Python – Django API REST

Objectifs du cours

- Les Bases de Django
  - Serveur de page, base de données, langage
  - L’architecture du Framework Django
  - Le modèle MVC et MVT
  - Lancement du serveur
  - Transfert d’une page
- **Django API REST**
  - Présentation de la logique des API REST
  - Mise en place de l’architecture
  - Cas concret

Présentation de l’API REST

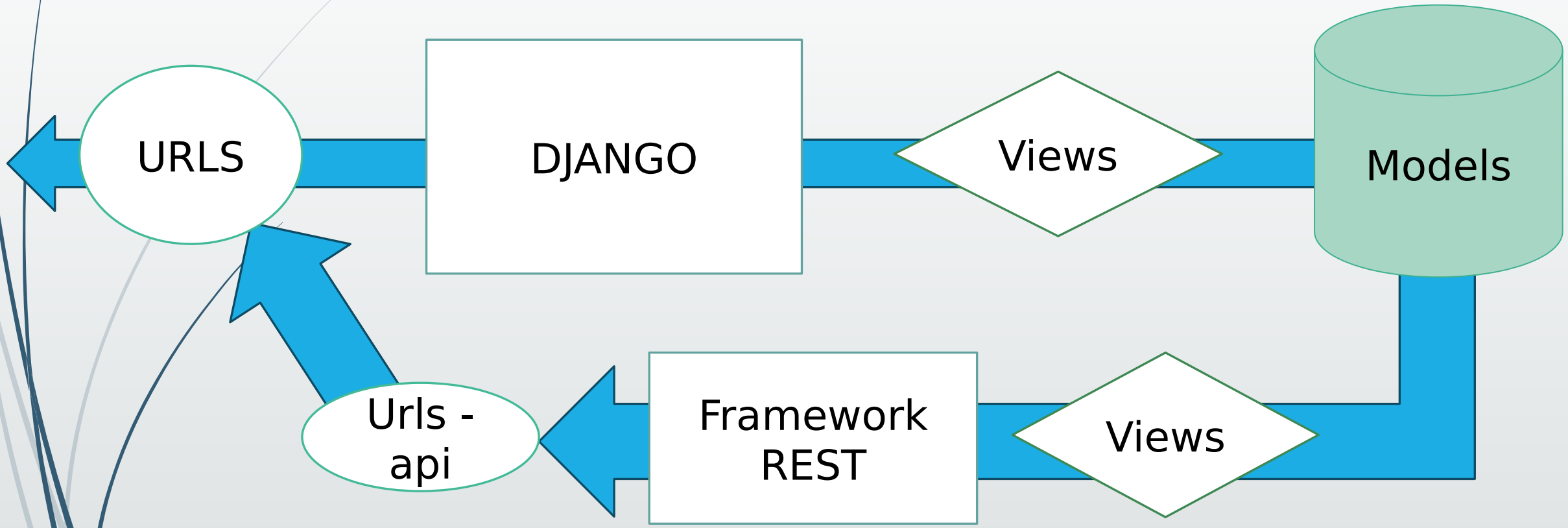
- Application Programming Interface (Interface de Programmation d’Application)
  - Permet à des applications de communiquer entre elles
  - Dans le cas d’une application mobile on parle de BackEnd et de FrontEnd
  - Le format d’échange est le JSON
- RESTful

  Ensemble de principe permettant de définir un échange comme REST
  - Utilise le protocole et les actions HTTP
  - Pas de maintien de connexion (session), on s’identifie à chaque échange
  - Contient le descriptif de l’échange

Les actions HTTP

- Echanges sous la forme d'un CRUD d'échange de données
  - **GET** : permet de récupérer des données de la ressource
  - **POST** : permet d'envoyer des données pour créer une ressource
  - **PUT** : remplace les données de la ressource
  - **DELETE** : supprime la ressource

Architecture d’une application API



Installation

- Récupérer le framework rest (permet d'ajouter la couche technique de l'API)

  pip install djangorestframework

- Créer une nouvelle application pour la partie api

  python manage.py startapp posts

- On ajoute dans firstapp/firstapp/settings.py les applications INSTALLED_APP
  - 'rest_framework', permet de créer l'api
  - 'rest_framework.authtoken', permet de sécuriser l'api
  - 'posts' défini l'application post dans django

Modification du models.py de l'application

- On saisit dans le fichier firstapp/posts/models.py la définition suivante :

```
from django.db import models
from django.contrib.auth import get_user_model

User = get_user_model()

class Post(models.Model):
    title = models.CharField(max_length=255)
    body = models.TextField()
    created_on = models.DateTimeField(auto_now_add=True)
    user = models.ForeignKey(User, on_delete=models.CASCADE)

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

- Get_user_model() permet de récupérer la référence du modèle User natif
- ForeignKey permet de définir un lien entre les deux tables (user et post)
- La fonction __str__ permet de renvoyer le titre au niveau de la page d'admin

Administration d’un modèle

- Modifier le fichier firstapp/posts/admin.py

  from .models import Post

  admin.site.register(Post)

- On lance ensuite la « migration » prise en compte de la classe

  python manage.py makemigrations

  python manage.py migrate

- On lance le server et on se connecte à la console d’administration (/admin)

  Python manage.py runserver

  http://localhost:8000/admin

Il y a l’application et la gestion des jetons d’authentifications

API REST : création d'un Serializer

- Ce qui permet de définir les modes d'appel de l'API

- Créer un fichier firstapp/posts/serializers.py

```
# posts/serializers.py
from rest_framework import serializers
from .models import Post


class PostSerializer(serializers.ModelSerializer):
    class Meta:
        model = Post
        fields = '__all__'
```

API REST : Création de la vue posts/views.py

```
from rest_framework import viewsets
from .models import Post
from .serializers import PostSerializer

class PostViewSet(viewsets.ModelViewSet):
    serializer_class = PostSerializer
    queryset = Post.objects.all()
```

API REST : gestion des routes /posts/urls.py

```
from django.urls import path, include
from rest_framework.routers import DefaultRouter
from .views import PostViewSet

# on utilise le routage par défaut du rest_framework
router = DefaultRouter()
# on définit le chemin de la vue avec le framework rest
router.register('posts', PostViewSet, 'posts')

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

Modification de la route du projet firstapp/urls.py

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

# on ajoute les routes de posts définie dans post/urls.py
urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('posts.urls')),
]
```

La liste des commandes API

- Lire la liste de tous nos articles : GET api/posts
- Lire un article en particulier: GET api/posts/1
- Créer un article : POST api/posts
- Modifier un article : PUT api/posts/1
- Effacer un article : DELETE api/posts/1

- Il est possible d'accéder à l'API par le navigateur
  - http://127.0.0.1:8000/api/posts

Appel de l’API depuis python

- On crée un fichier python

```
import requests
url = 'http://127.0.0.1:8000/api/posts/1/'
r = requests.get(url)
print (r.json())
```

Securisation de l'API

- On ajoute l'authentification à la vue /posts/views.py

```
from rest_framework.permissions import IsAuthenticated
```

- Dans la classe on ajoute

```
permission_classes = (IsAuthenticated,)
```

- Si on tente de se connecter, on a un message d'erreur d'authentification
  - http://127.0.0.1:8000/api/posts/?format=api

Mise en place de la gestion des token

- Dans le fichier firstapp/settings.py on ajoute les paramètres du framework rest

```
REST_FRAMEWORK = { 'DEFAULT_AUTHENTICATION_CLASSES':
[ 'rest_framework.authentication.TokenAuthentication',
], }
```

- Et on effectue la prise en compte de cette modif

```
Python manage.py migrate
```

Création des token d’identification

- Dans l’interface d’administration on crée des jetons d’authentification pour un utilisateur
- « ab41191e579e2f1e4e5f56553ada1d195e83ad77 »

Création d'un fichier python pour accéder à l'api

```
import requests

url = 'http://127.0.0.1:8000/api/posts/'

# pas bon

#headers = {'Authorization': 'Token
9054f7aa9305e012b3c2300408c3dfdf390fcddf'}

headers = {'Authorization': 'Token
ab41191e579e2f1e4e5f56553ada1d195e83ad77'}

r = requests.get(url, headers=headers)

print (r.json())
```