

Framework **django**

Support de cours/TD Framework django

Classes: CII-3-GLSI

Année universitaire 2023/2024

Réalisé et enseigné par: Nizar MAATOUG

Aperçu du contenu (1 / 2)

- ▶ Framework django: Big Picture
- ▶ Framework django: Architecture
- ▶ Installation des outils, premier projet django
- ▶ Anatomie, concepts Projet/Application django
- ▶ Couche Modèle:
 - ▶ Concept ORM
 - ▶ Modèle: définition, manipulation, relationships
 - ▶ Configuration de la base de données
- ▶ Interface Admin: découverte, publier, manipuler des modèles
- ▶ Couche View: définir des actions: FBV (Function Based Views), CBV (Class Based Views)
- ▶ django templates
- ▶ django Forms

Aperçu du contenu (2/2)

- ▶ Sécurité: Authentification, rôles, permissions, groupes.
- ▶ Application: développer une application web avec le framework **django**

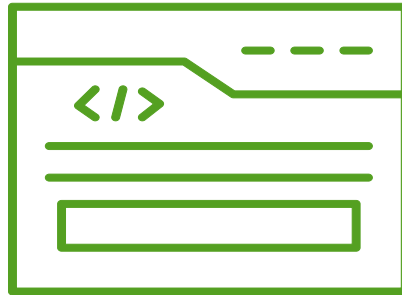
objectifs

- ▶ Maîtriser l'architecture du framework **django**
- ▶ Développer et tester une application web **django**

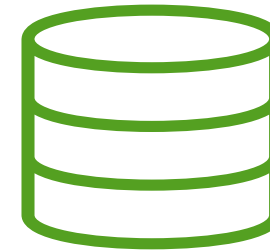
Prérequis



Framework django



Développement web
HTML, CSS, JavaScript



Base de données
Concepts de base
Table, rows
Insert, update

django

Framework web Python

Développer rapidement des applications web

Avec le minimum de code

Développé entre 2003 et 2005

Philosophie: Piles incluses

ORM

Templates

Forms

Admin

URL Mapping

Packages

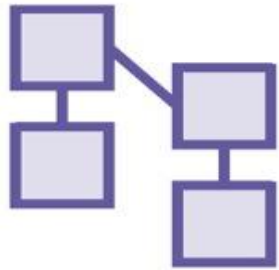
Polyvalent

- ▶ Django peut être (et a été) utilisé pour créer presque tous les genres de sites:
 - ▶ Site d'actualité
 - ▶ Gestionnaire de données
 - ▶ Wikis
 - ▶ Réseaux sociaux
- ▶ Qui utilise django ?
 - ▶ Instagram
 - ▶ Spotify
 - ▶ Youtube
 - ▶ Dropbox
 - ▶ ...

Autres

- ▶ Sécurisé
- ▶ Maintenable
- ▶ Scalable
- ▶ Portable

django: Architecture MVT



Model

Représente les données,
Assure le mapping
Objet/Relationnel

MVC: "Controller"



View

Reçoit une requête http,
effectue un traitement,
retourne une réponse http

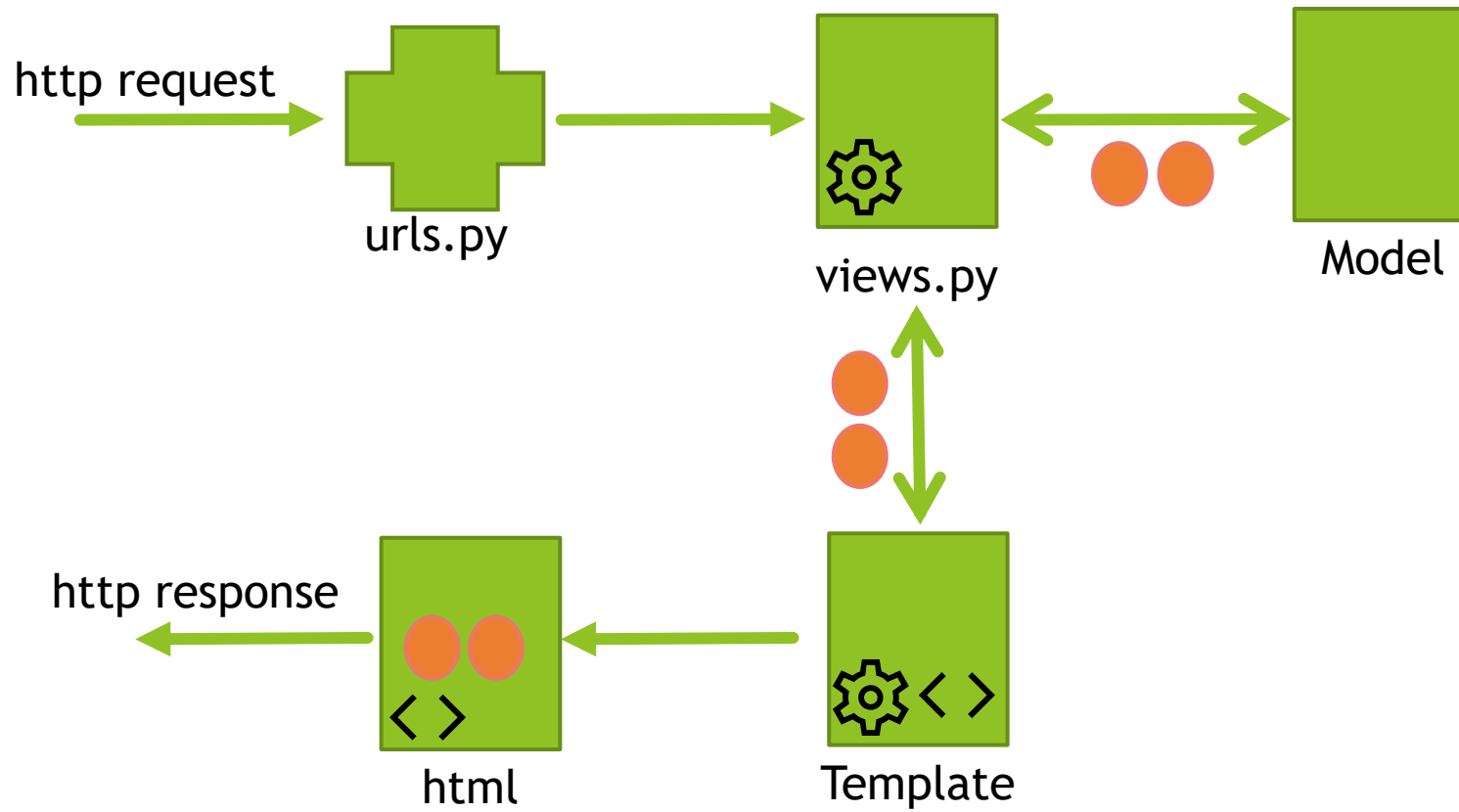
MVC: "View"



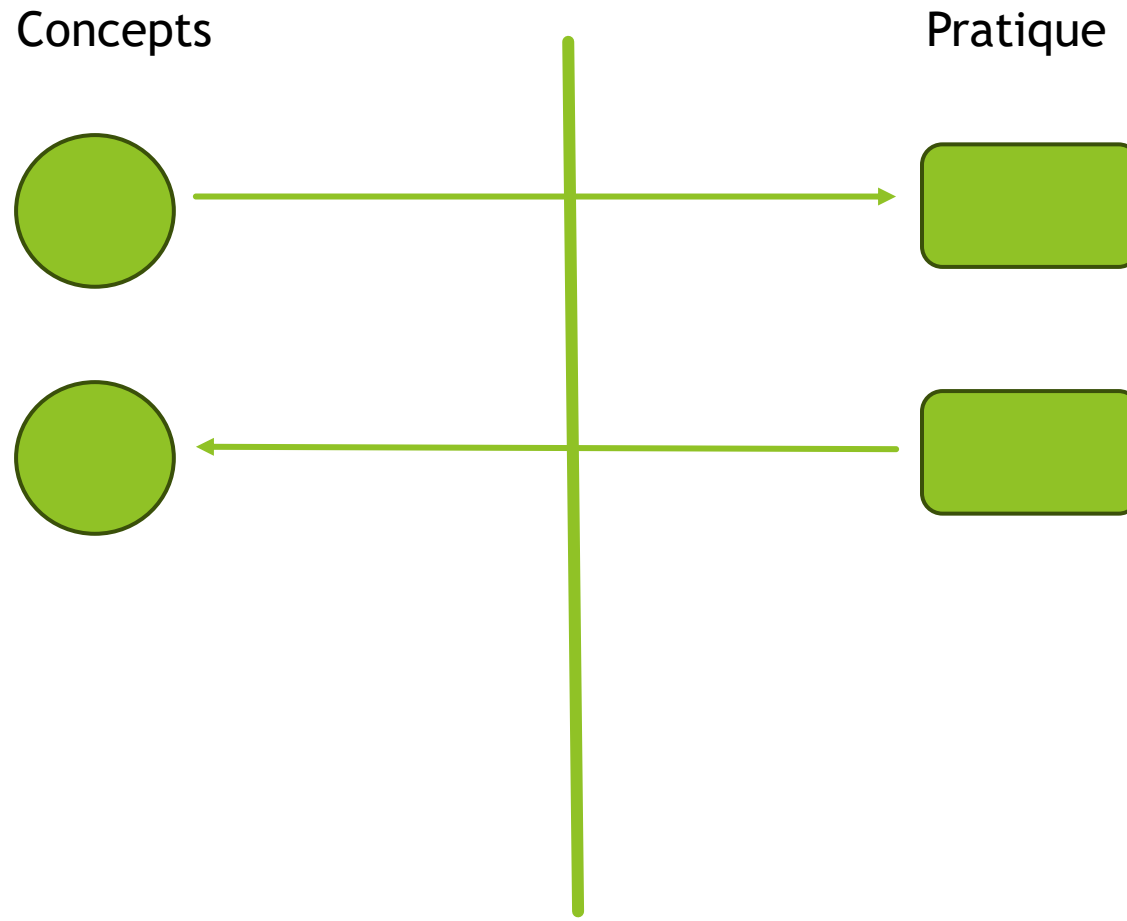
Template

Définit, génère la
présentation HTML

django: Architecture MVT

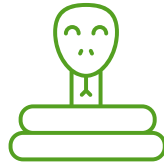


Approche





Système d'exploitation: windows, Linux, Mac OS



Python: Latest version 3.11



Éditeur de code: VS code, PyCharm

django: Première application

Étape par Étape

- ▶ Installation et configuration des outils
- ▶ Création et activation d'un environnement virtuel pour le projet
- ▶ Installation du framework django (virtual env activé)
- ▶ Sauvegarder les dépendances
- ▶ Création du projet django nommé **hello_word_project**
- ▶ Création de l'application **pages**
- ▶ Déclaration de l'application **pages**
- ▶ Première page web

Installation et configuration des outils

- ▶ Installer Python: <https://www.python.org/downloads/>
- ▶ Installer VS Code ou PyCharm

Environnement virtuel

Étape par Étape



- ▶ Ne pas installer les packages python globalement
- ▶ Toujours travailler dans un environnement virtuel
- ▶ Éviter les conflits de dépendances
- ▶ Travailler dans un contexte isolé.

Environnement virtuel

Étape par Étape

#créer un dossier pour le projet

```
$ cd framework_django\projects
```

```
$ mkdir helloworld
```

```
$ cd helloworld
```

#créer et activer l'environnement virtuel

```
$ python -m venv .venv
```

```
$ .venv\Scripts\Activate.ps1
```

```
(.venv) $ python -m pip install django
```

```
$ source .venv/bin/activate
```

Sauvegarder les dépendances

#sauvegarder les dépendances

(.venv) \$ pip freeze > requirements.txt

```
requirements.txt X
requirements.txt
1  asgiref==3.7.2
2  Django==4.2.4
3  sqlparse==0.4.4
4  tzdata==2023.3
5
```

Créer le projet django_project

Étape par Étape

```
# créer le projet django_project
```

```
(.venv) $ django-admin startproject django_project .
```

```
# lancer le projet dans VS code
```


```
(.venv) $ code .
```

django_project: Anatomie


Étape par Étape

✓  django_project

Marque le répertoire comme package => pouvoir importer les composants


>  __pycache__

Standard Python pour application et serveur asynchrone

 __init__.py

 asgi.py


Contrôle l'application django via des configurations


 settings.py

Binding entre requête http et traitement associé

 urls.py

Web Server Gateway Interface

 wsgi.py

 manage.py

Ne fait pas partie du projet, mais utile pour exécuter des commandes: runserver, créer nouvelle application,...

Lancer le projet

démarrer le projet

(.venv) \$ python manage.py runserver

appliquer les migrations

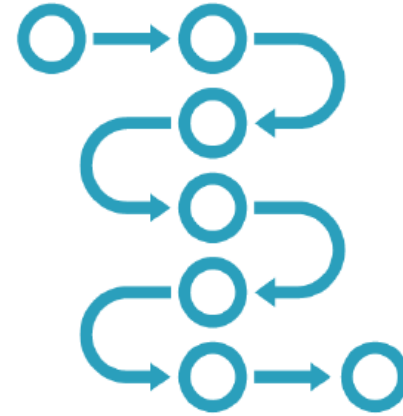
(.venv) \$ python manage.py migrate

Migrations



Models

Classes Python
Associées aux tables de
la BD



Migrations

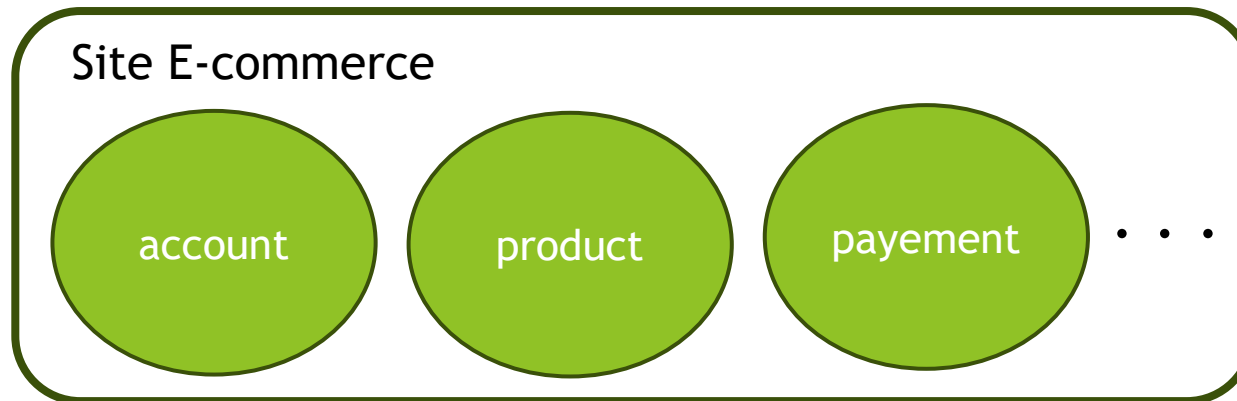
Ensemble de scripts
décrivant le schéma de
la BD

Étape par Étape

Application django: pages

Étape par Étape

- ▶ django utilise les concepts **projet** et **application** pour maintenir le code "clean"
- ▶ un projet django peut contenir plusieurs applications
- ▶ Chaque application contrôle une fonctionnalité isolée du projet.



Application django: pages

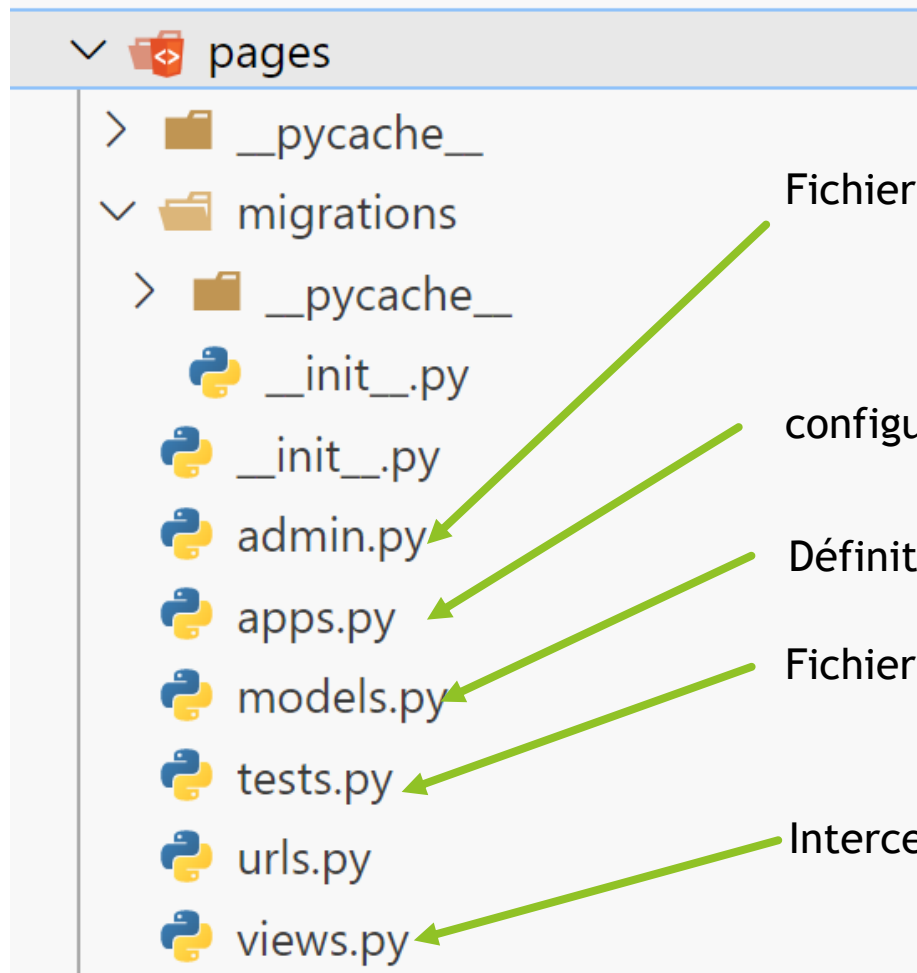
| # créer l'application pages

| (.venv) \$ python manage.py startapp pages

Étape par Étape

Application django: Anatomie

Étape par Étape



Déclaration de l'application pages

Étape par Étape

Application definition

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    "pages", #new
]
```

Première page web: views.py

Étape par Étape

views.py ×

pages > views.py > ...

```

1  from django.shortcuts import render
2  from django.http import HttpResponse
3
4  # Create your views here.
5
6  def homePageView(request):
7      return HttpResponse("Hello word")
8

```

Première page web: pages.urls.py

Étape par Étape

```

urls.py  ×
pages > urls.py > ...
1  from django.urls import path
2
3  from .views import homePageView
4
5
6  urlpatterns=[
7      path('',homePageView, name="home"),
8  ]

```

Première page web: urls.py

Étape par Étape

```

urls.py  ×
django_project > urls.py > ...

17  from django.contrib import admin
18  from django.urls import path,include
19
20  urlpatterns = [
21      path('admin/', admin.site.urls),
22      path('',include("pages.urls")),
23  ]
24

```



- ▶ Django: framework web Python
- ▶ Développement web simple et rapide
- ▶ Piles incluses: ORM, Admin module, ...
- ▶ Architecture MVT (Model, View, Template)
- ▶ Un projet django, plusieurs applications
- ▶ Virtual environnement

Application: Meeting Planner

Résultats

- ▶ Maîtriser l'architecture MVT
- ▶ Découvrir les concepts de base de django:
 - ▶ Model
 - ▶ View
 - ▶ Template
 - ▶ URLs
 - ▶ Admin panel
 - ▶ Forms

Meeting Planner

- ▶ Permet de planifier les réunions
- ▶ Planifier une **réunion**:
 - ▶ Définir la date, la durée
 - ▶ affecter une **salle** à cette réunion

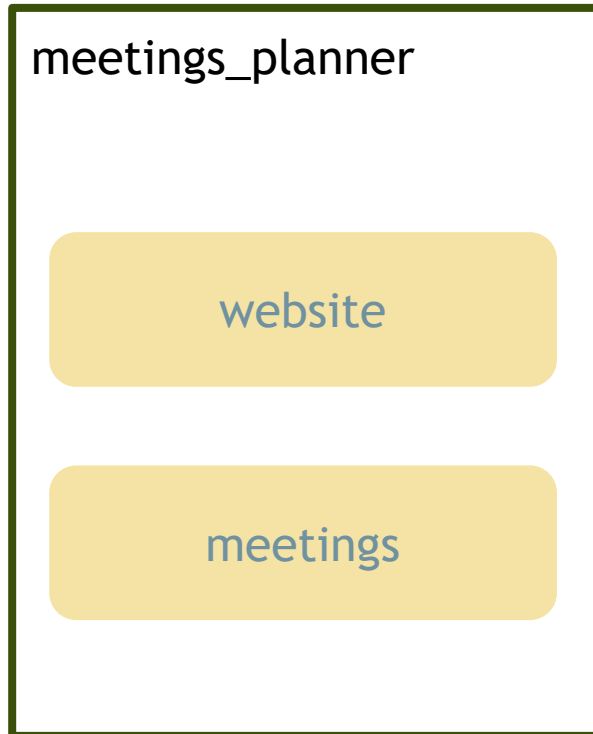
Démo de l'application

Démarche

- ▶ Créer le projet **meetings_planner**
 - ▶ Créer et activer l'environnement virtuel
 - ▶ Installer les dépendances
 - ▶ Créer le projet et les applications django
- ▶ Implémenter les pages web (accueil, about)
- ▶ Implémenter le modèle **Meeting**
- ▶ Admin Panel:
 - ▶ Créer un **superuser**
 - ▶ Gérer le modèle **Meeting**
- ▶ Gérer les **Meeting**
- ▶ implémenter le modèle Room
- ▶ Définir l'association Meeting-----Room

Étape par Étape

Architecture du projet



django apps



Python package

Contient models, views, templates, urls

Les projets django contiennent plusieurs apps

Les Apps peuvent être réutilisées

Maintenir les apps petites et simples

Création du projet

Étape par Étape

#créer un dossier pour le projet

```
$ cd framework_django\projects
```

```
$ mkdir meetings_planner
```

```
$ cd meetings_planner
```

#créer et activer l'environnement virtuel

```
$ python -m venv my_env
```

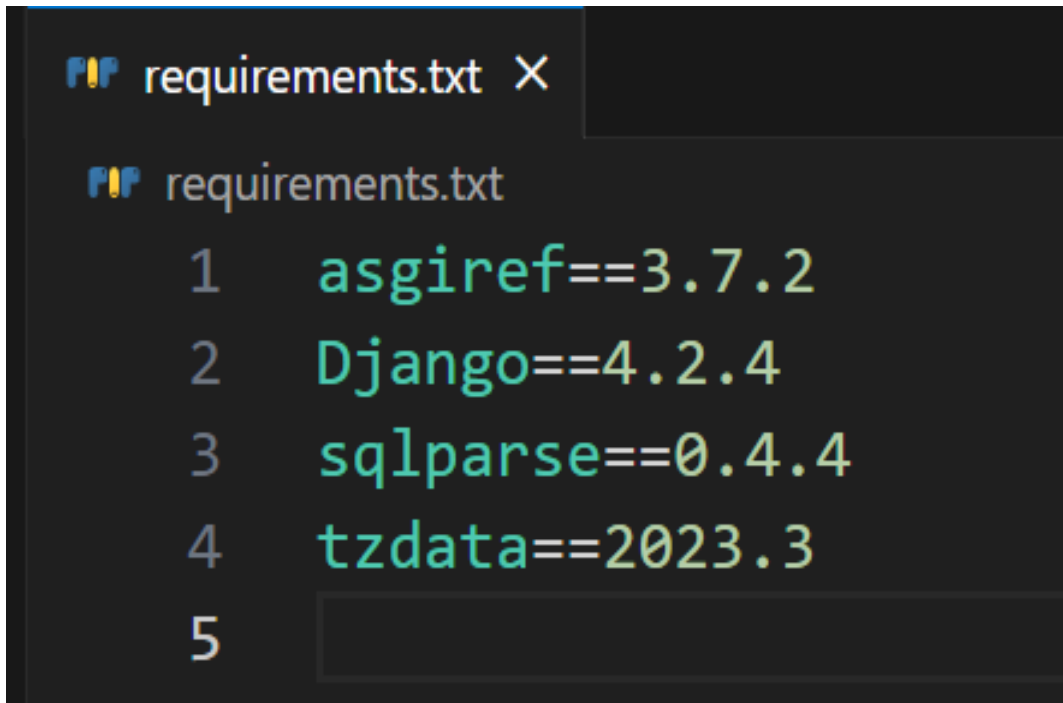
```
$ my_env\Scripts\Activate
```

```
(my_env) $ python -m pip install django
```

Sauvegarder les dépendances

#sauvegarder les dépendances

(my_env) \$ pip freeze > requirements.txt



```
requirements.txt X
requirements.txt
1  asgiref==3.7.2
2  Django==4.2.4
3  sqlparse==0.4.4
4  tzdata==2023.3
5
```

Création projet et applications

créer le projet django_project

(my_env) \$ django-admin startproject meetings_planner .

lancer le projet dans VS code

(my_env) \$ code .

créer l'application website

(my_env) \$ python manage.py startapp website









créer l'application meetings

(my_env) \$ python manage.py startapp meetings

Étape par Étape

Structure du projet

✓ MEETINGS_PLANNER

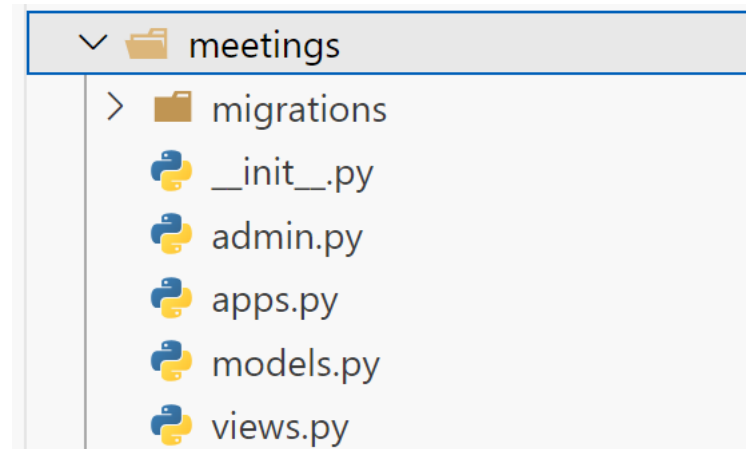
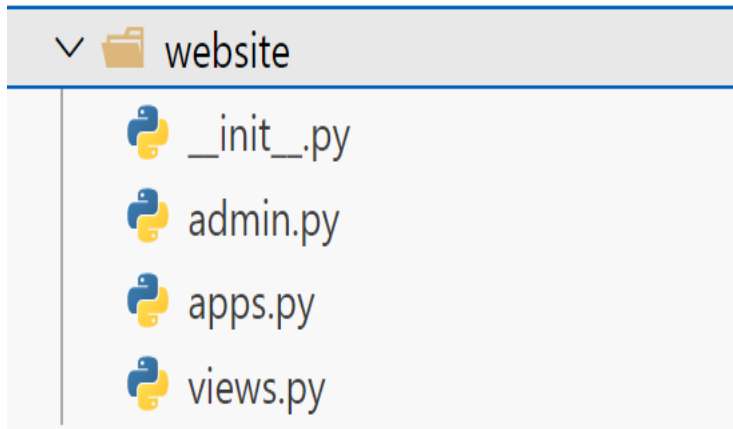
- >  meetings
- >  meetings_planner
- >  my_env
- >  website
-  .gitignore
-  db.sqlite3
-  manage.py
-  requirements.txt

Framework django

Étape par Étape

Structure du projet

Étape par Étape



Settings.py: Déclaration des applications

Étape par Étape

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'website', #new
    'meetings', #new
]
```

Application des migrations

Étape par Étape

appliquer les migrations

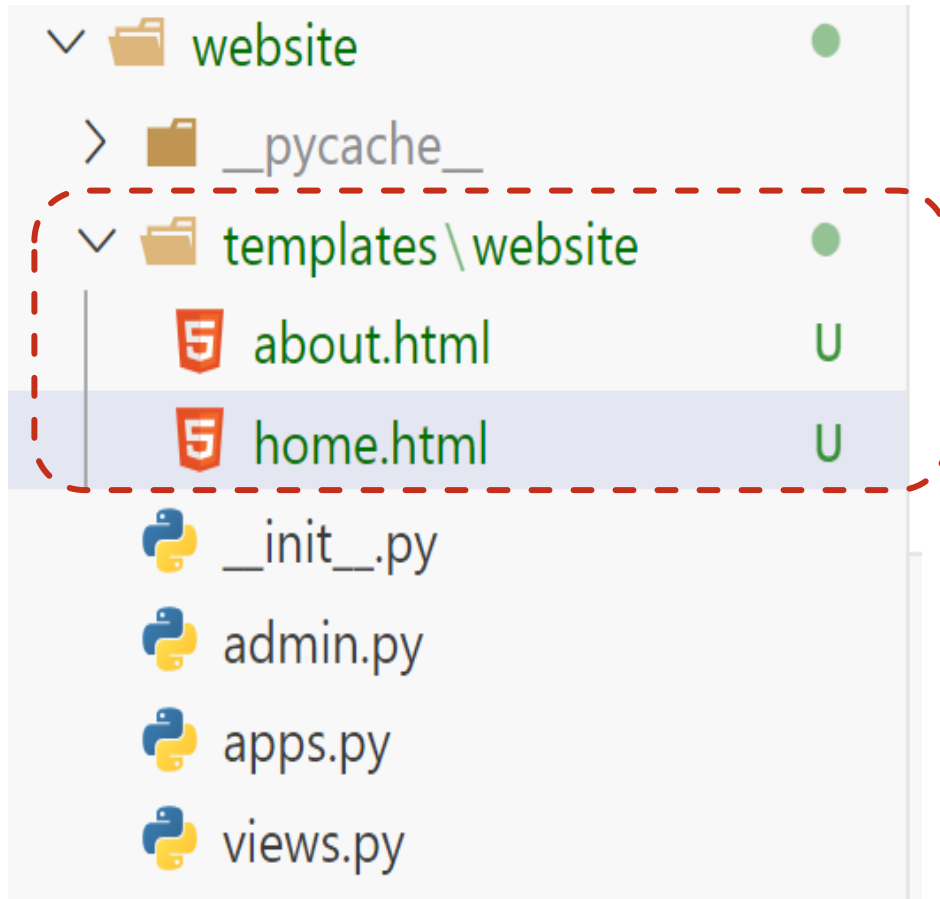
(my_env) \$ python manage.py migrate

démarrer le projet

(my_env) \$ python manage.py runserver

website: home.html & about.html

Étape par Étape



website: home.html

Étape par Étape

```
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Meeting planer: home page</title>
</head>
<body>
  <h2>Meeting planner application from Tek-up</h2>
  <h3>Lite des réunions</h3>
</body>
</html>
```

website: about.html

Étape par Étape

```
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Meeting planer: about page</title>
</head>
<body>
  <h2>Meeting planner application from Tek-up</h2>

</body>
</html>
```

website: views.py

```
from django.shortcuts import render

# Create your views here.

def home_view(request):
    return render(request, "website/home.html")

def about_view(request):
    return render(request, "website/about.html")
```

website: urls.py

```
from django.urls import path

from . import views

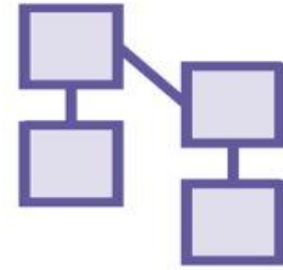
#domain.com/website/...
urlpatterns=[
    path('',views.home_view, name='home'),
    path('about',views.about_view, name='about'),
]
```

meetings_planner: urls.py

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

urlpatterns = [
    path('admin/', admin.site.urls),
    path('website/', include('website.urls')), #new
]
```

Créer le modèle



Django Models

Enregistrer les objets Python dans la BD
Les classes Models sont mappés à des tables
Les attributs sont mappés à des colonnes
SQL est généré
Create/Update tables (migrations)
Insert/Update/Delete lignes (admin)

Models

- Documentation: <https://docs.djangoproject.com/fr/4.2/topics/db/models/>

Démarche: Models

- ▶ Créer les classes modèles
- ▶ Créer les migrations
- ▶ appliquer les migrations
- ▶ Gérer le modèle avec Admin-interface

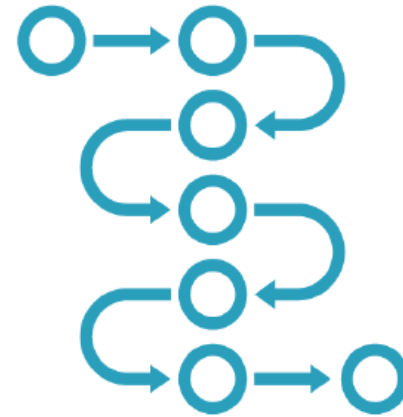
Étape par Étape

Migrations



Models

Classes Python
Associées aux tables de
la BD



Migrations

Ensemble de scripts
décrivant le schéma de
la BD

Étape par Étape

Show migrations

Étape par Étape

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
(.venv) PS C:\pc-nizar\Tek-up\AU2023-2024-S1\django\projects\meeting> python .\manage.py showmigrations
admin
[ ] 0001_initial
[ ] 0002_logentry_remove_auto_add
[ ] 0003_logentry_add_action_flag_choices
auth
[ ] 0001_initial
[ ] 0002_alter_permission_name_max_length
[ ] 0003_alter_user_email_max_length
[ ] 0004_alter_user_username_opts
[ ] 0005_alter_user_last_login_null
[ ] 0006_require_contenttypes_0002
[ ] 0007_alter_validators_add_error_messages
[ ] 0008_alter_user_username_max_length
[ ] 0009_alter_user_last_name_max_length
[ ] 0010_alter_group_name_max_length
[ ] 0011_update_proxy_permissions
[ ] 0012_alter_user_first_name_max_length
contenttypes
[ ] 0001_initial
[ ] 0002_remove_content_type_name
sessions
[ ] 0001_initial
website
```

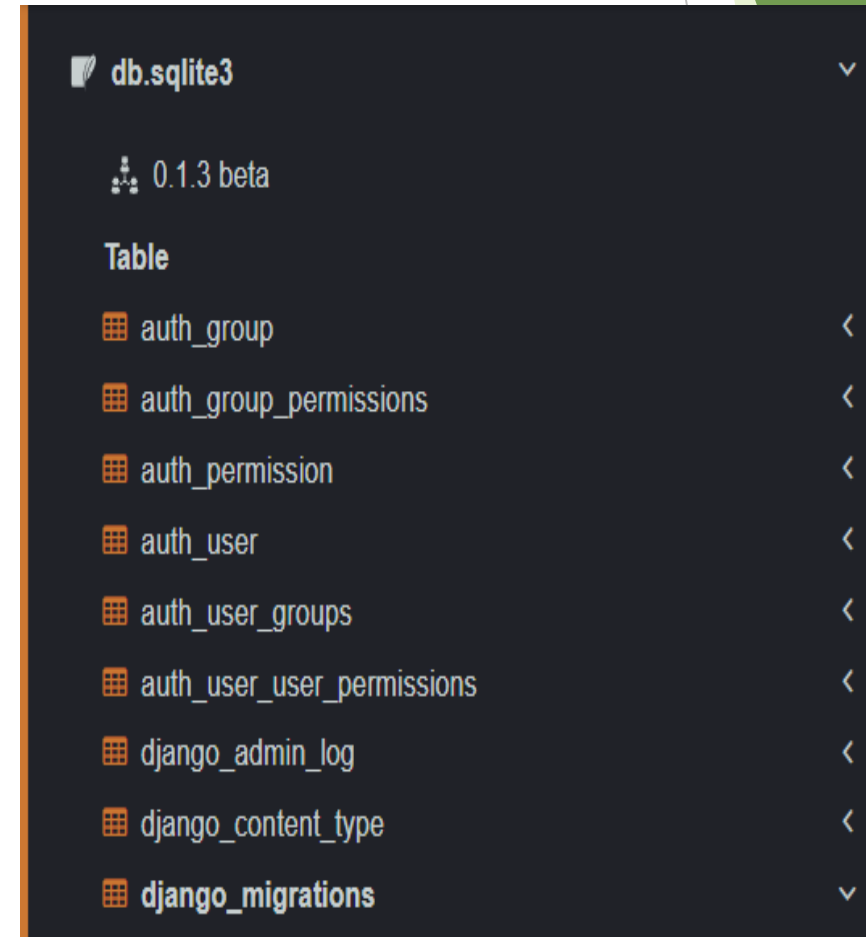
Appliquer les migrations initiales

Étape par Étape

appliquer les migrations

(my_env) \$ python manage.py migrate

Afficher le schéma de la BD: <https://sqliteonline.com/>



Implémenter la classe modèle Meeting

Étape par Étape

meetings.models.py

```
from django.db import models

# Create your models here.

class Meeting(models.Model):
    title=models.CharField(max_length=200)
    date=models.DateField()
```

Créer et appliquer la migration

Étape par Étape

créer la migration

(my_env) \$ python manage.py makemigrations

afficher SQL de la migration

(my_env) \$ python manage.py sqlmigrate meetings 0001

appliquer la migration

(my_env) \$ python manage.py migrate meetings

<https://sqliteonline.com/>

Migration Workflow

Step 1: Change Model code

Step 2: Generate migration script (check it!)

```
python manage.py makemigrations
```

Optional: Show migrations

```
python manage.py showmigrations
```

Optional: Show SQL for specific migration

```
python manage.py sqlmigrate appname migrationname
```

Step 3: Run migrations

```
python manage.py migrate
```

Django Admin

Deux parties de toute application web



End-users



Admin

django Admin

CRUD Model

Gestion Sécurité

Admin panel: éditer les modèles

Étape par Étape

- ▶ Enregistrer le modèle dans **admin.py**
- ▶ Créer un **superuser** pour pouvoir administrer l'application
- ▶ Démarrer l'application et connecter en tant que **superuser**
- ▶ Gérer le modèle

meetings.admin.py: enregistrar Meeting

Étape par Étape

```
from django.contrib import admin  
  
from .models import Meeting  
  
admin.site.register(Meeting)
```

Admin interface

Étape par Étape

127.0.0.1:8000/admin/login/?next=/admin/

Livre_ENI tekup Courrier - Nizar MA...

Django administration

Username:

Password:

Log in

Admin interface: créer superuser

Étape par Étape

```
# créer superuser
```

```
(my_env) $ python manage.py createsuperuser
```

Admin interface: ajouter des meetings

Étape par Étape

← → ↻ 🏠 ⓘ 127.0.0.1:8000/admin/meetings/meeting/add/

★ Bookmarks 📁 Livre_ENI 📁 tekup 📧 Courrier - Nizar MA...

Django administration

Home > Meetings > Meetings > Add meeting

Start typing to filter...

AUTHENTICATION AND AUTHORIZATION

Groups + Add

Users + Add

MEETINGS

Meetings + Add

Add meeting

Title:

Date: Today | 📅

Note: You are 2 hours ahead of server time.

Admin interface: ajouter des meetings

Étape par Étape

← → ↻ 🏠 ⓘ 127.0.0.1:8000/admin/meetings/meeting/

★ Bookmarks 📁 Livre_ENI 📁 tekup 📧 Courrier - Nizar MA...

Django administration

Home > Meetings > Meetings

Start typing to filter...

AUTHENTICATION AND AUTHORIZATION

Groups + Add

Users + Add

MEETINGS

Meetings + Add

Select meeting to change

Action: 0 of 2 selected

<input type="checkbox"/>	MEETING
<input type="checkbox"/>	Meeting object (2)
<input type="checkbox"/>	Meeting object (1)

2 meetings

Exercice: Mettre à jour Meeting

```
from django.db import models
from datetime import time
```

```
# Create your models here.
```

```
class Meeting(models.Model):
    title=models.CharField(max_length=200)
    date=models.DateField()
    start_time=models.TimeField(default=time(9))
    duration=models.IntegerField(default=1)

    def __str__(self):
        return f"{self.title} at {self.start_time} on {self.date}"
```

Créer et appliquer la migration

Étape par Étape

Étape par Étape

Exercice: Ajouter le modèle Room

```
class Room(models.Model):
    name = models.CharField(max_length=50)
    floor = models.IntegerField()
    room_number = models.IntegerField()

    def __str__(self):
        return f"{self.name}: room {self.room_number} on floor {self.floor}"
```

Étape par Étape

Ajouter association one to many

```
class Meeting(models.Model):
    title = models.CharField(max_length=200)
    date = models.DateField()
    start_time = models.TimeField(default=time(9))
    duration = models.IntegerField(default=1)
    room = models.ForeignKey(Room, on_delete=models.CASCADE)

    def __str__(self):
        return f"{self.title} at {self.start_time} on {self.date}"
```

Créer et appliquer la migration

Étape par Étape

Étape par Étape

meetings.views

```
def detail(request, id):
    meeting = Meeting.objects.get(pk=id)
    return render(request, "meetings/detail.html", {"meeting": meeting})
```

```
def detail(request, id):
    meeting = get_object_or_404(Meeting, id)
    return render(request, "meetings/detail.html", {"meeting": meeting})
```

Étape par Étape

website.views

```
def home_view(request):
    context={'nbre_meeting': Meeting.objects.count()}
    return render(request, "website/home.html", context=context)

def about_view(request):
    return render(request, "website/about.html")
```

Templates

Générer les pages web

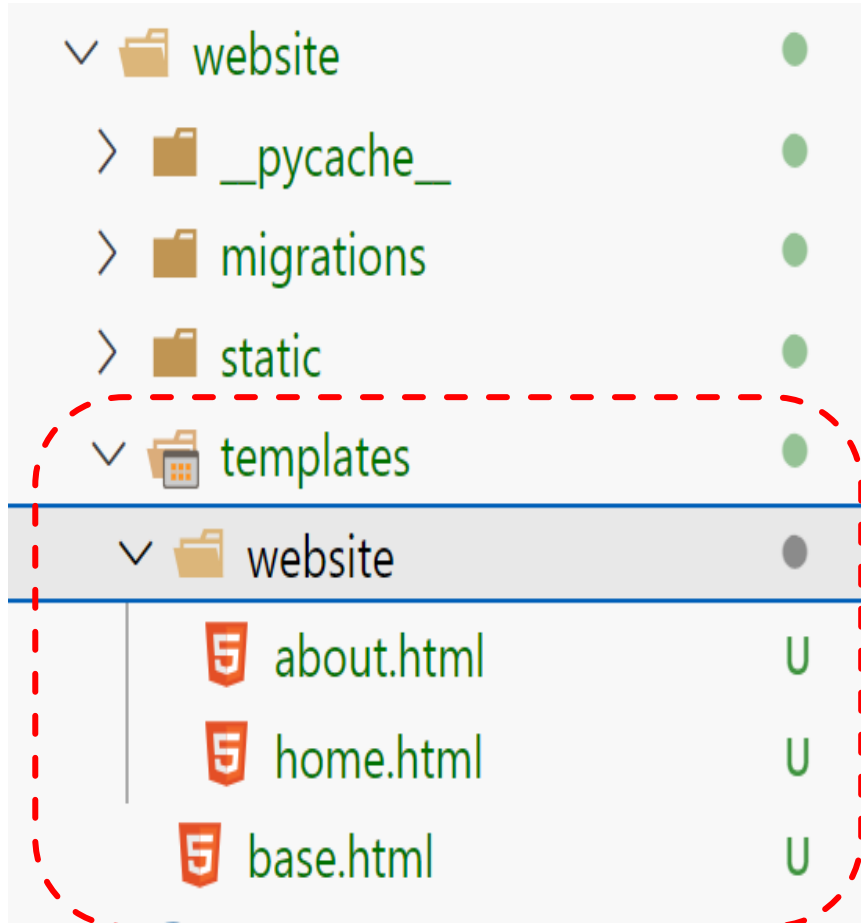
Template variables

Template tags

Template inheritance

Templates: héritage

Étape par Étape



Framework django

Étape par Étape

Templates: héritage (base.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">

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

</head>
<body>
  {% block content %}
  {% endblock %}
</body>
</html>
```

Étape par Étape

Templates: héritage (about.html)

```
{% extends "base.html" %}

{% block title %}about{% endblock %}

{% block content %}
    <h2>Meeting planner application from Tek-up</h2>

{% endblock %}
```

Étape par Étape

website: home.html v1

```
{% extends "base.html" %}

{% block title %}HOME{% endblock %}

{% block content %}
    <h2>Welcome to the meeting planner</h2>
    <p>there are currently {{nbre_meeting}} meetings</p>
{% endblock %}
```

Étape par Étape

website: home.html v2

```
<h2>Meetings</h2>
  <ul>
    {% for meeting in meetings %}
      <li>
        <a href="{% url 'detail' meeting.id %}">
          {{ meeting.title }}
        </a>
      </li>
    {% endfor %}
  </ul>
```

Étape par Étape

website: home.html v3

```
<h2>Meetings</h2>
  <ul>
    {% for meeting in meetings %}
      <li>
        <a href="{% url 'detail' meeting.id %}">
          {{ meeting.title }}
        </a>
      </li>
    {% endfor %}
  </ul>
  <a href="{% url 'rooms' %}">Rooms list</a>
```

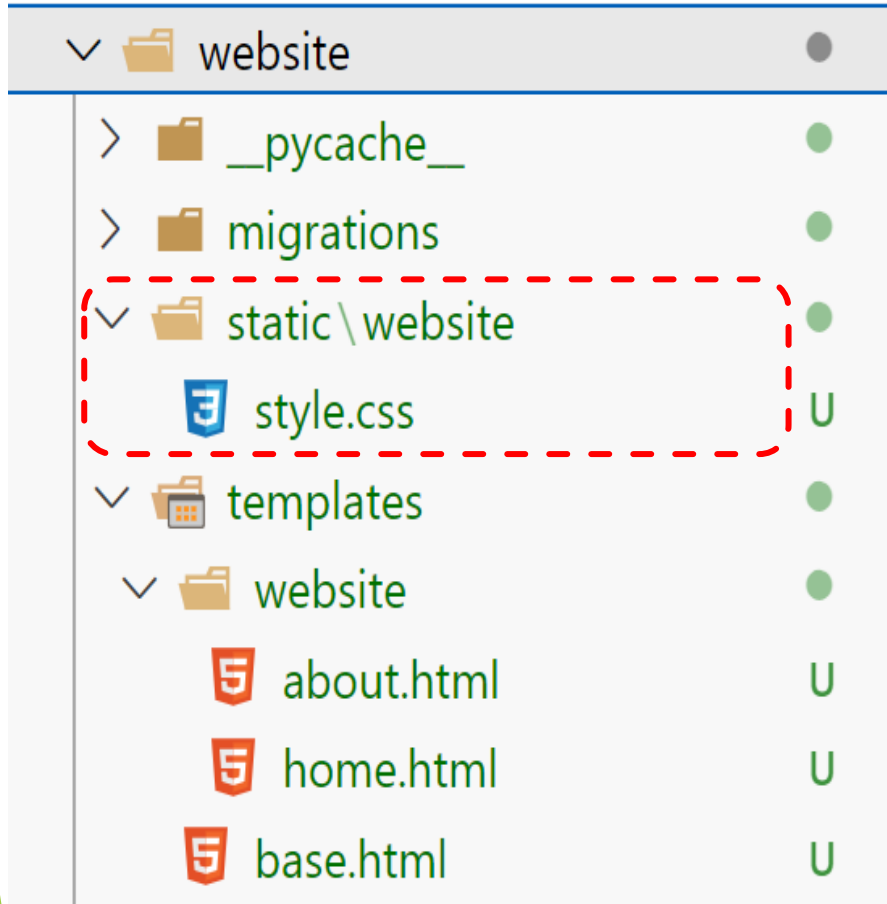
Étape par Étape

website.views v3

```
def home_view(request):
    context={'meetings': Meeting.objects.all()}
    return render(request, "website/home.html", context=context)
```

Templates: static files

Étape par Étape



```
body {
    font-family: sans-serif;
    color: cornflowerblue;
    background-color: floralwhite;
}
```

Étape par Étape

Templates: static files (settings.py)

```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/4.2/howto/static-files/

STATIC_URL = 'static/'
```

Étape par Étape

Templates: static files (base.html)

```
{% load static %}

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{% block title %}{% endblock %}</title>
    <link rel="stylesheet"
        href="{% static 'website/style.css' %}">
</head>
<body>
    {% block content %}
    {% endblock %}
</body>
</html>
```

Exercice

- ▶ Ajouter une page qui liste tous les rooms:
 - ▶ Views
 - ▶ template
 - ▶ url mapping

django forms

Valider les entrées utilisateurs

Automatise le code répétitif
sécurisé

Hautement personnalisé

Étape par Étape

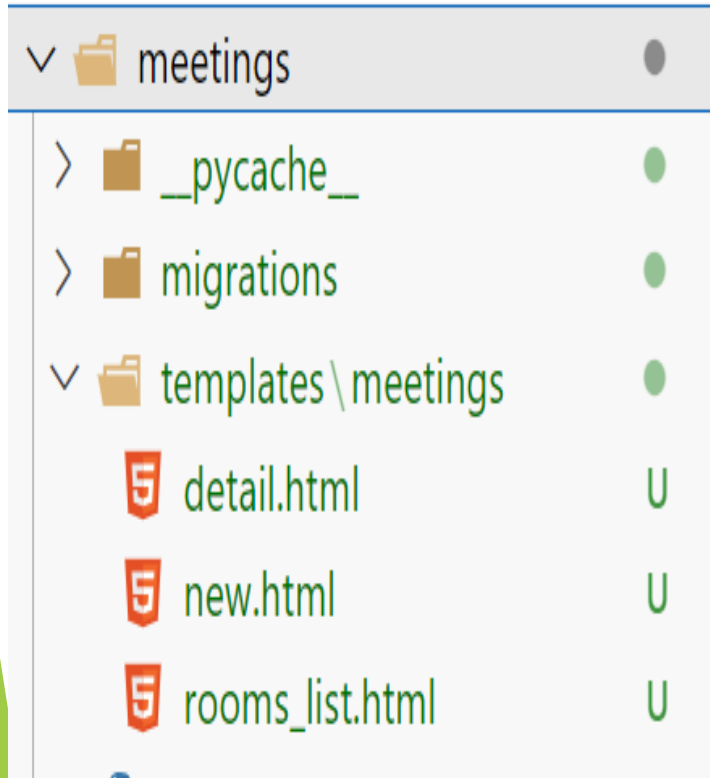
Formulaire: Meeting

```
class MeetingForm(ModelForm):
    class Meta:
        model = Meeting
        fields = '__all__'
        widgets = {
            'date': DateInput(attrs={"type": "date"}),
            'start': TimeInput(attrs={"type": "time"}),
            'duration': TextInput(attrs={"type": "number", "min": "1", "max": "4"})
        }

    def clean_date(self):
        d = self.cleaned_data.get("date")
        if d < date.today():
            raise ValidationError("Meetings cannot be in the past")
        return d
```

Formulaire: new.html

Étape par Étape



```
{% extends "base.html" %}
{% block title %}New Meeting{% endblock %}

{% block content %}
<h1>Plan a new meeting</h1>
<form method="post">
    <table>
        {{ form }}
    </table>
    {% csrf_token %}
    <button type="submit">Create</button>
</form>
{% endblock %}
```

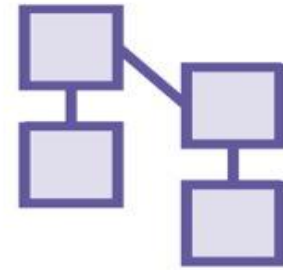
Exercice

- ▶ Ajouter tout le nécessaire pour permettre l'ajout d'une salle de réunion



- ▶ Architecture MVT
- ▶ Models: définition, manipulation, associations
- ▶ Admin-panel: CRUD, sécurité
- ▶ Templates: variables, tags, static files, héritage
- ▶ Forms: mapping entre templates et Models

django Models



Résultats

À la fin de cette séance, vous serez en mesure de:

- ▶ Maîtriser le composant **Models** dans l'architecture **MTV** de **django**
 - ▶ django ORM
 - ▶ django Models
 - ▶ Manipulation des modèles
 - ▶ Migrations, optimisation

Éléments du contenu

- ▶ Modéliser vos données
- ▶ Écrire des requêtes avancées
- ▶ Personnaliser le comportement de vos Models
- ▶ Créer et appliquer des migrations
- ▶ Optimiser les performances

Model-Template-View

Model
Données

Classes Modèle
Mapped to DB

Template
Présentation

Générer HTML

View
Comportement

Fonction python
Mapped to URL

Model

View

Controller

ORM

Object-Relational Mapping

Un autre nom de la couche **Models**

Relational data

- Lignes dans des tables de base de données
- SQL

Python data

- Classes et objets, valeurs et variables

ORM bridges the gaps

- les classes modèles sont mappés à des tables
- SQL est généré
- coder uniquement en python

ORM inconvenients

Moins de contrôle sur SQL

Moins de performances

Mais

- Possible de faire des optimisations avec django ORM
- Possible d'exécuter Raw SQL

django Models

Mapped to DB
tables

Générer UI
(ModelForm)

Valider
Forms

Générer
Admin interface

Ajouter
custom methods

django Models

Bases de données supportées:

- PostgreSQL, MariaDB, MySQL, Oracle, SQLite
- avec packages: DB2, MS SQL, ...

Projet de démonstrations

- ▶ Application web de e-commerce (gestion des produits)
- ▶ Focaliser le développement dans la couche Model
- ▶ Utiliser deux BD:
 - ▶ SQLite
 - ▶ PostgreSQL

Création du projet

Étape par Étape

#créer un dossier pour le projet

```
$ cd framework_django\projects\django_models
```

```
$ mkdir e_commerce
```

```
$ cd e_commerce
```

#créer et activer l'environnement virtuel

```
$ python -m venv my_env
```

```
$ my_env\Scripts\Activate
```

```
(my_env) $ python -m pip install django
```

Sauvegarder les dépendances

#sauvegarder les dépendances

(my_env) \$ pip freeze > requirements.txt

```
requirements.txt X
requirements.txt
1  asgiref==3.7.2
2  Django==4.2.4
3  sqlparse==0.4.4
4  tzdata==2023.3
5
```

Création du projet et applications

Étape par Étape

créer le projet e_commerce_site

(my_env) \$ django-admin startproject e_commerce_site .

lancer le projet dans VS code

(my_env) \$ code .

créer l'application products

(my_env) \$ python manage.py startapp products

settings.py: installer l'application

Étape par Étape

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'products.apps.ProductsConfig', #new
]
```

Installation de PostgreSQL

- <https://www.postgresql.org/download/>



settings.py: Configuration du Log

Étape par Étape

```
#logging all SQL
LOGGING={
    "version": 1,
    'handlers': {
        'console': {
            'level': 'DEBUG',
            'class': 'logging.StreamHandler',
        }
    },
    'loggers': {
        'django.db.backends': {
            'level': 'DEBUG',
            'handlers': ['console'],
        }
    }
}
```



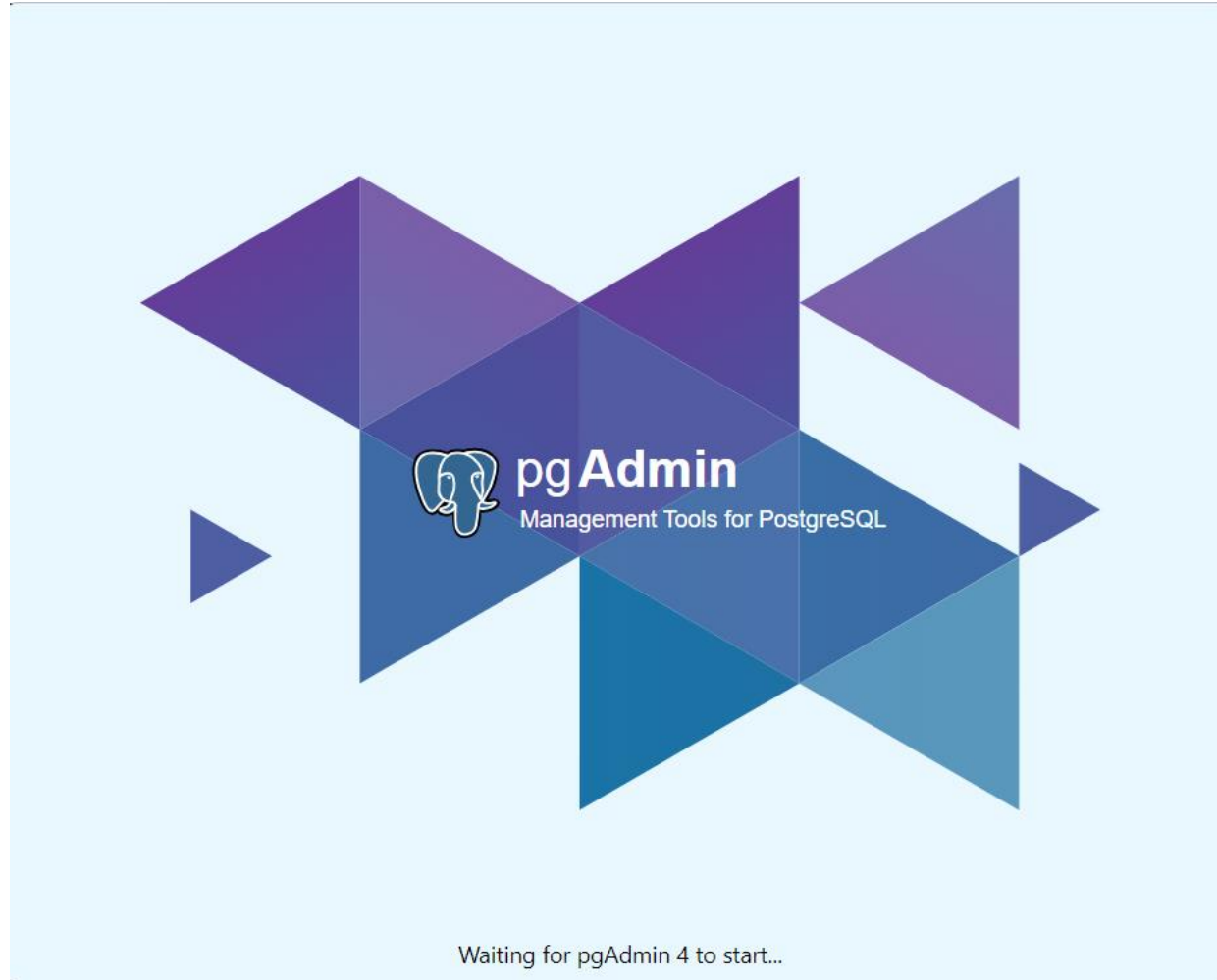
Étape par Étape

settings.py: configuration du

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.postgresql',  
        'NAME': 'products_db',  
        'HOST': '127.0.0.1',  
        'USER': 'postgres',  
        'PASSWORD': 'adminadmin',  
    }  
}
```

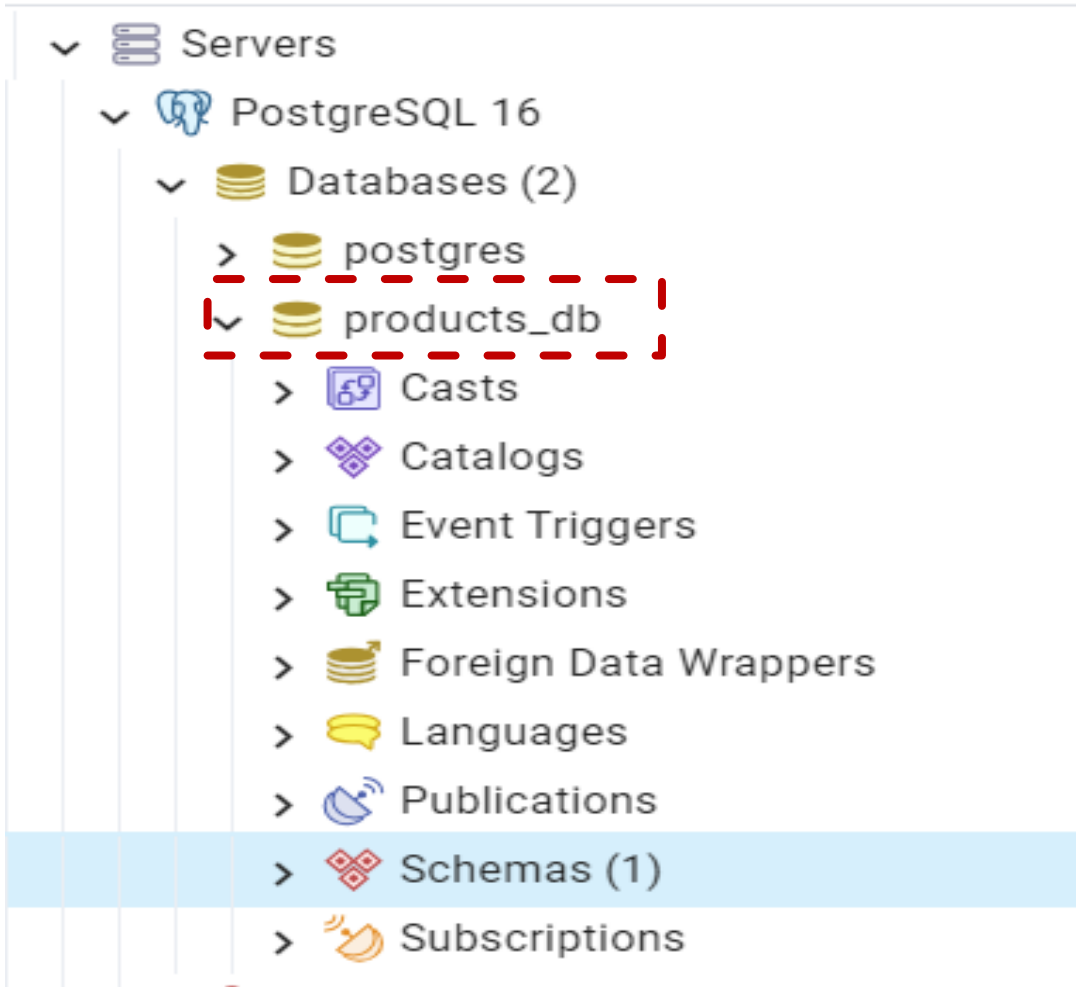
PostgreSQL: créer la base de données

Étape par Étape



PostgreSQL: créer la base de données

Étape par Étape



Framework django

Installer psycopg2

```
python -m pip install psycopg2
```

```
pip freeze > requirements.txt
```

Étape par Étape

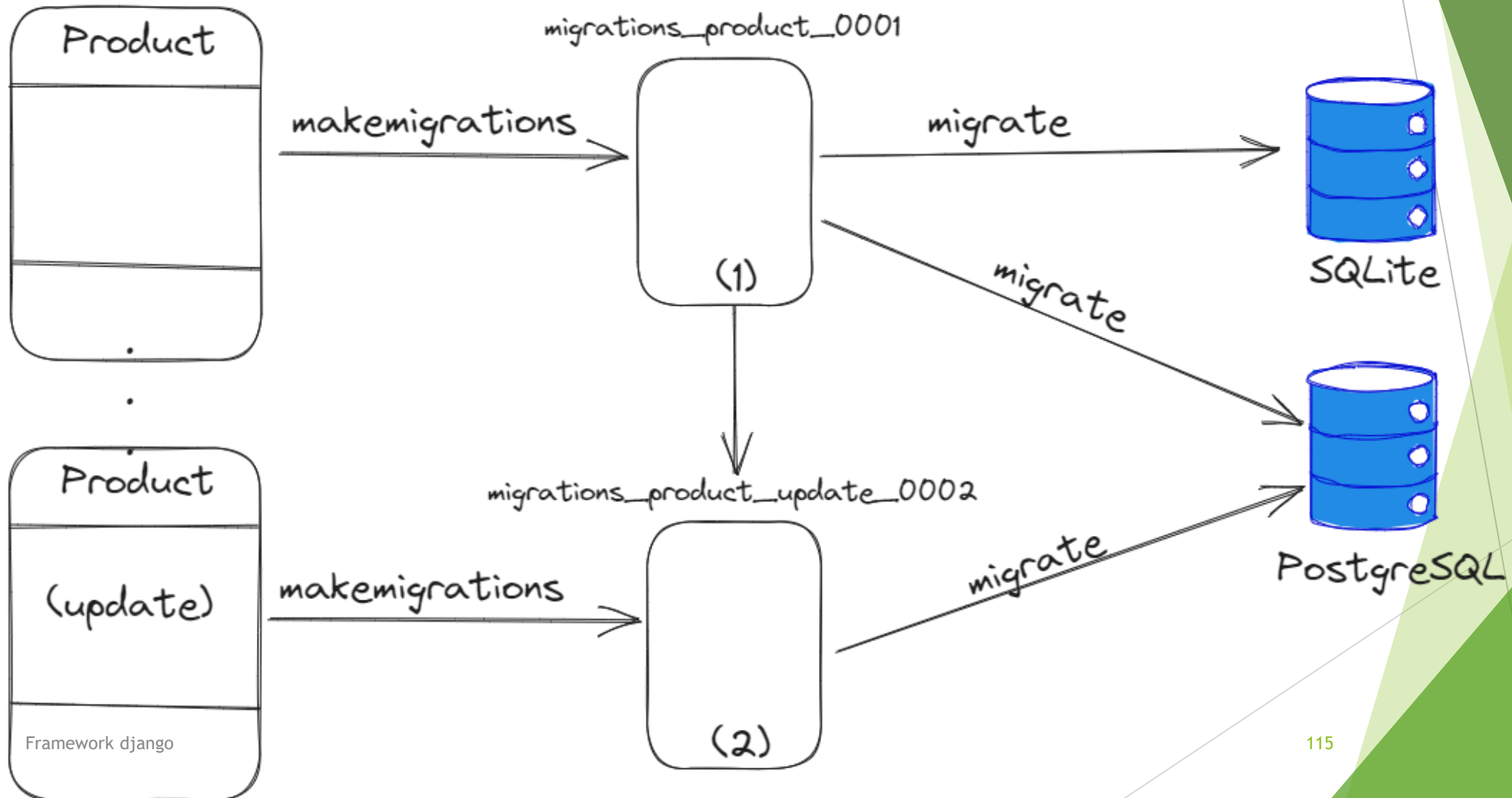
django Models: classe et instance

Démarche

- ▶ Créer classe Modèle
- ▶ Créer et appliquer les migrations
- ▶ Manipulation des instances du modèle:
 - ▶ Create
 - ▶ Update
 - ▶ Delete
- ▶ Différents backend

Étape par Étape

Models et Migrations

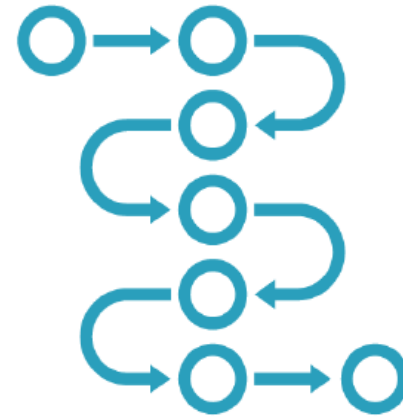


Migrations



Models

Classes Python
Associées aux tables de
la BD



Migrations

Ensemble de scripts
décrivant le schéma de
la BD

Migration Workflow

Important: Make sure your app is in INSTALLED_APPS

Step 1: Change model code

Step 2: Generate migration script (check it!)

```
python manage.py makemigrations
```

Step 3: Run migrations

```
python manage.py migrate
```

products.models.py: modèle Product

Étape par Étape

```
from django.db import models
```

```
class Product(models.Model):
    name=models.CharField(max_length=100)
    stock_count=models.IntegerField(default=0)
    price=models.DecimalField(max_digits=6,decimal_places=2)
```

settings.py: installer l'application

Étape par Étape

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'products.apps.ProductsConfig', #new
]
```

Créer et appliquer la migration

Étape par Étape

créer migrations

(my_env) \$ python manage.py makemigrations

afficher migrations

(my_env) \$ python manage.py sqlmigrate products 0001

appliquer les migrations

(my_env) \$ python manage.py migrate

Activité

- ▶ Changer la configuration du projet django pour se connecter à la base SQLite.
- ▶ Appliquer les migrations
- ▶ Interpréter le résultat: comparer les requêtes générées avec **PostgreSQL** et **SQLite**

Model Product: store data

Étape par Étape

lancer une console interactive

```
(my_env) $ python manage.py shell
```

Créer et sauvegarder un produit

```
>>> from products.models import Product
>>> p= Product(name="Clavier", stock_count=50, price=30)
>>> p.id
>>> p.id is None
>>> p.save()
>>> p.id
```

Model Product: update data

```
# update product
```

```
>>> p.price=40
```

```
>>> p.save()
```

Étape par Étape

Model Product: delete data

```
# delete product
```

```
>>> p.delete()
```

django Models: Fields

Éléments du contenu

- ▶ Model Fields
 - ▶ Type des Fields et options
 - ▶ Effet sur les formulaires et validation
- ▶ Relations:
 - ▶ ForeignKey
 - ▶ OneToOne
 - ▶ ManyToMany
 - ▶ Créer des relations entre les objets

```
class Person(models.Model):
    name = models.CharField(max_length=100)
    age = models.IntegerField()
```

Model Fields

Class attributes mapped to DB columns

Doivent être instances des **classes Field**

Exemple: CharField, IntegerField,...

Model Fields

Les Classes Fields déterminent:

- types de colonnes dans la BD (Integer, varchar,...)
- type de widget dans le formulaire

Field Options

- Validation au niveau de la BD
- Formulaire et validation
- valeurs par défaut
- . . .

documentation

- <https://docs.djangoproject.com/en/4.2/ref/models/fields/>

Storing Numbers

BooleanField

IntegerField
(et variantes)

FloatField

DecimalField

Storing Text

CharField

HTML: input text

Required
max_length

TextField

Text plus large

HTML: TextArea

EmailField

URLField

FilePathField

SlugField

Other Common Field Types

DateField

TimeField

DateTimeField

DurationField

FileField

ImageField

JSONField

BinaryField

Démo: TextFields

Étape par Étape

```
class Product(models.Model):
    name=models.CharField(max_length=100)
    stock_count=models.IntegerField(default=0)
    price=models.DecimalField(max_digits=6,decimal_places=2)
    description=models.TextField(default="")
```

migrations

rollback products migrations

(my_env) \$ python manage.py migrate products zero

créer migrations

(my_env) \$ python manage.py makemigrations

appliquer les migrations

(my_env) \$ python manage.py migrate

products.admin.py

```
from django.contrib import admin  
from .models import Product
```

```
# Register your models here.
```

```
admin.site.register(Product)
```

Étape par Étape

Admin interface

Add product

Name:

Stock count:

Price:

Description:

SAVE

Save and add another

Save and continue editing

Étape par Étape

Options: Null et Blank

- ▶ Ajouter un nouveau produit sans description
- ▶ Interpréter le résultat
- ▶ Par défaut, tous les champs n'acceptent pas la valeur **null**
- ▶ Option **null**: validation **coté BD**
- ▶ Option **blank**: autorisation de valeur vide **coté formulaire**

Options: blank et null

```
class Product(models.Model):
    name=models.CharField(max_length=100)
    stock_count=models.IntegerField(default=0)
    price=models.DecimalField(max_digits=6,decimal_places=2)
    description=models.TextField(default="",blank=True)
```

Model Field Options 1

Make Field Nullable (default is non-NULL)

```
models.IntegerField(null = True)
```

Allow empty values in forms (Not db-related!)

```
models.CharField(blank = True)
```

Default value

```
models.CharField(default = 'Example')
```

Model Field Options 2

Add unique constraint

```
models.CharField(unique = True)
```

Add an index

```
models.IntegerField(db_index = True)
```

Set column name

```
models.BooleanField(db_column = "my_column_name")
```

Type-specific options

```
models.DateTimeField(auto_now = True)
```

Model Field Options 3

Set field label

```
iban = models.CharField(verbose_name = "Bank Account", ...)
```

Additional help text

```
name = models.CharField(help_text = "Enter your full name")
```

Model Field: Exercice

- ▶ Ajouter un Field **sku**:
 - ▶ Un code unique pour chaque produit
 - ▶ Chaîne de caractère de taille maximale 20.
 - ▶ Le label du formulaire doit afficher **stock keeping unit** au lieu de sku
- ▶ Appliquer migration workflow

Model Field: Exercice Solution

```
class Product(models.Model):

    name=models.CharField(max_length=100)

    stock_count=models.IntegerField(default=0,help_text="how many items are currently in stock")

    price=models.DecimalField(max_digits=6,decimal_places=2)

    description=models.TextField(default="",blank=True)

    sku=models.CharField(verbose_name="Stock Keeping Unit", max_length=20, unique=True)
```

migrations

rollback products migrations

(my_env) \$ python manage.py migrate products zero

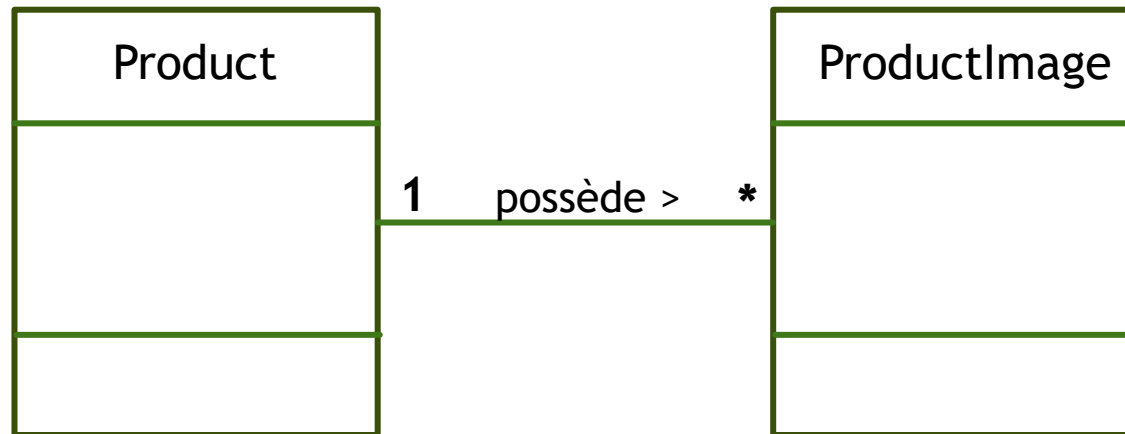
créer migrations

(my_env) \$ python manage.py makemigrations

appliquer les migrations

(my_env) \$ python manage.py migrate

Model Relations: One-To-Many



Définir le modèle ProductImage

Étape par Étape

```
class ProductImage(models.Model):
    image=models.ImageField()
    product=models.ForeignKey('Product',on_delete=models.CASCADE)
```

Install pillow package for ImageField

Étape par Étape

- > `python -m pip install Pillow`
- > `pip freeze > requirements.txt`

Étape par Étape

make and run migrations

- # rollback products migrations
(my_env) \$ python manage.py migrate products zero
- # créer migrations
(my_env) \$ python manage.py makemigrations
- # appliquer les migrations
(my_env) \$ python manage.py migrate

products.admin.py: register ProductImage

Étape par Étape

```
from django.contrib import admin
from .models import Product, ProductImage

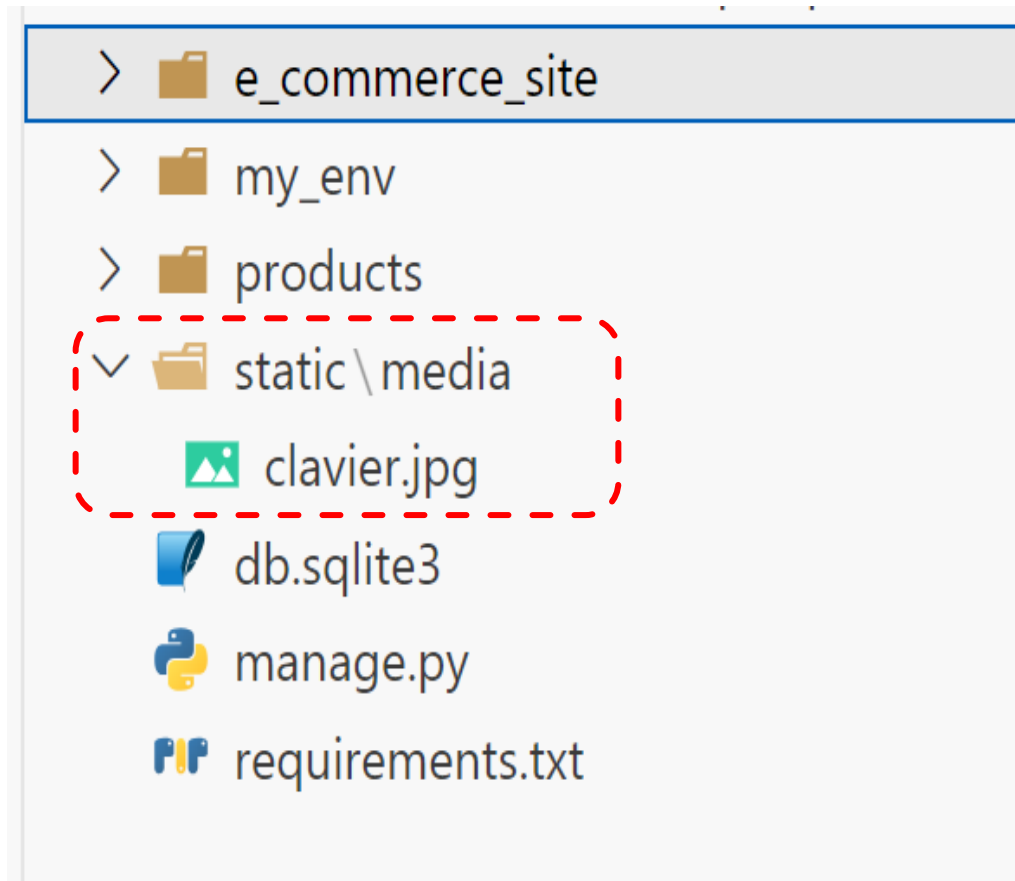
# Register your models here.

admin.site.register(Product)
admin.site.register(ProductImage)
```

settings.py: configuration media root

```
STATIC_ROOT = BASE_DIR / 'static'  
MEDIA_ROOT = STATIC_ROOT / 'media'  
MEDIA_URL = '/media/'
```

Créer le dossier static/media



urls.py: configuration media urls

```
from django.contrib import admin
from django.urls import path
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    path('admin/', admin.site.urls),
    static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
]
```

```
product=models.ForeignKey(

    'Product',

    on_delete=models.CASCADE

)
```

← One-to-many relation
Created by **ForeignKey** field on the “many” side

← Target Model class for relation
Best practice: use name of target as a string

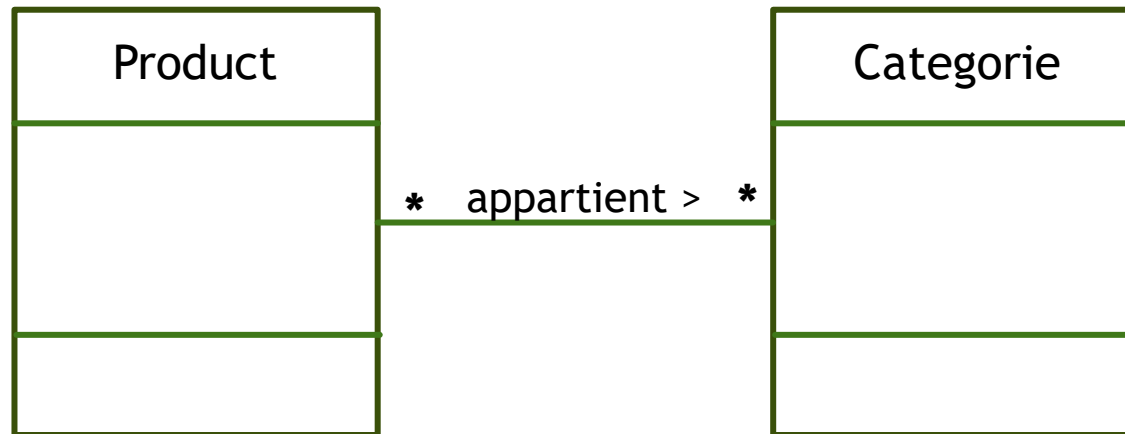
← Required: **on_delete**
Specify behaviour when related object is deleted

See:

<https://docs.djangoproject.com/en/4.2/ref/models/fields/#django.db.models.ForeignKey>

Similar: **OneToOneField**
Like **ForeignKey** with **unique=True**

Model Relations: Many-To-Many



products.models.py: Categorie Model

```
class Categorie(models.Model):  
    name=models.CharField(max_length=100)  
    products=models.ManyToManyField('Product')
```

Étape par Étape

make and run migrations

- # rollback products migrations
(my_env) \$ python manage.py migrate products zero
- # créer migrations
(my_env) \$ python manage.py makemigrations
- # appliquer les migrations
(my_env) \$ python manage.py migrate

products.admin.py: register Categorie

```
from django.contrib import admin
from .models import Product, ProductImage,
Categorie
```

```
# Register your models here.
```

```
admin.site.register(Product)
admin.site.register(ProductImage)
admin.site.register(Categorie)
```

```
class Category(models.Model):

    products = models.ManyToManyField(

        'Product'

    )
```

- ◀ **Many-to-many relation**
 - Created by ManyToManyField
 - Can be on either side of relation
 - Join table in database

- ◀ **Target Model class for relation**
 - Best practice: use name of target as a string

django Models: Managers et QuerySets

Éléments du contenu

- ▶ Créer des requêtes
 - ▶ Managers
 - ▶ QuerySets
 - ▶ filter() et exclude()
 - ▶ Laziness
 - ▶ Limiter et ordonner les résultats
 - ▶ Relations
 - ▶ Aggrégations
 - ▶ F() et Q()

Managers

Product.objects

- ▶ Un **gestionnaire** (objet **Manager**) est l'interface par laquelle les opérations de requêtes de base de données sont mises à disposition aux modèles django.
- ▶ il existe au moins un **Manager** pour chaque modèle.
- ▶ par défaut, django ajoute un **Manager** nommé **objects** à chaque modèle.

Manger et QuerySet

chaque classe Modèle a un Manager: **objects**

On utilise le **Manager** pour exécuter des requêtes sur des tables

```
Product.objects.count()
```

```
Product.objects.get(pk=1)
```

Many methods return a QuerySet instance

A QuerySet represents a database query

```
claviers = Product.objects.filter(category__name="Clavier")
```

Managers et QuerySets

Toutes les méthodes **QuerySet** sont disponibles dans le **Manager**

Exemples: `objects.count()`, `objects.filter()`, ...

Le **Manager** **a** un **QuerySet** **interne**

Les appels de méthodes **sont passés** à ce **QuerySet**

Laziness

QuerySets are **lazy**. The following **does NOT run** any SQL

```
in_stock = Product.objects.filter(stock_count__gt=0)
```

This allows **chaining** of filters

```
claviers_in_stock = in_stock.filter(category__name="Clavier")
```

```
moin_chers_claviers = claviers_in_stock.order_by("price")[:5]
```

Laziness

What will cause a QuerySet to run its SQL?

`str(in_stock)` or `template: {{ in_stock }}`

for `product` in `in_stock`:

...

`list(in_stock)`

filter() et exclude()

```
Product.objects.filter(name__contains="a")
```

```
Product.objects.exclude(stock_count=0, price=10)
```

Select 0 or more rows from the database

Multiple arguments are combined with **AND**

Return a QuerySet

get(): selecting a Single row

Select with primary key

```
Product.objects.get(pk=5)
```

```
Product.objects.get(id=5)
```

With a unique column

```
Product.objects.get(sku="abc123z")
```

Args work similar to filter()

Raises exception if not exactly 1 match

In views

```
get_object_or_404(Product, pk=5)
```

Lookups

Allow specifying more complex **WHERE** clauses with `get()`, `filter()`, `exclude()`

Syntax: `field__lookuptype=value`

```
in_stock = Product.objects.filter(stock_count__gt=0)
```

Lookup across relations

```
claviers_in_stock = in_stock.filter(category__name="Clavier")
```

Combining both: get all images of claviers that are in stock

```
ProductImage.objects.filter(product__category__name="Clavier", product__stock_count__gt=0)
```

Limiting and Ordering

Use order_by and reverse() to sort results

```
Product.objects.order_by("price")
```

```
Product.objects.order_by("-price")
```

```
Product.objects.order_by("name", "-price")
```

```
Product.objects.order_by("category__name").reverse()
```

Limit results by slicing

```
Product.objects.order_by("price")[:10]
```

Links

- ▶ All QuerySets methods: <https://docs.djangoproject.com/en/4.2/ref/models/queries/>
- ▶ All Lookups: <https://docs.djangoproject.com/en/4.2/ref/models/queries/#field-lookups>

Related Manager

lancer une console interactive

(my_env) \$ python manage.py shell

Related Manager

```
>>> from products.models import Category
```

```
>>> c= Category.objects.get(pk=1)
```

```
>>> c.products #related Manager object
```

```
>>> c.products.all()
```

```
>>> c.products.filter(price__lt=10)
```

Related Manager

Related Manager

```
>>> from products.models import Product
>>> p= Product.objects.get(pk=1)
>>> p.category_set #related Manager object
>>> p.category_set.all()
```