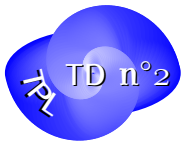

Introduction aux templates Django

Django templates

Exercice 1. Templates Django

Pour nous aider à afficher les données de façon plus structurée, Django fournit un système de templating intégré au framework.
Les développeurs peuvent s'il le souhaitent utiliser d'autres systèmes de templates, comme Jinja2 un peu plus répandu, mais le système de templates de Django est assez

Pour uniformiser nos pages nous allons créer un page de base dont on fera hériter les différentes pages de notre projet. Voici un exemple nommé modeleBase.html, que nous pouvons déposer dans le dossier `templates` de notre app, tandis que nos ressources statiques (css, js, images, etc.) sont à placer dans le dossier `static` de notre app. On utilise alors dans notre template la commande `load static` et on préfixe les adresses des ressources statiques du mot-clé `static` :

```
<!DOCTYPE html>
{% load static %}
<html lang="fr" >
  <head>
  <meta charset="utf-8" >
  <title>
  {% block title %}{% endblock %}
  </title>
  <link rel="stylesheet" href="{% static css/mon.css %}" >
  </head>
  <body>
      <header>
         <h2> Gestion des Taches </h2>
      </header>

    <section>
      {% block content %}
      {% endblock %}
    </section>
    <footer>
      <em>Copyright IUT O, 2019</em>
    </footer>
  </body>
```

```
</html>
```

Django utilise un système de templates volontairement doté d'une syntaxe simplifiée (par rapport à d'autres systèmes de templates commes Jinja en Python ou Twig en PHP). La page est structurée de façon classique et nous avons utilisé la syntaxe des templates django pour créer deux blocs vides (titre et content) Ces deux blocs seront mis en place par les templates héritant de modeleBase.html comme par exemple list.html

Rappelons que dans le fichier views.py nous avons défini la fonction :

```
def task_listing2(request):
    objets=Task.objects.all().order_by('-due_date')
    #print(objets)
    # - (inverse l'ordre )
    return render(request,'lesTaches/list.html',{'taches':
       objets})
```

Nous voyons que l'on passe à la fonction render() un dictionnaire et un template (list.html) Ce sont les données de ce dictionnaire que l'on va pouvoir utiliser dans ce template comme vous pouvez le voir dans l'exemple ci-dessous

```
{% extends 'lesTaches/modeleBase.html' %}
  {% block title %} Liste {% endblock %}
  {% block content %}
    <h3>Tâches en cours ou terminées</h3>
  <ul>
  {% for  tache in taches %}
  <li>
    <h3>
      {% if tache.closed %}
        <span class='closed'>{{tache.name}}</span>
      {% else %}
        {{tache.name}}
      {% endif %}
    </h3>
  <h4>{{tache.description }} </h4>
  <p>
    Date de rendu: {{tache.colored_due_date}}
  </p>
  <p>
    Date de debut: {{tache.schedule_date}}
  </p>
  </li>
```

```
23    {% endfor %}
24  </ul>
25  {% endblock %}
26  </html>
```

Vous pouvez constater :

— que l'on a fait hériter notre template de modeleBase.html

— que nous avons donné un contenu au bloc titre

— que nous avons complété le bloc content.

— Les objets définis dans la fonction task_listing2 se retouvent ici dans le mots clé taches qui était justement l'index des objets dans le dictionnaire. On parcourt l'ensemble des taches par une boucle for et pour chaque tache on effectue le traitement souhaité.

Exercice 2. Template générique

Les templates sont cherchés par défaut dans les dossiers templates de chacune des app qui composent notre projet mais aussi dans le dossier templates situé à la racine de notre projet. Les ressources statiques sont cherchées par défaut dans les dossiers `static` de nos app. On peut vouloir ajouter des dossiers de ressources statiques en dehors des apps. Pour celà, il suffit d'ajouter dans les `settings.py` de notre projet :

```
1  STATICFILES_DIRS = [
2    os.path.join(BASE_DIR), "static"),
3  ]
```

Ce qui ajoutera le dossier `static` placé à la racine du projet. Installons dedans quelques librairies avec la commande `yarn` :

```
     yarn add bootastrap jquery-slim popper.js
```

Ce qui installera dans le dossier `node_modules` les librairies demandées. Mettons maintenant en place un template assez générique, `base.html` dans le dossier `templates` du projet :

```
1  <!doctype html>
2
3  {% load static %}
4
5  <html lang="fr">
6  <head>
7      <meta charset="utf-8">
8      <title>
```

```
 9           {% block title %}{% endblock %}
10       </title>
11       {% block metas %}
12       <meta name="viewport" content="width=device-width,
           initial-scale=1, shrink-to-fit=no">
13       {% endblock %}
14       {% block styles %}
15       <link rel="stylesheet" href="{% static 'bootstrap/css/
           bootstrap.min.css' %}">
16       {% endblock %}
17   </head>
18
19   <body>
20       {% block nav %}
21       <nav class="navbar navbar-expand-md navbar-dark bg-dark
           mb-4">
22           <h1> Gestion des Tâches</h1>
23       </nav>
24       {% endblock %}
25       <main role="main" class="container top-pushed">
26           <div class="jumbotron">
27               {% block content %}
28               {% endblock %}
29           </div>
30       </main>
31       {% block js %}
32       <script src="{% static 'node_modules/jquery-slim/dist/
           jquery-slim.min.js' %}"></script>
33       <script src="{% static 'node_modules/bootstrap/dist/css/
           bootstrap.min.js' %}"></script>
34       <script src="{% static 'node_modules/popper.js/dist/
           popper.min.js' %}"></script>
35       {% endblock %}
36   </body>
37
38   </html>
```

Retournons maintenant dans notre app les Taches pour y installer un nouveau template list2.html qui hérite de ce template de base et qui ajoute l'utilisation d'icônes *fontawesome* :

```
1  {% extends "base.html" %}
2
3      {% block title %}
```

```
4           Les Tâches
5       {% endblock %}
6
7       {% block styles %}
8           {{ block.super }}
9           <link rel="stylesheet"
10                href="https://use.fontawesome.com/releases/v5
                      .3.1/css/all.css"
11                integrity="sha384-
                      mzrmE5qonljUremFsqc01SB46JvROS7bZs3IO2EmfFsd15uHvIt
                      +Y8vEf7N7fWAU"
12                crossorigin="anonymous">
13      {% endblock %}
14
15      {% block content %}
16      <h1>Liste des tâches</h1>
17
18      {% for tache in objects %}
19      <div class="row">
20          <div class="col">{{tache.name}}</div>
21          <div class="col">{{tache.created_date}}</div>
22          <div class="col">{{tache.colored_due_date}}</div>
23          <div class="col">
24                  {% if tache.closed %}
25                      <i class="fas fa-lock"></i>
26                  {% else %}
27                      <i class="fas fa-lock-open"></i>
28                  {% endif %}
29          </div>
30      </div>
31      {% endfor %}
32
33      {% endblock %}
```

Pour l'utiliser, modifions notre vue dans views.py en conséquence :

```
1  def task_listing(request):
2      objects = Task.objects.all().order_by('due_date')
3      return render(request, template_name='list2.html',
           context={'objects': objects} )
```

Nous avons déjà une liste mieux présentée. Vous pouvez améliorer encore un peu la présentation en utilisant d'autres styles de bootstrap ...