

Angular - Django

gael.durand@novolinko.com

Django et Angular ?

Front-end

Développement partie visible
du projet
(client web - Angular)

Back-end

Développement partie cachée
du projet
(serveur web - Django)



Technologies



Angular et Django sont 2 framework qui ont pour avantage d'être puissants, d'avoir un code bien architecturé et d'être simple d'utilisation.



Angular



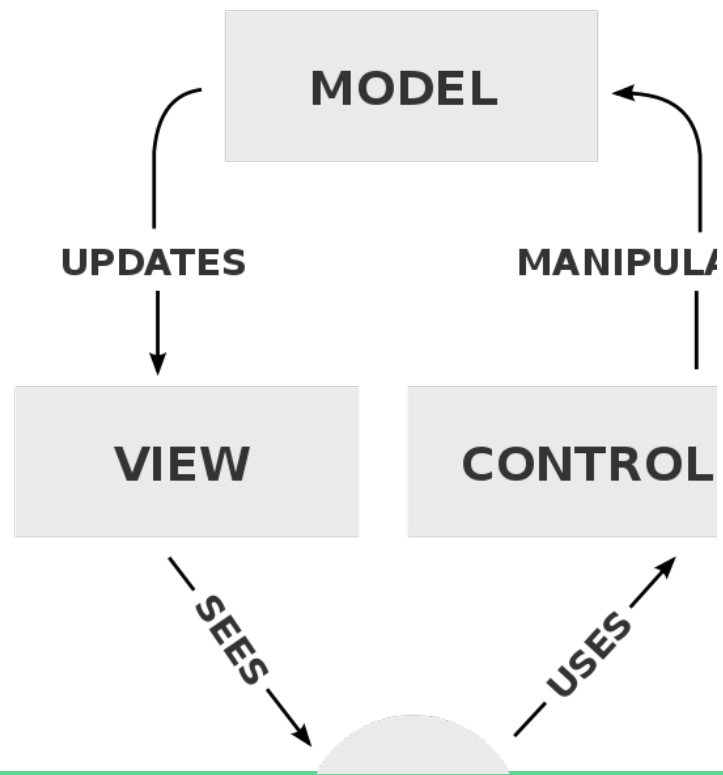
Angular

Angular est un framework JS.

Basé sur le langage Type Script
qui est transcrit à la compilation
en JS.



Angular - MVC



Angular - Set up

Dans un premier temps veuillez installer node Js, et npm (package manager)

Puis suivez ces étapes :

```
npm install -g @angular/cli
```

Installation de angular cli

```
ng new my-app
```

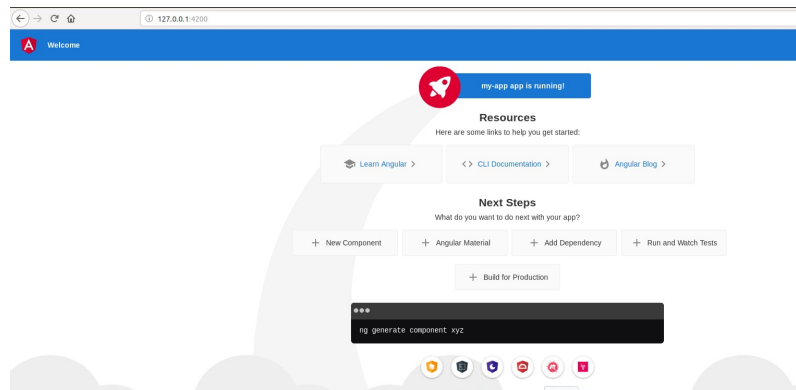
Création d'une nouvelle application angular. Son nom est my-app

Angular - Set up (2)

```
cd my-app  
ng serve --open
```

my-app = est le dossier dans lequel se trouve tout le projet.
ng serve est la commande pour demarrer le server node js, qui lance le projet angular.

Ouvrir un navigateur web et copier cette adresse : `http://127.0.0.1:4200`



Angular - Set up (3)

ng build et ng serve

(méthode pour la prod) ng build va builder l'application en prenant tous les fichiers, les transpiler en javascript et stocker la sortie dans le dossier "./build".

On pourra ensuite mettre le contenu de ce dossier "./build" sur un serveur apache ou nginx.

(méthode pour le developement) ng serve va faire la même chose que "ng build, c'est-à-dire qu'il va compiler les fichiers, mais sans créer le dossier "./build".

A la place, "ng serve" déploie un serveur node sur notre machine rendant le site accessible via notre navigateur.

A chaque update d'un fichier, le serveur va rebuild automatiquement et rafraîchir la page.

Angular - Architecture d'application

Avec une application Angular, nous avons un composant principal qui est décomposé en sous-composants.

Ce qui rend :

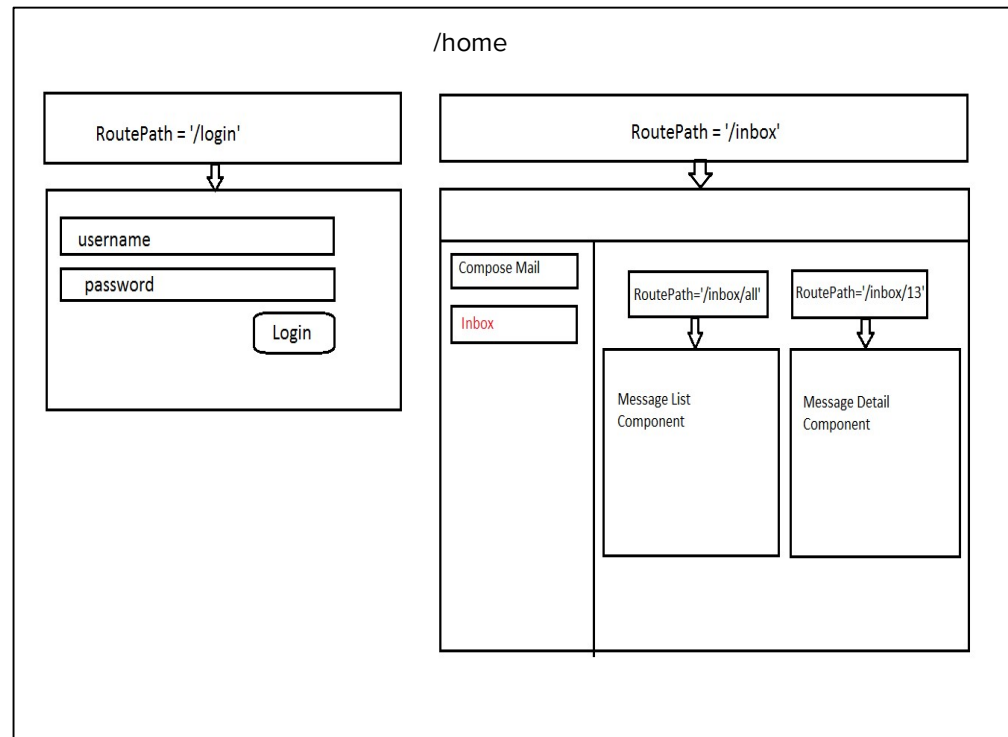
- L'application modulaire
- Le code plus lisible et maintenable
- Les composants réutilisables dans d'autres applications.

Chaque composant a ses fichiers html, css et ts.

Pour créer un nouveau composant, comme un composant login, nous exécutons cette commande :
`ng new component [nameComponent]`



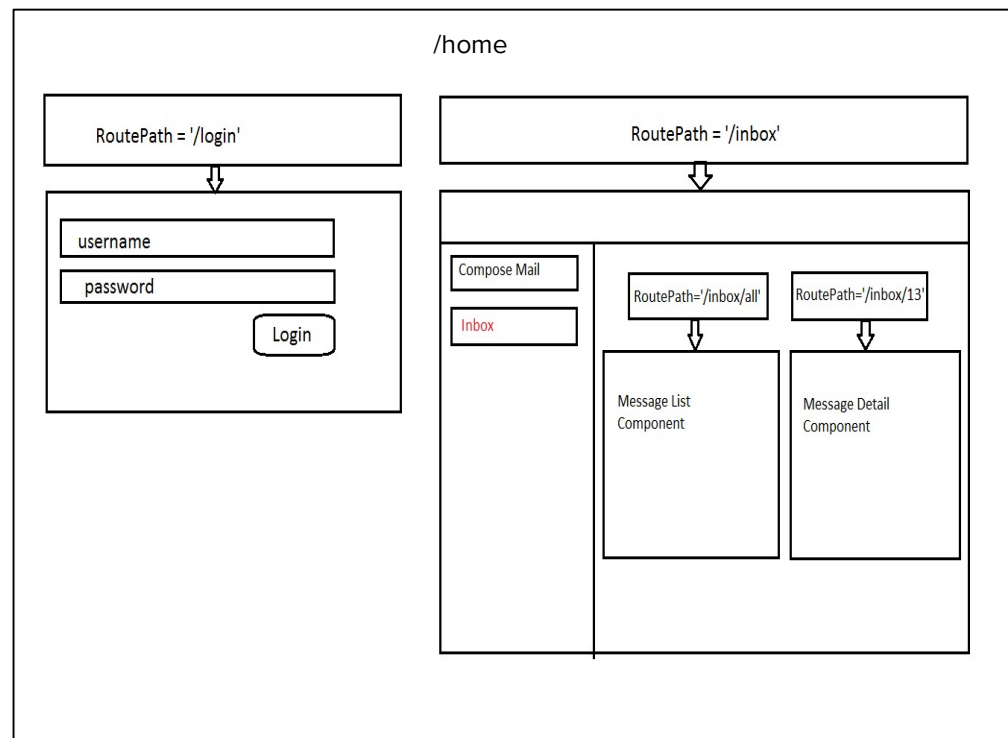
Angular - Architecture



On retrouve ici :

- un composant home
 - `home.component.html`
 - `home.component.css`
 - `home.component.ts`
- Un composant login
 - `login.component.html`
 - `login.component.css`
 - `login.component.ts`
- un composant inbox
 - `inbox.component.html`
 - `inbox.component.css`
 - `inbox.component.ts`

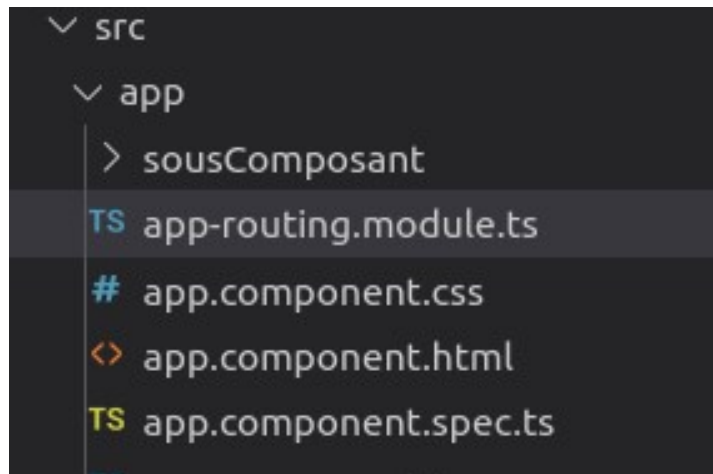
Angular - Architecture



Dans cette exemple, dans le composant home, en fonction de la route, nous affichons

- soit le sous-composant “login”
- soit le sous-composant “inbox”

Angular - Les routes (urls)



Ouvrir le fichier `app-routing.module.ts`, pour configurer les routes de l'application

Angular - Les routes (urls)

```
src > app > TS app-routing.module.ts > ...
1  import { NgModule } from '@angular/core';
2  import { Routes, RouterModule } from '@angular/router';
3
4  import { LoginpageComponent } from './loginpage/loginpage.
5  import { ListPageComponent } from './list-page/list-page.c
6
7  const routes: Routes = [
8    { path: 'login', component: LoginpageComponent },
9    { path: '', redirectTo: '/login', pathMatch: 'full' },
10   { path: 'list', component: ListPageComponent }
11 ];
12
13 @NgModule({
14   imports: [RouterModule.forRoot(routes)],
15   exports: [RouterModule]
16 })
```

Dans ce fichier, veuillez importer les composants.

Définir la route que vous souhaitez pour accéder à ce composant.

Et définir le tableau routingComponents, pour définir les composants des routes.

Angular - Les routes (urls)

Dans un composant vous souhaitez accéder à une nouvelle page ?
Vous avez deux solutions !

Dans le contrôleur avec navigate

```
src > app > loginpage > TS loginpage.component.ts > ...
1  import { Component, OnInit } from '@angular/core';
2  import { Router } from '@angular/router';
3
4  @Component({
5    selector: 'app-loginpage',
6    templateUrl: './loginpage.component.html',
7    styleUrls: ['./loginpage.component.scss']
8  })
9  export class LoginpageComponent implements OnInit {
10
11    constructor(private router: Router) { }
12
13    ngOnInit() {
14    }
15
16    onSubmit() {
17      if (user.connected == true)
```

OU

Dans le code html avec routerLink

```
30  <!-- Button -->
31  <button type="button"
32    routerLink="/list"
33    routerLinkActive="active"
34    >Aller à list page
```

<https://angular.io/guide/router>

Angular - Les formulaires

Inscription

Vous pouvez vous inscrire via ce formulaire.

Adresse email *

Mot de passe *

Confirmation du mot de passe *

Nom d'utilisateur

Inscription

Ouvrir le composant dans lequel vous souhaitez afficher votre formulaire.

Dans le fichier
“moncomposant.component.html”, veuillez
mettre le code html du formulaire.

Dans le fichier
“moncomposant.component.ts” veuillez
mettre le traitement du formulaire.

Angular - Les formulaires (2)

Utilisation du composant FormBuilder d'Angular.

input == FormControl

```
6   <div class="center">
7     <div class="center_bloc" style="width: 500px;">
8       <!-- Default form login -->
9       <form class="text-center border border-light p-5" [formGroup]="formNewConfig" (ngS
10
11         <p class="h4 mb-4">Sign in</p>
12
13         <!-- Email -->
14         <input type="email" FormControlName="loginFormEmail"
15           id="loginFormEmail" class="form-control mb-4" placeholder="E-mail">
16
17         <!-- Password -->
18         <input type="password" FormControlName="loginFormPassword"
19           id="loginFormPassword" class="form-control mb-4" placeholder="Password">
20
21         <!-- Sign in button -->
22         <button mdbBtn type="submit" color="info" block="true" class="my-4">Sign in</but
```

Angular - Les formulaires (3)

```
src > app > loginpage > TS loginpage.component.ts > LoginpageCompon
1  import { Component, OnInit } from '@angular/core'
2  import { FormGroup, FormControl } from '@angular
3  import { Router } from '@angular/router';
4
5  @Component({
6    selector: 'app-loginpage',
7    templateUrl: './loginpage.component.html',
8    styleUrls: ['./loginpage.component.scss']
9  })
10 export class LoginpageComponent implements OnInit
11
12   formNewConfig = new FormGroup ({
13     loginFormEmail: new FormControl(''),
14     loginFormPassword: new FormControl('')
15   });
16
17   constructor(private router: Router) { }
```

Angular - Les services

Un service ? Un service en angular est une classe qui permet de faire des traitements hors du composant.

On peut utiliser un service et avoir une méthode qui permet de calculer des statistiques par exemple.

Ou encore avoir un service comme `listPage.service.ts`, qui permet d'interroger une base de données via une requête HTTP et de recevoir une réponse au format JSON.



Angular - Les services (2)

Comment faire un service ?

Créer un fichier dans le composant principal, nommé `monservice.service.ts` (list.service.ts)

```
1  import { Injectable } from '@angular/
2
3  @Injectable()
4  export class ListService {
5
6      constructor() { }
7
8      getList() {
9          console.log("test");
```

Angular - Les services (3)

Comment intégrer et utiliser mon service dans mon composant ?

Tout d'abord veuillez importer le service, puis le déclarer dans le constructeur. Enfin celui-ci est prêt à être appelé.

```
1 | import {Component} from '@angular/core';
2 | import {ListService} from "../list.service"
3 |
4 | @Component({
5 |   selector: 'app-list-page',
6 |   templateUrl: './list-page.component.html',
7 |   styleUrls: ['./list-page.component.scss']
8 | })
9 | export class ListPageComponent {
10 |
11 |   public elements: any = [];
12 |
13 |   constructor(private _listService: ListServ
14 |
15 |   ngOnInit() {
```

Angular - Les requêtes, get, post, ...

Requête GET = recevoir des informations d'une base de données

Requête POST = ajouter un item à une base de données

Requête PUT = actualiser les données

En Angular, nous gérons ces requêtes dans les services.



Angular - Les requêtes, get, post, ... (2)

```
1 import { Injectable } from '@angular/core';
2 import { Http, RequestOptions } from '@angular/http';
3 import { Observable } from 'rxjs';
4
5 export interface Ticket {
6     id: Number,
7     libelle: String,
8     description: String,
9     status: string,
10    email: String,
11    creationDate: String,
12    priority: String,
13    category: String,
14    isUpdating: boolean
15 }
16
17 const API_URL: string = 'http://localhost:8000/ticket'
18
19 @Injectable({
20     providedIn: 'root'
```

Importations et déclarations

Nous allons créer une interface “Ticket”.

Nous allons aussi importer le module Http, et RequestOptions d’Angular.

On va déclarer le module Http dans le constructeur.

Angular - Les requêtes, get, post, ... (3)

```
22 export class TicketService {
23     private headers;
24     constructor(private http: Http) {
25     }
26
27     getTickets(): Observable<Ticket[]> {
28         return this.http.get(API_URL + '/api/v1/tickets/',
29             new RequestOptions({ headers: this.headers })
30         )
31         .map(res => {
32             let modifiedResult = res.json();
33             return modifiedResult;
34         });
35     }
36
37     addTicket(ticket): Observable<Ticket> {
38         return this.http.post(API_URL + '/api/v1/tickets/', ti
39             new RequestOptions({ headers: this.headers })
40         ).map(res => res.json());
41     }
42 }
```

Faire une requête Http dans le service

Pour chaque requête nous retournons une réponse au format json.

Angular - Les requêtes, get, post, ... (4)

```
30     getTickets() {  
31         this.ticketService  
32             .getTickets()  
33             .subscribe(  
34                 tickets => {  
35                     this.tickets = tickets;  
36                     console.log("tickets => ", tickets);  
37                     this.isLoading = false;  
38                 },  
39                 error => this.errorMessage = <any>
```

Récupérer la réponse de la requête dans le composant

Nous utilisons la même méthode pour appeler le service.

Puis dans la fonction “getTickets()” du service “ticketService” nous pouvons grâce à subscribe de récupérer la réponse de la requête GET.

Angular - Internationalisation

Le module internationalisation avec i18n est inclus dans Angular.

Pour l'utiliser, nous ajoutons le module :

```
npm install @ngx-translate/core @ngx-translate/http-loader --save
```



Angular - Internationalisation (1)

```
src > app > TS app.module.ts > ...
1  import { BrowserModule } from '@angular/platform-browser';
2  import { NgModule } from '@angular/core';
3
4  |
5  import { TranslateModule, TranslateLoader } from '@ngx-translate/core';
6  import { TranslateHttpLoader } from '@ngx-translate/http-loader';
7  import { HttpClientModule, HttpClient } from '@angular/common/http';
8
9  import { AppRoutingModule } from './app-routing.module';
10 import { AppComponent } from './app.component';
11
12 export function HttpLoaderFactory(http: HttpClient) {
13   return new TranslateHttpLoader(http);
14 }
15
16 @NgModule({
17   declarations: [
18     AppComponent
19   ],
20   imports: [
21     BrowserModule,
22     AppRoutingModule,
23     HttpClientModule,
24     TranslateModule.forRoot({
25       loader: {
26         provide: TranslateLoader,
27         useFactory: HttpLoaderFactory,
28         deps: [HttpClient]
29       }
30     })
31   ],
32   providers: [],
33   bootstrap: [AppComponent]
34 })
35 export class AppModule {}
```

Dans le fichier “app.module.ts”, nous allons importer les modules nécessaires et les déclarer dans “import”.

Angular - Internationalisation (2)

```
src > app > < app.component.html > ...
1  <div>
2    <h2>{{ 'HOME.TITLE' | translate }}</h2>
3    <p>
4      {{ 'HOME.TEXT' | translate }}
5    </p>
6
7
8    <label for="">
9      {{ 'HOME.SELECT' | translate}}
10     <select #langselect (change)="translate.use(langselect.value)">
11       <option *ngFor="let lang of translate.getLangs()" [value]="lang">{
```

“HOME.TITLE”
est une
variable
définie dans
les fichiers
fr.json et
en.json

Angular - Internationalisation (3)

```
src > app > TS app.component.ts > AppComponent
1  import { Component } from '@angular/core';
2  import { TranslateService } from '@ngx-translate/core';
3
4  @Component({
5    selector: 'app-root',
6    templateUrl: './app.component.html',
7    styleUrls: ['./app.component.css']
8  })
9  export class AppComponent {
10    title = 'my-app';
11
12    constructor(public translate: TranslateService) {
13      translate.addLangs(['fr', 'en']);
14      translate.setDefaultLang('en');
15    }
16  }
```

Dans le contrôleur, nous définissons un tableau de sélection de langue (addLangs(['fr', 'en'])) et nous mettons en place la récupération de la langue du navigateur.

Angular - Internationalisation (4)

```
src > assets > i18n > {} fr.json > ...  
1  {  
2    "HOME": {  
3      "TITLE": "Ceci est le titre"  
4      "TEXT": "Ceci est le text",  
5      "SELECT": "Selectionner une
```

Nous retrouvons ici les variables dans 2 fichiers.

Un pour l'anglais (en.json) et l'autre pour le français (fr.json).

```
src > assets > i18n > {} en.json > ...  
1  {  
2    "HOME": {  
3      "TITLE": "This is the  
4      "TEXT": "This is the  
5      "SELECT": "Select lan
```

Ils sont placés dans le dossier "assets/i18n"



Angular - Interaction avec Django

L'intégration avec Django se fait simplement étant donné en mettant en place une api Django.

Par exemple, pour récupérer une liste d'items d'une base de données, nous créons la requête dans le service Angular avec l'adresse de l'API Django. cette API lance le traitement côté backend et retourne la réponse au frontend Angular en JSON.

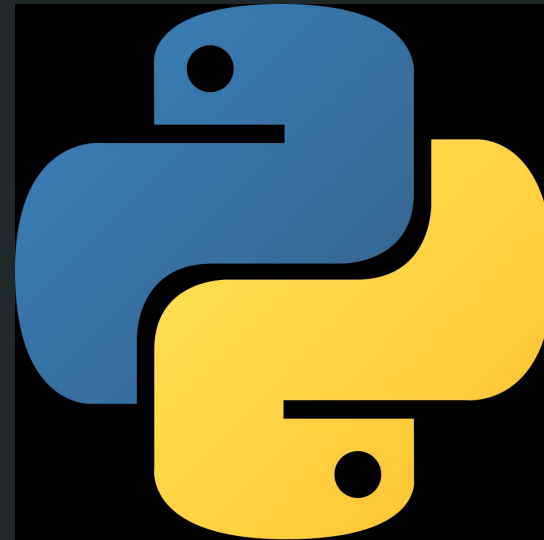


Django

django

Django

Django est un framework python.



<https://docs.djangoproject.com/en/2.2/>

Django - Set up

Installation de Django et de ses dépendances.

Pour ceci, veuillez suivre ces étapes :

- `pip install pipenv`
- `pipenv install Django`
- `pip install djangorestframework` #permet de pouvoir utiliser notre programme en tant qu'Api rest

Pip est un gestionnaire de modules Python.

pipenv permet de combiner un environnement virtuel (venv) et pip.

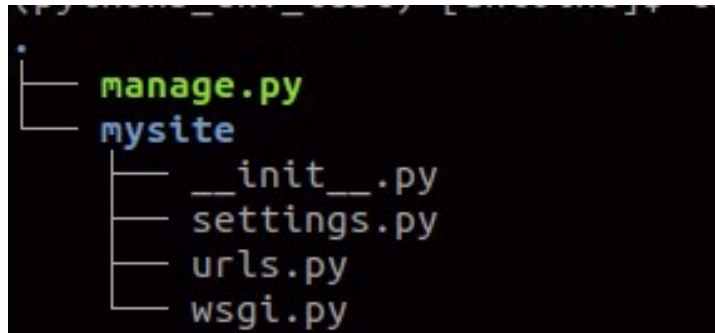


Django - Set up (1)

Une fois dans notre environnement nous avons un fichier PipFile.
Dans ce fichier sont stockés toutes les dépendances. Pour l'instant nous avons seulement Django et djangoRestFramework.

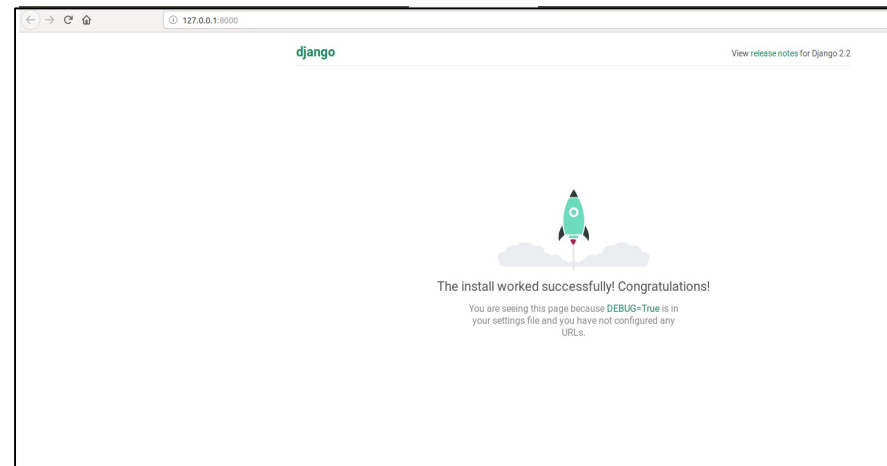
Nous initialisons un projet Django :

- django-admin **startproject** mysite
- cd mysite



Django - Set up (2)

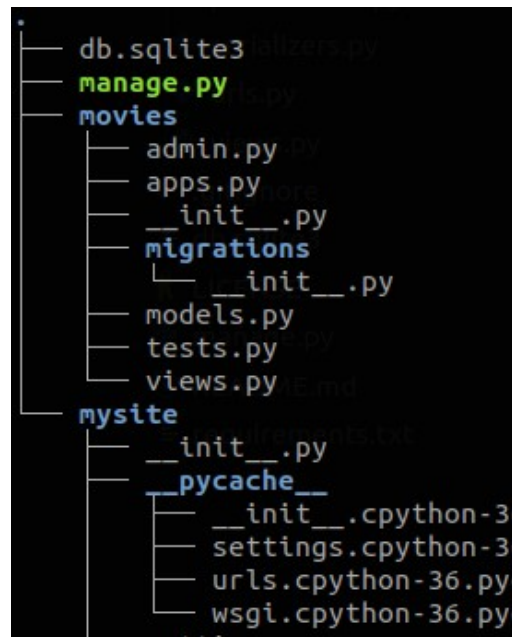
- `python manage.py makemigrations` # Vérifier si le modèle a changé
- `python manage.py migrate` # Les migrations permettent de lier notre modèle à la base de données.
- `python manage.py runserver` # permet de démarrer le serveur python qui fait tourner l'application Django



Django - L'architecture d'une application

Une fois le projet initialisé, nous allons créer une application au sein du projet Django.

- `python manage.py startapp movie`



Django - Settings.py ?

Le fichier settings.py dans l'application principale, permet de configurer le projet.
Il faut ajouter une référence après création de nouvelle application dans INSTALLED_APPS

```
37  # Application definition
38
39  INSTALLED_APPS = [
40      'django.contrib.admin',
41      'django.contrib.auth',
42      'django.contrib.content',
43      'django.contrib.session',
44      'django.contrib.messages',
45      'django.contrib.staticfiles',
46      'rest_framework'
```

Django - Settings.py ? (2)

Nous pouvons aussi configurer les problèmes de cross-origin.

Avec votre application Angular, lorsque vous allez devoir faire une requête HTTP, il est très probable que la requête n'aboutisse pas. Pourquoi ? C'est une question de sécurité lorsque l'application Angular n'a pas le même nom de domaine que l'API Django.


Pour cela nous autorisons les requêtes cross-origin dans les settings, comme ceci :

- Ajouter "corsheaders" dans "INSTALLED_APPS=[]"
- Ajouter "CORS_ORIGIN_WHITELIST = ["<http://127.0.0.1:4200>"] # cette adresse correspond à l'adresse de la plateforme Angular.
- Ajouter 'corsheaders.middleware.CorsMiddleware' dans "MIDDLEWARE = [...]"
- Enfin ajouter la dépendance avec 'pip install corsheaders' dans votre terminal



Django - Modèle (et base de données)

Nous ajoutons dans l'application "movies" les champs de la table movies.


```
movies >  models.py
1  from django.db import models
2
3  # Create Movie Model
4  class Movie(models.Model):
5      title = models.CharField(max_le
6      genre = models.CharField(max_le
```

Puis nous faisons une nouvelle migration pour actualiser le modèle et mettre à jour la BD SQLite.

- python manage.py **makemigrations**
- python manage.py **migrate**


Django - Endpoint de l'api

Nous créons dans l'application "movies" un serializer "serializers.py" pour retourner une réponse au format JSON à partir du modèle, lors d'un appel get ou autre.

```
movies >  serializers.py
1  from rest_framework import serializers
2  from .models import Movie
3
4  class MovieSerializer(serializers.ModelSerializer): # create class to
5
6      class Meta:
7          model = Movie
```

Django - Endpoint de l'api

Nous importons le modèle et le fichier Serializer dans le fichier “views.py”, ainsi que les modules nécessaires pour créer une API.

```
movies >  views.py
1  from rest_framework import status
2  from rest_framework.response import Response
3  from rest_framework.generics import RetrieveUpdateDestroyAPIView,
4  from .models import Movie
```

Django - Endpoint de l'api

```
9 class get_delete_update_movie(RetrieveUpdateDestroyAPIView):
10     serializer_class = MovieSerializer
11
12     def get_queryset(self, pk):
13         try:
14             movie = Movie.objects.get(pk=pk)
15         except Movie.DoesNotExist:
16             content = {
17                 'status': 'Not Found'
18             }
19             return Response(content, status=status.HTTP_404_NOT_F
20         return movie
21
22     # Get a movie
23     def get(self, request, pk):
24
25         movie = self.get_queryset(pk)
26         serializer = MovieSerializer(movie)
27         return Response(serializer.data, status=status.HTTP_200_0
28
29     # Update a movie
30     def put(self, request, pk):
31
32         movie = self.get_queryset(pk)
```

La classe `get_delete_update_movie` hérite de la classe `"RetrieveUpdateDestroyAPIView"`.

Cela permet de définir les méthodes de l'API (get, put, delete...)

Django - Les routes (urls)

“l'url dispatcher” est appelé pour associer une URL à une vue. Les routes du projet Django sont définies dans le fichier “urls.py” de chaque application du projet.

```
1
2 from django.contrib import admin
3 from django.conf.urls import include
4
5 # urls
6 urlpatterns = [
7     url(r'^$', include('movies.urls'))
```

Ajouter votre configuration des routes dans le projet principal.

Pour accéder à l'application : <http://127.0.0.1:8000/>

Avec `url(r'^movies', include...)` :
<http://127.0.0.1:8000/movies>

```
movies > urls.py
1 from django.urls import include, path, re_path
2 from . import views
3
4
5 urlpatterns = [
6     re_path(r'^api/v1/movies/(?P<pk>[0-9]+)$', # Url to get update o
7         views.get_delete_update_movie.as_view(),
8         name='get delete update movie')
```

Dans l'application movies, on crée le fichier urls.py de cette application.

On ajoute nos routes pour accéder à l'API
<http://127.0.0.1:8000/api/v1/movies>

Django - Interface administration

L'interface d'administration de DRF est une interface Django qui permet d'administrer ses APIs.

Ces pages sont simples et ne sont pas conçues pour être utilisées en production.



Django - Interface administration

```
GET /api/v1/movies/

HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
  "count": 4,
  "next": null,
  "previous": null,
  "results": [
    {
      "title": "Avengers: Infinity War",
      "genre": "Superheros",
      "year": 2018,
      "creator": 2
    },
    {
      "title": "Ocean 8",
      "genre": "Comedy",
      "year": 2018,
      "creator": 2
    },
    {
      "title": "Incredibles 2",
      "genre": "Action",
      "year": 2018,
      "creator": 1
    }
  ]
}
```

Par exemple, l'API de movies est accessible à la page :

<http://127.0.0.1:8000/api/v1/movies/>

Django - Suite de tests

Comment créer une suite de test avec Django ?

Dans le fichier “test.py” de l'application django, nous créons plusieurs méthodes de test.

```
movies > test.py
1  from django.test import TestCase
2  from .models import Movie
3  from django.urls import reverse, resolve
4  from .views import get_delete_update_movie, get_post_movies
5
6  class TestUrls(TestCase):
7
8      def testUrlGetDeteUpdate(self):
9          url = reverse("get_delete_update_movie", args=["1"])
10         print (resolve(url))
11         self.assertEqual(resolve(url).func.view_class, get_del
12
13     def testUrlGetPost(self):
```

Ici nous avons trois méthodes qui test si la route correspond bien au bon composant. Ceci, n'est qu'un exemple de test.

Pour lancer les tests, aller à la racine du projet et exécuter :
python manage.py test