TP 1

Université Toulouse Paul Sabatier - KEAT9AA1

2024-09-25

Guilhem Saurel

This presentation

Available at

https://homepages.laas.fr/gsaurel/teach/2024-2025/M2_ISTR/a-tp-1.pdf

Under License

https://creativecommons.org/licenses/by-sa/4.0/

This presentation (continued)

Source

https://gitlab.laas.fr/gsaurel/teach : 2024-2025/M2_ISTR/a-tp-1.md

Contact

Matrix: @gsaurel:laas.fr
Mail: gsaurel@laas.fr

Séance 1

Pre-requis

- Git: **`git --version`**
- Python: **`python -V`** (**`python3`** ?)
- Pip: **`python -m pip -V`**
- Venv: **`python -m venv`**

Création d'un projet

```
$ mkdir tp_coo
$ cd tp_coo
$ git init
$ python -m venv .venv
$ echo .venv >> .gitignore
$ source .venv/bin/activate
$ pip install -U pip
$ pip install django
$ django-admin startproject crayon
```

Configuration d'un dépôt distant

- Créer un compte puis un dépôt sur github:
  - sans fichier README.md
  - sans license
  - sans .gitignore

```
$ ssh-keygen -t ed25519
$ cat ~/.ssh/id_ed25519.pub
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```

https://github.com/settings/ssh/new

Publication du projet

```
$ git branch -M main
$ git remote add origin \
    git@github.com:votre-nom/votre-dépôt.git
$ git add .
$ git commit -m "start project"
$ git push -u origin main
```

Configuration des outils

```
$ wget https://gitlab.laas.fr/gsaurel/teach\
      /-/raw/main/.pre-commit-config.yaml
$ pip install pre-commit
$ pre-commit install
$ pre-commit run -a
```

Configuration des outils

```
$ wget https://gitlab.laas.fr/gsaurel/teach\
      /-/raw/main/.pre-commit-config.yaml
$ pip install pre-commit
$ pre-commit install
$ pre-commit run -a

$ pre-commit run -a
$ git add .
$ git commit -m "setup tooling"
$ git push
```

License & gitignore

Depuis l'UI github:

- Ajoutez un fichier **`LICENSE`** et choisissez un template
- Éditez **`.gitignore`** et choisissez le template python

puis **`git pull`**

Création d'une application

```
$ cd crayon
$ ./manage.py startapp high_level
# éditez crayon/settings.py:
# ajoutez `high_level` dans `INSTALLED_APPS`
$ git add .
$ git commit -m "start app high_level"
$ git push
```

Modèles

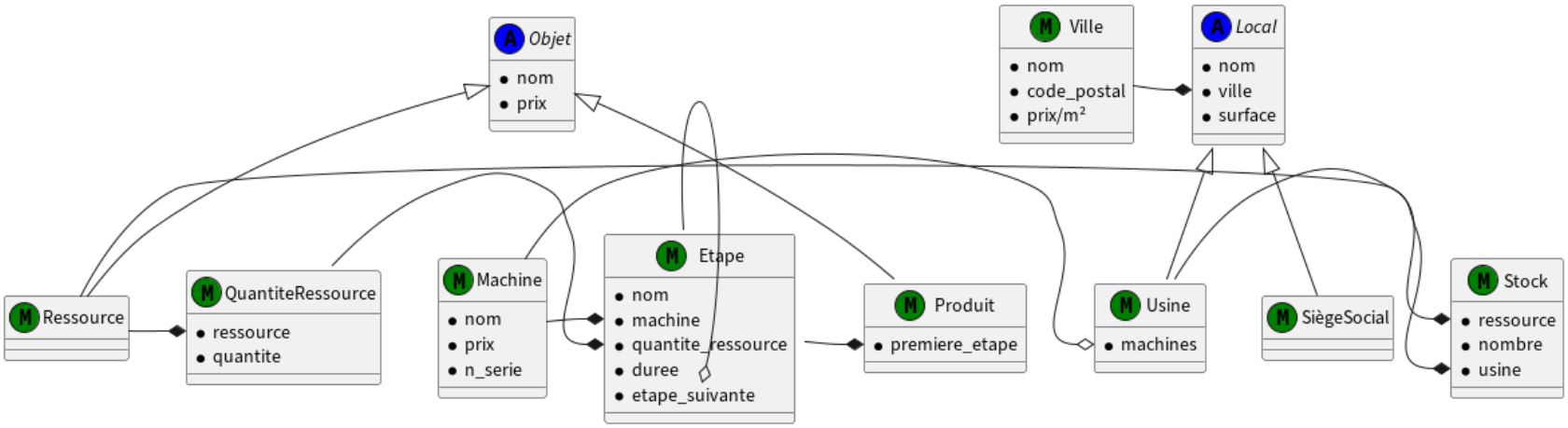

Figure 1: modèles

Création des modèles

Éditez **`high_level/models.py`**, ref:

https://homepages.laas.fr/gsaurel/teach/2024-2025/M2_ISTR/7-api-python.pdf

- nombres: “**`models.IntegerField()`**”
- texte: “**`models.CharField(max_length=100)`**”
- relation vers plusieurs objets: “**`models.ManyToManyField(Machine)`**”
- relation vers un objet:

```
models.ForeignKey(
    Ville,  # ou "self",
    on_delete=models.PROTECT,
    # blank=True, null=True,
    # related_name="+",
)
```

Création des modèles: class Abstraite

```
class Local(models.Model):
    nom = models.CharField(max_length=100)
    ville = models.ForeignKey(
        Ville,
        on_delete=models.PROTECT
    )
    surface = models.IntegerField()

    class Meta:
        abstract = True
```

Création de l'interface d'administration

Éditez **high_level/admin.py**

```
$ ./manage.py makemigrations
$ ./manage.py migrate
$ ./manage.py createsuperuser
$ git add .
$ git commit -m "add high_level models & admin"
$ git push
$ ./manage.py runserver
```

Utilisation de l'interface d'administration

http://localhost:8000/admin

- Ajoutez une méthode "**`__str__(self):`**" dans chaque modèle
- Créez au moins un objet de chaque modèle

Séance 2

On vérifie que tout est dans git

Dans un nouveau dossier:

```
git clone git@github.com:votre-nom/votre-dépôt.git
```

Ligne de commande: requêtes sur un modèle

```
$ pip install ipython
$ ./manage.py shell
```

```
In [1]: from high_level.models import Ville

In [2]: v = Ville.objects.first()

In [3]: v.code_postal
Out[3]: 31444

In [5]: v.usine_set.get(nom="TLS-01").surface
Out[5]: 209
```

Example de fichier de test

Éditez **high_level/tests.py**:

```
from django.test import TestCase

from .models import Machine


class MachineModelTests(TestCase):
    def test_machine_creation(self):
        self.assertEqual(Machine.objects.count(), 0)
        Machine.objects.create(nom="scie",
                               prix=1_000,
                               n_serie=44365)
        self.assertEqual(Machine.objects.count(), 1)
```

```
$ ./manage.py test
```

Calcul des coûts

Implémentez une méthode “**`def costs(self):`**” dans chaque modèle où ça a un sens:

- **`Machine`**
- **`QuantiteIngredient`**
- **`Usine`**
- **`Stock`**

Écrire un test unitaire

Implémentez un scenario de test qui valide le calcul des coûts dans un cas connu. Par exemple:

- une **`Usine`** de 50 m^2
- dans la **`Ville`** Labège à 2 000 €/m^2
- avec une **`Machine`** à 1 000 €, et une autre à 2 000 €
- et en stock
  - 1000 kg de bois à 10 €/kg
  - 50 m de mine à 15 €/m

On s'attend à ce que **`Usine.objects.first().costs()`** vaille 110 750 €

Approvisionnement (facultatif)

- Achetez automatiquement les stocks de l'usine en fonction des recettes choisies

Séance 3

JSON

- Implémentez une méthode "**`def json(self):`**" pour sérialiser chaque modèle dans **`high_level/models.py`**
- Implémentez des **`DetailView`** pour vos classes qui appellement ces **`.json()`** dans **`high_level/views.py`**. Pour cela, utilisez **`django.http.JsonResponse`** dans la méthode **`render_to_response`**
- Ajoutez des routes pour ces vues dans **`crayon/urls.py`**
- Testez les avec **`curl`**: toutes les informations sur un modèle en particulier doivent apparaître

ref. https://ccbv.co.uk/DetailView

JSON étendu

- Implémentez une méthode “**`def json_extended(self):`**” pour sérialiser chaque modèle et ses relations
- Ajoutez une vue **`ApiView`** dans **`high_level/views.py`**
- Ajoutez une route **`/api/<int:pk>`** dans **`crayon/urls.py`**
- Testez la avec **`curl`**: toutes les informations nécessaires au code C++ doivent apparaître

API avancée (facultatif)

Ajoutez des vues, urls et tests pour les coûts et l'approvisionnement en JSON.