')

# Récupérer premier/dernier
first = Book.objects.first()
last = Book.objects.last()

# Filtrer
cheap = Book.objects.filter(price__lt=30)
django_books = Book.objects.filter(
    title__icontains='django'
)

# Exclure
books = Book.objects.exclude(
    category='Fiction'
```

Lookups : Filtres Avancés

Lookup	SQL	Exemple	Résultat
`exact`	`=`	`title__exact='Python'`	Égalité stricte
`iexact`	`ILIKE`	`title__iexact='python'`	Égalité insensible casse
`contains`	`LIKE %x%`	`title__contains='django'`	Contient
`icontains`	`ILIKE %x%`	`title__icontains='Django'`	Contient (insensible)
`startswith`	`LIKE x%`	`title__startswith='Python'`	Commence par
`endswith`	`LIKE %x`	`title__endswith='Guide'`	Finit par
`gt`	`>`	`price__gt=30`	Supérieur

Requêtes sur Relations

```python
class Author(models.Model):
    name = models.CharField(max_length=100)

class Book(models.Model):
    title = models.CharField(max_length=200)
    author = models.ForeignKey(Author, on_delete=models.CASCADE, related_name='books')
    price = models.DecimalField(max_digits=6, decimal_places=2)
```

Traverser les Relations

```python
# Livres dont l'auteur s'appelle "Doe"
books = Book.objects.filter(
    author__name__contains='Doe'
)

# Auteurs qui ont écrit des livres chers
authors = Author.objects.filter(
    books__price__gte=50
)

# Plusieurs niveaux
# (si Book avait Publisher, etc.)
```

Relation Inverse

```python
# Depuis un auteur, ses livres
author = Author.objects.get(id=1)
books = author.books.all()

# Filtrer les livres de l'auteur
cheap_books = author.books.filter(
    price__lt=30
)

# Compter
count = author.books.count()
```

Tri et Limitation

```python
# Trier par un champ (ordre croissant)
books = Book.objects.order_by('title')

# Ordre décroissant (-)
books = Book.objects.order_by('-price')

# Trier par plusieurs champs
books = Book.objects.order_by('-published_date', 'title')

# Trier par relation
books = Book.objects.order_by('author__name')

# Inverser l'ordre
books = Book.objects.order_by('title').reverse()

# Limiter (LIMIT SQL)
first_10 = Book.objects.all()[:10]

# Offset et limit (OFFSET/LIMIT)
books_11_to_20 = Book.objects.all()[10:20]

# Slicing déclenche l'exécution !
book = Book.objects.all()[0]  # Premier livre
```

Q Objects : Requêtes Complexes

```python
from django.db.models import Q

# OR avec Q
books = Book.objects.filter(
    Q(title__icontains='python') | Q(title__icontains='django')
)
# SQL: WHERE title ILIKE '%python%' OR title ILIKE '%django%'

# AND (implicite avec filter, explicite avec Q)
books = Book.objects.filter(
    Q(price__gte=20) & Q(price__lte=50)
)

# NOT avec ~
books = Book.objects.filter(
    ~Q(category='Fiction')
)

# Combinaisons complexes
books = Book.objects.filter(
    (Q(title__icontains='python') | Q(title__icontains='django')) &
    Q(price__lt=50) &
    Q(author__name='John Doe')
)
```

Exercice Pratique : Modèles et Requêtes

Objectif : Créer des modèles avec relations et faire des requêtes

1. Dans l'app `books` , créer les modèles :
   - `Author` (name, birth_date, nationality)
   - `Book` (title, isbn, published_date, price, author FK)
2. Créer et appliquer les migrations :

```
python manage.py makemigrations
python manage.py migrate
```

3. Utiliser le shell Django :

```
python manage.py shell
```

4. Créer des données et faire des requêtes :
   - Créer 2-3 auteurs
   - Créer 5-6 livres liés aux auteurs

Pause Déjeuner

Rendez-vous à 13h30 pour l’après-midi

Cet après-midi : Migrations, Admin, et Vues

Formation Django

Jour 2 : Formulaires, Authentification et Architecture

Développeurs 4ème année

Programme du Jour 2

Matin

- Système de formulaires Django
- ModelForms et validation
- Authentification Django
- Système de permissions

Après-midi

- Architecture et organisation
- Configuration multi-environnement
- Messages et notifications
- Gestion des fichiers
- Projet fil rouge - intégration

Système de Formulaires Django

Pattern Form Object et Validation

Pourquoi un Système de Formulaires ?

Problèmes du HTML brut

- Validation côté serveur répétitive
- Code de validation éparpillé
- Gestion des erreurs manuelle
- Sécurité (CSRF, XSS)
- Réaffichage avec erreurs complexe
- Pas de réutilisation

Exemple : formulaire de création de livre

Validation à coder manuellement :

- ISBN valide
- Année entre 1450 et aujourd'hui
- Auteur existe en base

Solution Django Forms

- Validation centralisée et réutilisable
- Protection CSRF automatique
- Génération HTML automatique
- Gestion des erreurs unifiée
- Réaffichage avec données
- Widgets personnalisables
- Intégration avec modèles

Pattern Form Object :

- Encapsule les règles de validation
- Sépare validation et présentation
- Réutilisable entre vues

Anatomie d'un Formulaire Django

```python
from django import forms

class BookForm(forms.Form):
    """Formulaire de création/modification de livre"""

    title = forms.CharField(
        max_length=200,
        label="Titre",
        help_text="Titre complet de l'ouvrage"
    )

    isbn = forms.CharField(
        max_length=13,
        label="ISBN-13",
        validators=[validate_isbn]
    )

    publication_year = forms.IntegerField(
        label="Année de publication",
        min_value=1450,
        max_value=2025
    )
```

Cycle de Vie d'un Formulaire

Vue avec Formulaire

```python
from django.shortcuts import render, redirect

def create_book(request):
    if request.method == 'POST':
        # Formulaire soumis
        form = BookForm(request.POST)
        if form.is_valid():
            # Données valides
            data = form.cleaned_data
            # Créer le livre
            book = Book.objects.create(
                title=data['title'],
                isbn=data['isbn'],
                publication_year=data['publication_
                author=data['author'],
                copies_total=data['copies_total'],
                copies_available=data['copies_total
            )
            return redirect('book_detail', pk=book.
    else:
```

Flux

```
1. GET /books/create/
   └─> Créer form vide
   └─> Rendre template

2. Utilisateur remplit

3. POST /books/create/
   └─> Créer form avec POST data
   └─> form.is_valid()
       ├─> True
       │   └─> cleaned_data disponible
       │   └─> Sauvegarder
       │   └─> Rediriger
       └─> False
           └─> Erreurs dans form.errors
           └─> Ré-afficher formulaire
```

Pattern : POST-Redirect-GET

Rendu du Formulaire dans le Template

Rendu Automatique

```
<form method="post">
    {% csrf_token %}

    <!-- Tout le formulaire -->
    {{ form.as_p }}

    <button type="submit">Créer</button>
</form>

<!-- Avec erreurs non-field -->
{% if form.non_field_errors %}
<div class="errors">
    {{ form.non_field_errors }}
</div>
{% endif %}

<!-- Variantes de rendu -->
{{ form.as_table }}
{{ form.as_ul }}
{{ form.as_div }}
```

Rendu Manuel (contrôle total)

```
<form method="post">
    {% csrf_token %}

    {% for field in form %}
    <div class="field">
        {{ field.label_tag }}
        {{ field }}

        {% if field.errors %}
        <div class="errors">
            {{ field.errors }}
        </div>
        {% endif %}

        {% if field.help_text %}
        <small>{{ field.help_text }}</small>
        {% endif %}
    </div>
```

Validation au Niveau Champ

```python
from django import forms
from datetime import date

class BookForm(forms.Form):
    title = forms.CharField(max_length=200)
    isbn = forms.CharField(max_length=13)
    publication_year = forms.IntegerField()

    def clean_isbn(self):
        """Validation personnalisée de l'ISBN"""
        isbn = self.cleaned_data['isbn']

        # Retirer les tirets
        isbn = isbn.replace('-', '')

        # Vérifier la longueur
        if len(isbn) != 13:
            raise forms.ValidationError(
                "L'ISBN doit contenir 13 chiffres"
            )

        # Vérifier que ce sont des chiffres
        if not isbn.isdigit():
```

Validation au Niveau Formulaire

```python
class LoanForm(forms.Form):
    """Formulaire de création d'emprunt"""
    book = forms.ModelChoiceField(
        queryset=Book.objects.filter(copies_available__gt=0),
        label="Livre"
    )
    borrower_name = forms.CharField(max_length=100)
    borrower_email = forms.EmailField()
    card_number = forms.CharField(max_length=20)

    def clean(self):
        """Validation globale du formulaire"""
        cleaned_data = super().clean()

        book = cleaned_data.get('book')
        card_number = cleaned_data.get('card_number')

        if book and card_number:
            # Vérifier que l'usager n'a pas déjà 5 emprunts
            active_loans = Loan.objects.filter(
                card_number=card_number,
                status='active'
            ).count()
```

ModelForms

Génération Automatique depuis les Modèles

De Form à ModelForm

Form Classique (manuel)

```python
class BookForm(forms.Form):
    title = forms.CharField(max_length=200)
    isbn = forms.CharField(max_length=13)
    publication_year = forms.IntegerField()
    author = forms.ModelChoiceField(
        queryset=Author.objects.all()
    )
    copies_total = forms.IntegerField()
    description = forms.CharField(
        widget=forms.Textarea
    )
    category = forms.ChoiceField(
        choices=CATEGORY_CHOICES
    )
    language = forms.CharField(max_length=50)
    pages = forms.IntegerField()
    publisher = forms.CharField(max_length=100)
```

Beaucoup de duplication avec le modèle

ModelForm (automatique)

```python
from django import forms
from .models import Book

class BookForm(forms.ModelForm):
    class Meta:
        model = Book
        fields = [
            'title',
            'isbn',
            'publication_year',
            'author',
            'copies_total',
            'description',
            'category',
            'language',
            'pages',
            'publisher'
        ]
        # Ou tout inclure
        # fields = '__all__'
```

Personnalisation de ModelForm

```python
from django import forms
from .models import Book

class BookForm(forms.ModelForm):
    class Meta:
        model = Book
        fields = '__all__'

        # Personnaliser les labels
        labels = {
            'isbn': 'Numéro ISBN',
            'publication_year': 'Année de parution',
        }

        # Textes d'aide
        help_texts = {
            'isbn': 'Format ISBN-13 avec ou sans tirets',
            'copies_total': 'Nombre total d\'exemplaires possédés',
        }

        # Messages d'erreur personnalisés
        error_messages = {
```

Sauvegarder un ModelForm

Sauvegarde Simple

```
def create_book(request):
    if request.method == 'POST':
        form = BookForm(request.POST, request.FILES
        if form.is_valid():
            # Sauvegarde directe
            book = form.save()

            return redirect('book_detail', pk=book.
    else:
        form = BookForm()

    return render(request, 'create_book.html', {
        'form': form
    })
```

form.save() crée et sauvegarde l'objet

Sauvegarde avec Modification

```
def create_book(request):
    if request.method == 'POST':
        form = BookForm(request.POST, request.FILES
        if form.is_valid():
            # Ne pas sauvegarder encore
            book = form.save(commit=False)

            # Modifier avant sauvegarde
            book.copies_available = book.copies_tot
            book.added_by = request.user

            # Maintenant sauvegarder
            book.save()

            return redirect('book_detail', pk=book.
    else:
        form = BookForm()
```

Validation dans ModelForm

```python
from django import forms
from .models import Book
from datetime import date

class BookForm(forms.ModelForm):
    class Meta:
        model = Book
        fields = '__all__'

    def clean_isbn(self):
        """Validation de l'ISBN"""
        isbn = self.cleaned_data['isbn']
        isbn_clean = isbn.replace('-', '')

        if not isbn_clean.isdigit() or len(isbn_clean) != 13:
            raise forms.ValidationError(
                "ISBN invalide : doit contenir 13 chiffres"
            )

        # Vérifier l'unicité sauf si on édite
        qs = Book.objects.filter(isbn=isbn_clean)
        if self.instance.pk:
            # Édition : exclure le livre actuel
```

Système d'Authentification Django

Users, Login, Permissions

Architecture du Système Auth

Composants

Modèles

- User : utilisateur du système
- Group : groupe d'utilisateurs
- Permission : permission granulaire

Vues intégrées

- LoginView, LogoutView
- PasswordChangeView
- PasswordResetView

Middleware

- AuthenticationMiddleware

Modèle User

Champs de base :

- username (unique)
- email
- password (hashé)
- first_name, last_name
- is_staff (accès admin)
- is_active (compte actif)
- is_superuser (tous droits)
- last_login
- date_joined

Méthodes utiles :

Configuration de l'Authentification

```python
# settings.py

INSTALLED_APPS = [
    'django.contrib.auth',          # Système d'auth
    'django.contrib.contenttypes', # Nécessaire pour auth
    # ...
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',  # Important
    'django.contrib.messages.middleware.MessageMiddleware',
    # ...
]

# Configuration auth
AUTH_USER_MODEL = 'auth.User'  # Modèle User par défaut

LOGIN_URL = '/accounts/login/'  # URL de login
LOGIN_REDIRECT_URL = '/'        # Redirection après login
```

Vues de Login et Logout

URLs

```python
# urls.py
from django.urls import path
from django.contrib.auth import views as auth_views

urlpatterns = [
    path('login/',
         auth_views.LoginView.as_view(
             template_name='login.html'
         ),
         name='login'),

    path('logout/',
         auth_views.LogoutView.as_view(),
         name='logout'),
]
```

Template login.html

```html
<h2>Connexion</h2>
```

Vue Personnalisée

```python
from django.contrib.auth import authenticate, login
from django.shortcuts import render, redirect

def custom_login(request):
    if request.method == 'POST':
        username = request.POST['username']
        password = request.POST['password']

        # Authentifier
        user = authenticate(
            request,
            username=username,
            password=password
        )

        if user is not None:
            # Login : créer la session
            login(request, user)

            # Rediriger
```

Restriction d'Accès avec Décorateurs

```python
from django.contrib.auth.decorators import login_required, permission_required
from django.shortcuts import render

@login_required
def my_loans(request):
    """Vue accessible uniquement aux utilisateurs authentifiés"""
    loans = Loan.objects.filter(
        borrower_email=request.user.email
    )
    return render(request, 'my_loans.html', {
        'loans': loans
    })

@login_required(login_url='/custom-login/')
def profile(request):
    """Login URL personnalisée"""
    return render(request, 'profile.html')

@permission_required('library.add_book')
def create_book(request):
    """Uniquement avec permission add_book"""
    # ...
    pass
```

Restriction d'Accès dans les CBV

```python
from django.contrib.auth.mixins import LoginRequiredMixin, PermissionRequiredMixin, UserPassesTestMixin
from django.views.generic import ListView, CreateView, UpdateView

class MyLoansView(LoginRequiredMixin, ListView):
    """Liste des emprunts de l'utilisateur"""
    model = Loan
    template_name = 'my_loans.html'
    login_url = '/accounts/login/'

    def get_queryset(self):
        return Loan.objects.filter(
            borrower_email=self.request.user.email
        )

class CreateBookView(PermissionRequiredMixin, CreateView):
    """Création de livre avec permission"""
    model = Book
    fields = '__all__'
    permission_required = 'library.add_book'
    raise_exception = True  # 403 au lieu de redirect

class EditBookView(PermissionRequiredMixin, UpdateView):
    """Édition avec permissions multiples"""
```

Utilisateur dans les Templates

```
<!-- Vérifier si authentifié -->
{% if user.is_authenticated %}
    <p>Bonjour {{ user.username }} !</p>
    <a href="{% url 'logout' %}">Déconnexion</a>
{% else %}
    <a href="{% url 'login' %}">Connexion</a>
{% endif %}

<!-- Informations utilisateur -->
{% if user.is_authenticated %}
    <div class="user-info">
        <p>Email : {{ user.email }}</p>
        <p>Nom : {{ user.get_full_name }}</p>
        <p>Inscrit le : {{ user.date_joined|date:"d/m/Y" }}</p>
    </div>
{% endif %}

<!-- Vérifier les permissions -->
{% if perms.library.add_book %}
    <a href="{% url 'create_book' %}">Ajouter un livre</a>
{% endif %}

{% if perms.library.change_book %}
```

Système de Permissions

Groupes et Permissions Personnalisées

Permissions par Défaut

Génération Automatique

Pour chaque modèle, Django crée :

- **add_modelname** : créer une instance
- **change_modelname** : modifier
- **delete_modelname** : supprimer
- **view_modelname** : voir (Django 2.1+)

Exemple pour Book :

- library.add_book
- library.change_book
- library.delete_book
- library.view_book

Attribution

Via l'admin :

- User admin : section Permissions
- Cocher les permissions individuelles
- Ou donner is_staff / is_superuser

Par code :

```
from django.contrib.auth.models import Permission
from django.contrib.contenttypes.models import Cont

# Récupérer une permission
content_type = ContentType.objects.get_for_model(Bo
permission = Permission.objects.get(
    codename='add_book',
    content_type=content_type
)
```

Permissions Personnalisées

```python
# models.py
from django.db import models

class Book(models.Model):
    title = models.CharField(max_length=200)
    # ... autres champs

    class Meta:
        permissions = [
            ('can_reserve_book', 'Peut réserver un livre'),
            ('can_extend_loan', 'Peut prolonger un emprunt'),
            ('can_view_statistics', 'Peut voir les statistiques'),
        ]

class Loan(models.Model):
    # ... champs

    class Meta:
        permissions = [
            ('can_force_return', 'Peut forcer le retour d\'un livre'),
            ('can_waive_penalty', 'Peut annuler les pénalités'),
        ]
```

Groupes et Rôles

```python
# Créer des groupes et assigner des permissions
from django.contrib.auth.models import Group, Permission

# Groupe Bibliothécaires
librarians = Group.objects.create(name='Librarians')
librarians.permissions.add(
    Permission.objects.get(codename='add_book'),
    Permission.objects.get(codename='change_book'),
    Permission.objects.get(codename='delete_book'),
    Permission.objects.get(codename='can_force_return'),
    Permission.objects.get(codename='can_waive_penalty'),
)

# Groupe Usagers
readers = Group.objects.create(name='Readers')
readers.permissions.add(
    Permission.objects.get(codename='view_book'),
    Permission.objects.get(codename='can_reserve_book'),
)

# Ajouter un utilisateur à un groupe
user.groups.add(librarians)
```

Architecture et Organisation

Applications, Services, Configuration

Découpage en Applications Django

Principes

Une app = un domaine fonctionnel

Cohérence interne forte, couplage externe faible

Exemples de découpage :

Mauvais :

- models
- views
- templates

Bon :

- books (gestion du catalogue)

Structure d'une App

```
library/
├── __init__.py
├── admin.py
├── apps.py
├── models.py
├── views.py
├── urls.py
├── forms.py
├── managers.py
├── services.py
├── tests/
│   ├── __init__.py
│   ├── test_models.py
│   ├── test_views.py
│   └── test_services.py
├── templates/
│   └── library/
│       ├── book_list.html
│       └── book_detail.html
└── static/
```

Pattern Service Layer

Problème

Vue trop chargée :

```python
@login_required
def create_loan(request):
    if request.method == 'POST':
        form = LoanForm(request.POST)
        if form.is_valid():
            book = form.cleaned_data['book']

            # Logique métier dans la vue
            if book.copies_available <= 0:
                messages.error(request, 'Livre indi
                return render(request, 'create_loan

            if Loan.objects.filter(card_number=form
                messages.error(request, 'Limite att
                return render(request, 'create_loan

            loan = form.save(commit=False)
```

Solution : Service

```python
# services.py
from django.db import transaction
from datetime import timedelta
from django.utils import timezone

class LoanService:
    @staticmethod
    @transaction.atomic
    def create_loan(book, borrower_name, borrower_e
        """Créer un emprunt avec toute la logique m

        # Vérifications métier
        if book.copies_available <= 0:
            raise ValueError("Livre indisponible")

        active_loans = Loan.objects.filter(
            card_number=card_number,
            status='active'
        ).count()
```

Vue Simplifiée avec Service

```python
# views.py
from django.contrib.auth.decorators import login_required
from django.shortcuts import render, redirect
from django.contrib import messages
from .forms import LoanForm
from .services import LoanService

@login_required
def create_loan(request):
    """Vue légère : gère la requête, délègue au service"""
    if request.method == 'POST':
        form = LoanForm(request.POST)
        if form.is_valid():
            try:
                # Déléguer au service
                loan = LoanService.create_loan(
                    book=form.cleaned_data['book'],
                    borrower_name=form.cleaned_data['borrower_name'],
                    borrower_email=form.cleaned_data['borrower_email'],
                    card_number=form.cleaned_data['card_number']
                )

                messages.success(request, f'Emprunt créé : {loan.book.title}')
```

Managers Personnalisés

```python
# managers.py
from django.db import models

class AvailableBookManager(models.Manager):
    """Manager pour les livres disponibles"""
    def get_queryset(self):
        return super().get_queryset().filter(copies_available__gt=0)

class BookQuerySet(models.QuerySet):
    """QuerySet personnalisé avec méthodes chainables"""

    def available(self):
        return self.filter(copies_available__gt=0)

    def by_category(self, category):
        return self.filter(category=category)

    def published_after(self, year):
        return self.filter(publication_year__gte=year)

    def search(self, query):
        return self.filter(
            models.Q(title__icontains=query) |
```

Configuration Multi-Environnement

Dev, Staging, Production

Problème de Configuration Unique

settings.py Standard

```
# Dangereux et inflexible
SECRET_KEY = 'django-insecure-hardcoded-secret'
DEBUG = True  # Oublié en production !

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}

ALLOWED_HOSTS = []  # Pas de restriction

EMAIL_BACKEND = 'django.core.mail.backends.console.

STATIC_ROOT = BASE_DIR / 'staticfiles'
MEDIA_ROOT = BASE_DIR / 'media'

# Même config pour tous les environnements
```

Solution : Settings Multiples

```
myproject/
├── settings/
│   ├── __init__.py
│   ├── base.py          # Commun
│   ├── dev.py           # Développement
│   ├── staging.py       # Pré-production
│   └── production.py    # Production
└── ...
```

base.py : configuration commune

dev.py :

- DEBUG=True
- SQLite

Organisation des Settings

```python
# settings/base.py - Configuration commune
from pathlib import Path

BASE_DIR = Path(__file__).resolve().parent.parent.parent

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'library',
    'loans',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
```

Settings Développement

```python
# settings/dev.py
from .base import *

# Debug activé
DEBUG = True

# Secret key pour dev (pas grave si exposée)
SECRET_KEY = 'dev-secret-key-not-for-production'

# Pas de restriction d'hôtes en dev
ALLOWED_HOSTS = ['*']

# SQLite pour simplicité
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}

# Email dans la console
EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'
```

Settings Production

```python
# settings/production.py
from .base import *
import os

# Debug désactivé
DEBUG = False

# Secret key depuis variable d'environnement
SECRET_KEY = os.environ['DJANGO_SECRET_KEY']

# Hôtes autorisés stricts
ALLOWED_HOSTS = os.environ.get('DJANGO_ALLOWED_HOSTS', '').split(',')

# PostgreSQL depuis variables d'environnement
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': os.environ['DB_NAME'],
        'USER': os.environ['DB_USER'],
        'PASSWORD': os.environ['DB_PASSWORD'],
        'HOST': os.environ['DB_HOST'],
        'PORT': os.environ.get('DB_PORT', '5432'),
    }
```

Variables d'Environnement

Avec python-decouple

```
# Installation
# pip install python-decouple

# settings/production.py
from decouple import config, Csv

SECRET_KEY = config('SECRET_KEY')

DEBUG = config('DEBUG', default=False, cast=bool)

ALLOWED_HOSTS = config(
    'ALLOWED_HOSTS',
    cast=Csv()
)

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': config('DB_NAME'),
        'USER': config('DB_USER'),
```

Fichier .env

```
# .env (jamais commité dans Git)
SECRET_KEY=votre-cle-secrete-super-longue
DEBUG=False
ALLOWED_HOSTS=example.com,www.example.com

DB_NAME=myproject_db
DB_USER=myproject_user
DB_PASSWORD=mot-de-passe-complexe
DB_HOST=localhost
DB_PORT=5432

EMAIL_HOST=smtp.example.com
EMAIL_PORT=587
EMAIL_USER=noreply@example.com
EMAIL_PASSWORD=email-password
```

.gitignore

```
# .gitignore
```

Messages et Fichiers

Feedback Utilisateur et Uploads

Framework de Messages Django

Dans les Vues

```python
from django.contrib import messages

def create_book(request):
    if request.method == 'POST':
        form = BookForm(request.POST, request.FILES
        if form.is_valid():
            book = form.save()

            # Message de succès
            messages.success(
                request,
                f'Le livre "{book.title}" a été cré
            )
            return redirect('book_detail', pk=book.
        else:
            # Message d'erreur
            messages.error(
                request,
                'Erreur lors de la création'
            )
```

Dans les Templates

```html
<!-- Afficher les messages -->
{% if messages %}
<div class="messages">
    {% for message in messages %}
    <div class="alert alert-{{ message.tags }}">
        {{ message }}
    </div>
    {% endfor %}
</div>
{% endif %}

<!-- Avec Bootstrap -->
{% if messages %}
{% for message in messages %}
<div class="alert alert-{% if message.tags == 'erro
    <button type="button" class="close"
            data-dismiss="alert">&times;</button>
    {{ message }}
</div>
{% endfor %}
```

Upload de Fichiers

Configuration

```
# settings.py

# URL de base pour les médias
MEDIA_URL = '/media/'

# Chemin système pour les médias
MEDIA_ROOT = BASE_DIR / 'media'
```

URLs Dev

```
# urls.py
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    # ... vos URLs
]

# Servir les médias en développement
```

Modèle avec Image

```
from django.db import models

class Book(models.Model):
    title = models.CharField(max_length=200)
    # ...

    cover_image = models.ImageField(
        upload_to='book_covers/',
        null=True,
        blank=True,
        help_text="Image de couverture du livre"
    )

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

upload_to : sous-dossier dans MEDIA_ROOT

null=True, blank=True : champ optionnel

Formulaire et Template d'Upload

Formulaire

```python
# forms.py
from django import forms
from .models import Book

class BookForm(forms.ModelForm):
    class Meta:
        model = Book
        fields = [
            'title',
            'author',
            'cover_image'
        ]
        widgets = {
            'cover_image': forms.FileInput(attrs={
                'accept': 'image/*',
                'class': 'form-control'
            })
        }
```

Template

```html
<form method="post" enctype="multipart/form-data">
    {% csrf_token %}

    {{ form.as_p }}

    <button type="submit">Créer</button>
</form>

<!-- Afficher l'image -->
{% if book.cover_image %}
    <img src="{{ book.cover_image.url }}"
         alt="{{ book.title }}">
{% else %}
    <img src="{% static 'placeholder.jpg' %}"
         alt="Pas d'image">
{% endif %}

<!-- Attributs disponibles -->
{{ book.cover_image.url }}   <!-- URL -->
{{ book.cover_image.path }}  <!-- Chemin système --
```

Vue

Validation de Fichiers

```python
from django.core.exceptions import ValidationError

def validate_image_size(image):
    """Valider la taille de l'image"""
    max_size = 5 * 1024 * 1024  # 5 MB
    if image.size > max_size:
        raise ValidationError(
            f'La taille de l\'image ne doit pas dépasser 5 MB. '
            f'Taille actuelle : {image.size / 1024 / 1024:.2f} MB'
        )

def validate_image_dimensions(image):
    """Valider les dimensions"""
    from PIL import Image
    img = Image.open(image)
    width, height = img.size

    max_width = 2000
    max_height = 2000

    if width > max_width or height > max_height:
        raise ValidationError(
            f'Les dimensions ne doivent pas dépasser {max_width}x{max_height}. '
```

Exercice Pratique : Intégration Complète

Objectif : Ajouter l'authentification et les uploads au projet fil rouge

1. **Authentification**
   - Créer un profil utilisateur (bibliothécaire/usager)
   - Pages de login/logout
   - Restriction des vues selon le rôle
   - Admin réservé aux bibliothécaires
2. **Formulaires**
   - Formulaire d'emprunt avec validation métier
   - Formulaire de livre avec upload d'image
   - Messages de feedback
3. **Organisation**

Fin du Jour 2

Demain : Jour 3 - Tests, Déploiement et Best Practices

Intégrez les fonctionnalités au projet fil rouge

Je reste disponible pour vous aider, n’oubliez pas vous n’aurez pas le temps de tout faire, concentrez-vous sur la qualité du code et l’application des patterns vus aujourd’hui.

Formation Django

Système de Templates Avancé

Développeurs 4ème année

Système de Templates Django

Pattern Template et Moteur de Rendu

Architecture du Système de Templates

Composants

Template Engine

- Parser : analyse la syntaxe
- Compiler : génère du bytecode
- Renderer : produit le HTML final
- Context : données passées au template

Syntaxe Django

- Variables :
- Tags :
- Filtres : 0
- Commentaires :

Configuration

```
# settings.py
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django
        'DIRS': [
            BASE_DIR / 'templates',
        ],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors
                'django.template.context_processors
                'django.contrib.auth.context_proces
                'django.contrib.messages.context_pr
            ],
        },
    },
]
```

Variables et Résolution

Accès aux Variables

```
<!-- Variable simple -->
{{ book.title }}

<!-- Attribut d'objet -->
{{ book.author.last_name }}

<!-- Méthode sans arguments -->
{{ book.get_absolute_url }}

<!-- Index de liste -->
{{ books.0 }}

<!-- Clé de dictionnaire -->
{{ user_data.username }}

<!-- Accès en chaîne -->
{{ book.author.nationality.name }}

<!-- Si None ou absent -->
{{ book.subtitle|default:"Pas de sous-titre" }}
```

Contexte de Vue

```python
from django.shortcuts import render

def book_detail(request, pk):
    book = Book.objects.get(pk=pk)

    context = {
        'book': book,
        'related_books': Book.objects.filter(
            author=book.author
        ).exclude(pk=pk)[:5],
        'user_has_borrowed': Loan.objects.filter(
            book=book,
            borrower_email=request.user.email
        ).exists(),
        'stats': {
            'total_loans': book.loan_set.count(),
            'available': book.copies_available,
        }
    }
```

Tags de Contrôle de Flux

If / Elif / Else

```
{% if user.is_authenticated %}
    <p>Bonjour {{ user.username }}</p>

    {% if user.is_staff %}
        <a href="{% url 'admin:index' %}">Admin</a>
    {% elif user.groups.all %}
        <p>Membre de groupes</p>
    {% else %}
        <p>Utilisateur standard</p>
    {% endif %}
{% else %}
    <a href="{% url 'login' %}">Connexion</a>
{% endif %}

<!-- Opérateurs -->
{% if book.copies_available > 0 %}
{% if book.copies_available >= 5 %}
{% if book.copies_available == 0 %}
{% if book.copies_available != 0 %}
```

For

```
<!-- Boucle simple -->
{% for book in books %}
    <div class="book">
        <h3>{{ book.title }}</h3>
        <p>{{ book.author.last_name }}</p>
    </div>
{% endfor %}

<!-- Avec empty -->
{% for book in books %}
    <p>{{ book.title }}</p>
{% empty %}
    <p>Aucun livre disponible</p>
{% endfor %}

<!-- Variables de boucle -->
{% for book in books %}
    <p>
        {{ forloop.counter }}. {{ book.title }}
        {% if forloop.first %}(Premier){% endif %}
```

Filtres de Transformation

Filtres de Texte

```
<!-- Casse -->
{{ book.title|upper }}
{{ book.title|lower }}
{{ book.title|title }}
{{ book.title|capfirst }}

<!-- Longueur et troncature -->
{{ book.description|length }}
{{ book.description|truncatewords:20 }}
{{ book.description|truncatechars:100 }}

<!-- Formatage -->
{{ book.title|center:50 }}
{{ book.title|ljust:30 }}
{{ book.title|rjust:30 }}

<!-- Slugify -->
{{ book.title|slugify }}

<!-- Line breaks -->
```

Filtres Numériques et Dates

```
<!-- Nombres -->
{{ book.price|floatformat:2 }}
{{ 1234567|filesizeformat }}
{{ number|add:5 }}

<!-- Dates -->
{{ book.publication_date|date:"d/m/Y" }}
{{ book.publication_date|date:"D d M Y" }}
{{ loan.loan_date|time:"H:i" }}
{{ loan.loan_date|timesince }}
{{ loan.due_date|timeuntil }}

<!-- Liste et dictionnaire -->
{{ books|length }}
{{ books|first }}
{{ books|last }}
{{ books|join:", " }}
{{ books|slice:":5" }}
```

Chaîner les Filtres

```
<!-- Chaîner plusieurs filtres -->
{{ book.title|lower|truncatewords:5 }}

{{ loan.loan_date|date:"d/m/Y"|default:"Date inconnue" }}

{{ book.description|striptags|truncatechars:200|linebreaksbr }}

{{ books|length|add:10 }}

{{ book.title|slugify|upper }}

<!-- Ordre d'exécution : gauche à droite -->
{{ value|default:"N/A"|upper }}
<!-- Si value est None, devient "N/A" puis "N/A" -->

{{ value|upper|default:"N/A" }}
<!-- Si value est None, upper ne fait rien, puis default donne "N/A" -->

<!-- Dans un contexte logique -->
{% if book.title|length > 50 %}
    <p class="long-title">{{ book.title|truncatechars:50 }}</p>
{% else %}
```

Héritage de Templates

Pattern Template et DRY

Template de Base

```
<!-- templates/base.html -->
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>{% block title %}Bibliothèque{% endblock %}</title>

    <!-- CSS commun -->
    <link rel="stylesheet" href="{% static 'css/bootstrap.min.css' %}">
    <link rel="stylesheet" href="{% static 'css/style.css' %}">

    <!-- CSS additionnel -->
    {% block extra_css %}{% endblock %}
</head>
<body>
    <!-- Header -->
    <header>
        <nav class="navbar">
            <a href="{% url 'home' %}">Accueil</a>
            <a href="{% url 'book_list' %}">Catalogue</a>
```

Template Enfant

```
<!-- templates/library/book_list.html -->
{% extends 'base.html' %}
{% load static %}

{% block title %}Catalogue des livres - {{ block.super }}{% endblock %}

{% block extra_css %}
<link rel="stylesheet" href="{% static 'css/book-list.css' %}">
<style>
    .book-grid {
        display: grid;
        grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
        gap: 20px;
    }
</style>
{% endblock %}

{% block content %}
<div class="container">
    <h1>Catalogue des livres</h1>

    <!-- Formulaire de recherche -->
    <form method="get" class="search-form">
```

Surcharge et Extension de Blocks

Surcharge Complète

```
<!-- Remplace complètement le block -->
{% block content %}
<div class="custom-layout">
    <h1>Mon contenu</h1>
    <p>Remplace tout le block parent</p>
</div>
{% endblock %}
```

Extension avec block.super

```
<!-- Ajoute au contenu du parent -->
{% block content %}
    {{ block.super }}
    <div class="additional">
        <p>Contenu additionnel</p>
    </div>
{% endblock %}
```

Blocks Optionnels

```
<!-- Dans base.html -->
{% block sidebar %}
<!-- Rien par défaut -->
{% endblock %}

<!-- Dans enfant qui en a besoin -->
{% block sidebar %}
<aside>
    <h3>Navigation</h3>
    <ul>
        <li><a href="#">Lien 1</a></li>
        <li><a href="#">Lien 2</a></li>
    </ul>
</aside>
{% endblock %}

<!-- Dans enfant qui n'en a pas besoin -->
<!-- On ne définit pas le block -->
```

Blocks Imbriqués

Inclusion de Templates

Réutilisation et Composants

Tag Include

Template Réutilisable

```
<!-- templates/library/_book_card.html -->
<div class="book-card">
    {% if book.cover_image %}
        <img src="{{ book.cover_image.url }}"
             alt="{{ book.title }}">
    {% else %}
        <img src="{% static 'img/no-cover.jpg' %}"
             alt="Pas de couverture">
    {% endif %}

    <h3>{{ book.title }}</h3>
    <p class="author">{{ book.author.last_name }}</

    {% if book.copies_available > 0 %}
        <p class="available">
            {{ book.copies_available }} disponibles
        </p>
    {% else %}
        <p class="unavailable">Indisponible</p>
    {% endif %}
```

Utilisation

```
<!-- book_list.html -->
{% extends 'base.html' %}
{% load static %}

{% block content %}
<div class="book-grid">
    {% for book in books %}
        {% include 'library/_book_card.html' %}
    {% endfor %}
</div>
{% endblock %}

<!-- search_results.html -->
{% extends 'base.html' %}

{% block content %}
<h1>Résultats de recherche</h1>
<div class="results">
    {% for book in results %}
        {% include 'library/_book_card.html' %}
```

Include avec Contexte Personnalisé

Passer des Variables

```
<!-- Syntaxe with -->
{% include 'library/_book_card.html' with show_deta

{% include 'library/_book_card.html' with
    show_details=True
    highlight=False
%}

<!-- Renommer une variable -->
{% include 'library/_book_card.html' with book=feat

<!-- Isoler le contexte -->
{% include 'library/_book_card.html' with book=book
```

with : ajoute ou surcharge des variables

only : n'envoie que les variables spécifiées

Template Utilisant les Variables

```
<!-- _book_card.html -->
<div class="book-card {% if highlight %}highlighted
    <img src="{{ book.cover_image.url }}" alt="{{ b
    <h3>{{ book.title }}</h3>
    <p>{{ book.author.last_name }}</p>

    {% if show_details %}
        <p class="description">
            {{ book.description|truncatewords:20 }}
        </p>
        <p class="year">{{ book.publication_year }}
    {% endif %}

    <a href="{% url 'book_detail' book.pk %}">Voir<
</div>
```

Utilisation Contextuelle

Include Dynamique

```
<!-- Nom de template conditionnel -->
{% if user.is_staff %}
    {% include 'library/_book_card_admin.html' %}
{% else %}
    {% include 'library/_book_card_user.html' %}
{% endif %}

<!-- Ou plus court avec une variable -->
{% with template_name='library/_book_card_'|add:user_role|add:'.html' %}
    {% include template_name %}
{% endwith %}

<!-- Include depuis une variable de vue -->
<!-- Dans la vue -->
context = {
    'books': books,
    'card_template': 'library/_book_card_compact.html'
}

<!-- Dans le template -->
{% for book in books %}
    {% include card_template %}
{% endfor %}
```

Tags et Filtres Personnalisés

Étendre le Moteur de Templates

Créer un Filtre Personnalisé

```python
# library/templatetags/__init__.py
# Fichier vide nécessaire

# library/templatetags/library_filters.py
from django import template
from datetime import date

register = template.Library()

@register.filter
def days_overdue(loan):
    """Calculer le nombre de jours de retard d'un emprunt"""
    if loan.return_date or not loan.due_date:
        return 0

    delta = date.today() - loan.due_date
    return max(0, delta.days)

@register.filter
def calculate_penalty(loan):
    """Calculer la pénalité de retard"""
    days = days_overdue(loan)
    if days == 0:
```

Utiliser les Filtres Personnalisés

```
<!-- Charger les filtres -->
{% load library_filters %}

<!-- Utiliser days_overdue -->
{% for loan in loans %}
<tr>
    <td>{{ loan.book.title }}</td>
    <td>{{ loan.due_date|date:"d/m/Y" }}</td>

    {% with days=loan|days_overdue %}
        {% if days > 0 %}
            <td class="overdue">
                {{ days }} jour{{ days|pluralize }}
            </td>
        {% else %}
            <td class="ok">À jour</td>
        {% endif %}
    {% endwith %}
</tr>
{% endfor %}

<!-- Calculer et afficher la pénalité -->
{% for loan in overdue_loans %}
```

Créer un Tag Simple

```python
# library/templatetags/library_tags.py
from django import template
from django.utils.html import format_html

register = template.Library()

@register.simple_tag
def loan_status_badge(loan):
    """Afficher un badge de statut d'emprunt"""
    if loan.return_date:
        css_class = 'badge-success'
        text = 'Retourné'
    elif loan.due_date < date.today():
        css_class = 'badge-danger'
        text = 'En retard'
    else:
        css_class = 'badge-primary'
        text = 'Actif'

    return format_html(
        '<span class="badge {}">{}</span>',
        css_class,
```

Utiliser les Tags Personnalisés

```
<!-- Charger les tags -->
{% load library_tags %}

<!-- Utiliser loan_status_badge -->
{% for loan in loans %}
<tr>
    <td>{{ loan.book.title }}</td>
    <td>{{ loan.borrower_name }}</td>
    <td>{% loan_status_badge loan %}</td>
</tr>
{% endfor %}

<!-- Utiliser availability_indicator -->
{% for book in books %}
<div class="book-card">
    <h3>{{ book.title }}</h3>
    {% availability_indicator book %}
    <a href="{% url 'book_detail' book.pk %}">Voir</a>
</div>
{% endfor %}

<!-- Utiliser greeting (utilise le contexte) -->
```

Tag Inclusion

```python
# library/templatetags/library_tags.py

@register.inclusion_tag('library/_book_summary.html')
def show_book_summary(book):
    """Afficher un résumé de livre"""
    return {
        'book': book,
        'is_available': book.copies_available > 0,
        'popularity': book.loan_set.count()
    }

@register.inclusion_tag('library/_loan_history.html')
def show_loan_history(book, limit=5):
    """Afficher l'historique d'emprunts d'un livre"""
    loans = book.loan_set.all().order_by('-loan_date')[:limit]
    return {
        'loans': loans,
        'total_loans': book.loan_set.count()
    }

@register.inclusion_tag('library/_stats_widget.html', takes_context=True)
def stats_widget(context, widget_type='summary'):
    """Widget de statistiques"""
```

Utiliser les Inclusion Tags

```
{% load library_tags %}

<!-- Utiliser show_book_summary -->
<div class="featured-books">
    {% for book in featured %}
        {% show_book_summary book %}
    {% endfor %}
</div>

<!-- Utiliser show_loan_history -->
<section>
    <h2>Historique d'emprunts</h2>
    {% show_loan_history book %}
</section>

<!-- Avec paramètre optionnel -->
<section>
    <h2>Derniers emprunts</h2>
    {% show_loan_history book limit=10 %}
</section>

<!-- Utiliser stats_widget -->
```

Exercice Pratique : Templates Avancés

Objectif : Améliorer les templates du projet fil rouge

1. **Héritage**
   - Créer un template de base complet
   - Header avec navigation conditionnelle (authentifié/non)
   - Footer avec informations
   - Blocks pour title, extra_css, content, extra_js
   - Affichage des messages Django
2. **Includes**
   - Composant carte de livre réutilisable
   - Composant formulaire de recherche
   - Composant pagination
   - Utiliser avec et only

Templates Django : Récapitulatif

Héritage : Structure commune avec blocks

Inclusion : Composants réutilisables

Filtres : Transformation de valeurs

Tags : Logique de présentation complexe

Séparation : HTML propre, logique en Python

Formation Django

Jour 3 Après-midi : Sécurité, i18n et Déploiement

Développeurs 4ème année

Sécurité dans Django

CSRF, XSS, SQL Injection et Plus

Sécurité : Vue d'Ensemble

Menaces Courantes

Injection

- SQL Injection
- Command Injection
- Template Injection

Cross-Site Attacks

- XSS (Cross-Site Scripting)
- CSRF (Cross-Site Request Forgery)
- Clickjacking

Autres

- Session Hijacking

Protections Django

Par défaut :

- Protection CSRF activée
- Protection XSS (auto-escaping)
- Protection SQL Injection (ORM)
- Protection Clickjacking (X-Frame-Options)
- Validation des entrées
- Hashage sécurisé des mots de passe

Configuration :

- HTTPS/SSL
- Cookies sécurisés

Protection CSRF (Cross-Site Request Forgery)

Qu'est-ce que CSRF ?

Attaque :

1. Utilisateur authentifié sur site A
2. Visite un site B malveillant
3. Site B fait une requête vers site A
4. Le navigateur envoie les cookies de A
5. Action exécutée à l'insu de l'utilisateur

Exemple :

```
<!-- Site malveillant -->
<img src="https://bank.com/transfer?
  to=attacker&amount=1000">
```

Protection Django

Token CSRF :

- Généré par le serveur
- Stocké dans la session
- Inséré dans le formulaire
- Vérifié à chaque POST

Dans les templates :

```
<form method="post">
    {% csrf_token %}
    <!-- champs du formulaire -->
</form>
```

Protection XSS (Cross-Site Scripting)

Qu'est-ce que XSS ?

Attaque : Injecter du JavaScript malveillant dans une page web

Exemple :

```
# Vue vulnérable (sans Django)
def search(request):
    query = request.GET.get('q', '')
    html = f"<p>Résultats pour : {query}</p>"
    return HttpResponse(html)
```

URL : `?q=<script>alert('XSS')</script>`

Résultat : le script s'exécute

Impact :

- Vol de cookies/sessions

Protection Django

Auto-escaping dans les templates :

```
<!-- Automatiquement échappé -->
<p>Résultats pour : {{ query }}</p>

<!-- Si query = "<script>alert('XSS')</script>" -->
<!-- Rendu : -->
<p>Résultats pour :
  &lt;script&gt;alert('XSS')&lt;/script&gt;
</p>
```

Marquer comme safe (attention !) :

```
<!-- Désactive l'escaping -->
{{ content|safe }}

<!-- Ou dans le code -->
from django.utils.safestring import mark_safe
```

Protection SQL Injection

Qu'est-ce que SQL Injection ?

Attaque : Injecter du SQL malveillant dans une requête

Exemple vulnérable :

```
# NE JAMAIS FAIRE ÇA
def get_user(username):
    query = f"SELECT * FROM users WHERE username =
    cursor.execute(query)
```

Input malveillant : `admin' OR '1'='1`

Requête résultante :

```
SELECT * FROM users
WHERE username = 'admin' OR '1'='1'
```

Protection Django

L'ORM protège automatiquement :

```
# Sécurisé
User.objects.filter(username=user_input)

# Django génère :
# SELECT * FROM users
# WHERE username = %s
# Avec paramètre : user_input
```

Requêtes SQL brutes (paramétrisées) :

```
# Sécurisé
from django.db import connection

cursor = connection.cursor()
cursor.execute(
    "SELECT * FROM library_book WHERE title = %s",
```

Configuration de Sécurité pour la Production

```
# settings/production.py

# HTTPS/SSL
SECURE_SSL_REDIRECT = True  # Rediriger HTTP vers HTTPS
SECURE_PROXY_SSL_HEADER = ('HTTP_X_FORWARDED_PROTO', 'https')

# Cookies sécurisés
SESSION_COOKIE_SECURE = True  # Cookie de session sur HTTPS uniquement
CSRF_COOKIE_SECURE = True     # Cookie CSRF sur HTTPS uniquement
SESSION_COOKIE_HTTPONLY = True  # Pas accessible via JavaScript
CSRF_COOKIE_HTTPONLY = True

# HSTS (HTTP Strict Transport Security)
SECURE_HSTS_SECONDS = 31536000  # 1 an
SECURE_HSTS_INCLUDE_SUBDOMAINS = True
SECURE_HSTS_PRELOAD = True

# Content Security Policy
CSP_DEFAULT_SRC = ("'self'",)
CSP_SCRIPT_SRC = ("'self'", "https://cdn.example.com")
CSP_STYLE_SRC = ("'self'", "https://cdn.example.com")

# Autres headers de sécurité
```

Internationalisation (i18n)

Applications Multilingues

i18n et l10n

Concepts

Internationalisation (i18n) : Préparer l'application à être traduite

- Marquer les chaînes traduisibles
- Format des dates/nombres adaptable
- Support de langues multiples

Localisation (l10n) : Traduction effective dans une langue

- Fichiers de traduction (.po)
- Formats locaux (dates, monnaies)

Locale : Combinaison langue + région

Configuration Django

```
# settings.py

# Activer i18n
USE_I18N = True
USE_L10N = True

# Langue par défaut
LANGUAGE_CODE = 'fr-fr'

# Langues disponibles
LANGUAGES = [
    ('fr', 'Français'),
    ('en', 'English'),
    ('es', 'Español'),
]

# Répertoires des traductions
LOCALE_PATHS = [
    BASE_DIR / 'locale',
]
```

Marquer les Chaînes Traduisibles

Dans le Code Python

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

# Dans une vue
def book_list(request):
    message = _("Liste des livres")
    return render(request, 'books.html', {
        'message': message
    })

# Dans un modèle (lazy)
class Book(models.Model):
    title = models.CharField(
        max_length=200,
        verbose_name=gettext_lazy("Titre")
    )

    class Meta:
        verbose_name = gettext_lazy("Livre")
        verbose_name_plural = gettext_lazy("Livres"
```

Dans les Templates

```
{% load i18n %}

<!-- Traduction simple -->
<h1>{% trans "Bibliothèque" %}</h1>

<!-- Avec variable -->
<p>{% blocktrans with name=user.name %}
    Bonjour {{ name }}
{% endblocktrans %}</p>

<!-- Pluriel -->
{% blocktrans count counter=books|length %}
    {{ counter }} livre disponible
{% plural %}
    {{ counter }} livres disponibles
{% endblocktrans %}

<!-- Contexte -->
{% trans "Read" context "verb" %}
{% trans "Read" context "past participle" %}
```

Créer et Compiler les Traductions

Générer les Fichiers de Traduction

```
# Créer/mettre à jour les messages
python manage.py makemessages -l fr
python manage.py makemessages -l en
python manage.py makemessages -l es

# Pour JavaScript
python manage.py makemessages -d djangojs -l fr

# Ignorer certains dossiers
python manage.py makemessages -l fr \
  --ignore=venv/* --ignore=node_modules/*
```

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

Structure d'un Fichier .po

Compiler les Traductions

```
# Compiler tous les fichiers .po en .mo
python manage.py compilemessages

# Compiler une langue spécifique
python manage.py compilemessages -l fr
```

Crée `locale/fr/LC_MESSAGES/django.mo` (binaire)

Workflow de Traduction

1. Développeur marque les chaînes avec `_()` et `{% trans %}`
2. Lancer `makemessages` pour générer les .po

Sélection de la Langue

```python
# urls.py (racine)
from django.conf.urls.i18n import i18n_patterns
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('i18n/', include('django.conf.urls.i18n')),  # Changement de langue
]

# URLs avec préfixe de langue
urlpatterns += i18n_patterns(
    path('', include('library.urls')),
    prefix_default_language=False,
)
# Génère : /fr/books/, /en/books/, /es/books/
```

Template de sélection de langue :

```python
{% load i18n %}

<form action="{% url 'set_language' %}" method="post">
    {% csrf_token %}
```

Déploiement Django

Mettre en Production

Déploiement : Vue d'Ensemble

Composants en Production

Serveur Web :

- Nginx ou Apache
- Sert les fichiers statiques
- Reverse proxy vers l'application

Serveur WSGI :

- Gunicorn ou uWSGI
- Exécute le code Django
- Gère les workers Python

Base de Données :

Architecture Typique

```
Internet
    ↓
[Nginx :80/:443]
    ↓ (reverse proxy)
[Gunicorn :8000]
    ↓
[Django Application]
    ↓
[PostgreSQL :5432]
    ↓
[Redis :6379]
```

Nginx :

- Terminaison SSL
- Fichiers statiques
- Proxy vers Gunicorn

Préparation au Déploiement

```python
# settings/production.py

import os
from .base import *

# Sécurité
DEBUG = False
SECRET_KEY = os.environ['DJANGO_SECRET_KEY']
ALLOWED_HOSTS = ['monsite.com', 'www.monsite.com']

# Base de données
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': os.environ['DB_NAME'],
        'USER': os.environ['DB_USER'],
        'PASSWORD': os.environ['DB_PASSWORD'],
        'HOST': os.environ.get('DB_HOST', 'localhost'),
        'PORT': os.environ.get('DB_PORT', '5432'),
    }
}

# Fichiers statiques
```

Collecte des Fichiers Statiques

```
# Collecter tous les fichiers statiques
python manage.py collectstatic --noinput

# Résultat :
# Copie tous les fichiers de :
# - app/static/
# - STATICFILES_DIRS
# Vers STATIC_ROOT

# Exemple de sortie :
# 121 static files copied to '/var/www/monsite/static'
```

Configuration :

```
# settings/production.py

STATIC_ROOT = '/var/www/monsite/static/'
STATIC_URL = '/static/'

# Dossiers statiques supplémentaires
STATICFILES_DIRS = [
```

Gunicorn et Systemd

Installer Gunicorn :

```
pip install gunicorn
```

Démarrer Gunicorn :

```
gunicorn myproject.wsgi:application --bind 0.0.0.0:8000 --workers 3
```

Fichier systemd (`/etc/systemd/system/django.service`) :

Unit

```
Description=Gunicorn daemon for Django project
After=network.target

[Service]
User=www-data
Group=www-data
WorkingDirectory=/var/www/myproject
Environment="DJANGO_SETTINGS_MODULE=myproject.settings.production"
Environment="DJANGO_SECRET_KEY=your-secret-key"
```

Configuration Nginx

```
# /etc/nginx/sites-available/monsite

upstream django {
    server unix:/var/www/myproject/myproject.sock;
}

server {
    listen 80;
    server_name monsite.com www.monsite.com;

    # Redirection HTTP vers HTTPS
    return 301 https://$server_name$request_uri;
}

server {
    listen 443 ssl http2;
    server_name monsite.com www.monsite.com;

    # SSL
    ssl_certificate /etc/letsencrypt/live/monsite.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/monsite.com/privkey.pem;
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers HIGH:!aNULL:!MD5;
```

Checklist de Déploiement

Avant le Déploiement

- ☐ `DEBUG = False`
- ☐ `SECRET_KEY` en variable d'env
- ☐ `ALLOWED_HOSTS` configuré
- ☐ PostgreSQL configuré
- ☐ Migrations appliquées
- ☐ Superutilisateur créé
- ☐ `collectstatic` exécuté
- ☐ Dépendances installées
- ☐ Variables d'environnement définies
- ☐ Certificat SSL obtenu
- ☐ Nginx configuré

Après le Déploiement

- ☐ Tester toutes les fonctionnalités
- ☐ Vérifier les logs
- ☐ Tester les emails
- ☐ Vérifier les fichiers statiques
- ☐ Tester les uploads
- ☐ Vérifier les redirections HTTPS
- ☐ Tester sur mobile
- ☐ Analyser les performances
- ☐ Configurer le monitoring
- ☐ Configurer les sauvegardes BDD
- ☐ Documentation de déploiement

Récapitulatif et Conclusion

Ce que nous avons vu en 3 jours

Jour 1 : Fondamentaux et Architecture

Matin

Architecture MVT

- Pattern MVT vs MVC
- Philosophie Django
- Installation et setup

Routage

- URLconf et patterns
- URL Dispatcher pattern
- Includes et namespaces

Structure Projet

- Projet vs Applications

Après-midi

Modèles et ORM

- Pattern Active Record
- Types de champs
- Relations (FK, M2M, O2O)

QuerySets

- API de requêtes
- Lookups et filtres
- Optimisation (select_related, prefetch_related)

Migrations

Jour 2 : Vues, Formulaires et Templates

Matin

Vues

- Function-Based Views
- Class-Based Views
- Generic Views
- Pattern Template Method
- Mixins

Formulaires

- Pattern Form Object
- Validation
- ModelForms

Après-midi

Templates

- Héritage (Pattern Template)
- Includes et composants
- Tags et filtres
- Context processors

Middleware

- Pattern Chain of Responsibility
- Ordre d'exécution
- Middlewares personnalisés

Architecture

Jour 3 : Admin, Tests, Sécurité et Production

Matin

Admin Django

- Pattern Admin/Backoffice
- Personnalisation ModelAdmin
- Actions et Inlines
- Interface auto-générée

Signaux

- Pattern Observer
- Signaux intégrés
- Signaux personnalisés
- Cas d'usage

Après-midi

Sécurité

- CSRF, XSS, SQL Injection
- Configuration production
- Headers de sécurité
- Validation et sanitization

i18n/l10n

- Internationalisation
- Fichiers de traduction
- Sélection de langue
- Formats locaux

Design Patterns Vus

Pattern	Où dans Django	Utilité
MVT (MVC)	Architecture globale	Séparation des responsabilités
Active Record	Modèles ORM	Encapsulation données + persistance
URL Dispatcher	Système de routage	Découplage URL/logique
Template	Héritage de templates	Structure réutilisable
Template Method	Generic Views	Squelette d'algorithme
Form Object	Formulaires	Encapsulation validation
Admin/Backoffice	Django Admin	Séparation admin/public

Bonnes Pratiques Django

Architecture

- Apps petites et focalisées
- Service Layer pour logique complexe
- Managers pour requêtes réutilisables
- Settings par environnement
- Secrets en variables d'env

Performance

- select_related pour FK/OneToOne
- prefetch_related pour ManyToMany
- Indexes sur colonnes filtrées
- Cache des requêtes coûteuses
- Pagination systématique
- only() et defer() si besoin

Code

- PEP 8 et conventions Django
- Type hints (Python 3.10+)
- Docstrings claires
- Code DRY

Sécurité

- DEBUG=False en production
- ALLOWED_HOSTS configuré
- Cookies sécurisés sur HTTPS

Ressources pour Continuer

Documentation

Officielle :

- Django Docs
- Django Tutorial
- Django Best Practices

Livres :

- *Two Scoops of Django* (Greenfeld & Roy)
- *Django for Professionals* (William S. Vincent)
- *Mastering Django* (Nigel George)

Communauté :

- Django Forum

Packages Utiles

API :

- Django REST Framework
- Django Ninja

Async :

- Django Channels
- Celery

Admin :

- django-import-export
- django-admin-tools

Projet Fil Rouge : Suite

Ce qu'on a construit

- Modèles complets (Book, Author, Loan)
- CRUD avec vues et formulaires
- Authentification et permissions
- Templates avec héritage
- Interface admin personnalisée
- Tests unitaires et d'intégration

Pour aller plus loin

- API REST avec Django REST Framework
- Tâches asynchrones (emails, rapports)

Examen et Rendu du TP

Examen théorique : 1h30

- Questions sur les concepts vus
- Design patterns
- Bonnes pratiques
- Sécurité

Rendu du TP fil rouge :

- Code source sur Git
- Documentation (README)
- Tests fonctionnels
- Démonstration

Questions / Discussions

Des questions sur Django ?

Sur un point spécifique ?

Sur votre projet ?

Bonne chance pour l’examen et vos futurs projets Django

Formation Django - Jours 1, 2 et 3