15. Finaliser votre application

Nous avons déjà franchi toutes les étapes nécessaires à la création de notre site web : nous savons maintenant comment écrire un modèle, une URL, une vue et un template. Nous avons même réussi à rendre notre site web plus joli .

C'est le moment de pratiquer tout ce que vous avez appris !

Tout d'abord, il faudrait que notre blog possède une page qui permet d'afficher un billet, n'est-ce pas ?

Nous avons déjà un modèle `Billet`, nous n'avons donc pas besoin d'ajouter quoi que ce soit à `models.py`.

Créer un lien dans un template

Nous allons tout d'abord ajouter un lien à l'intérieur du fichier `blog/templates/blog/post_list.html`. Ouvrez-le dans l'éditeur de code et voyez qu'il devrait ressembler à ceci :

`blog/templates/blog/post_list.html`

```
{% extends 'blog/base.html' %}

{% block content %}
    {% for billet in billets %}
        <div class="billet">
            <div class="date">
                {{ billet.published_date }}
            </div>
            <h2><a href="">{{ billet.title }}</a></h2>
            <p>{{ billet.text|linebreaksbr }}</p>
        </div>
    {% endfor %}
{% endblock %}
```

Nous aimerions pouvoir cliquer sur le titre du billet et arriver sur une page avec le contenu de celui-ci. Pour cela, changeons <h2> billet.title </h2> pour qu'il pointe vers la page de contenu du billet :

`blog/templates/blog/post_list.html`

```
<h2><a href="{% url 'post_detail' pk=billet.pk %}">{{ billet.title
}}</a></h2>
```

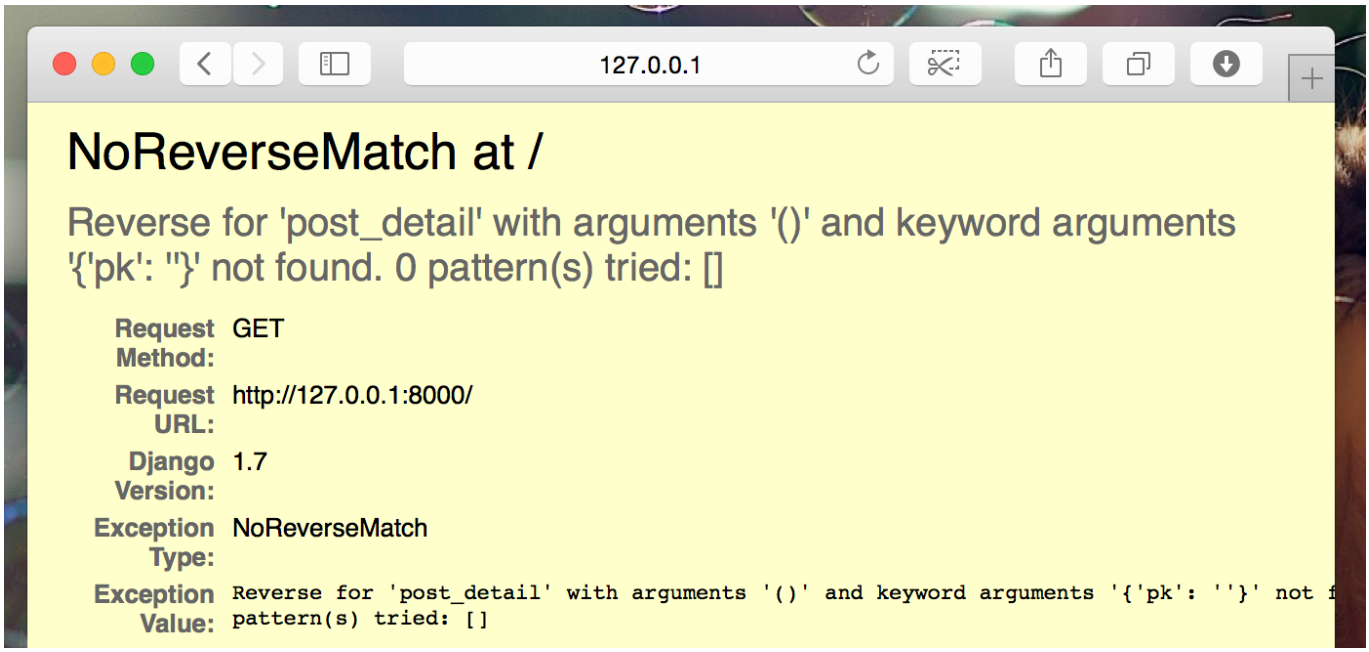
C'est le moment parfait pour expliquer ce mystérieux `{% url 'post_detail' pk=billet.pk %}`. Vous vous souvenez peut-être que la notation `{% %}` nous permet d'utiliser les balises de

template Django. Cette fois-ci, nous allons utiliser des balises qui vont s'occuper de créer des URLs à notre place !

La partie post_detail signifie que Django s'attend de trouver une URL en `blog/urls.py` avec `nom = post_detail`

Et qu'en est-il de `pk=billet.pk` ? `pk` est l'abréviation de clé primaire (« primary key » en anglais), qui est un identifiant unique pour chaque entrée dans une base de données. Chaque modèle Django a un champ qui sert de clé primaire, et peu importe son autre nom, il peut aussi être appelé par le nom « pk ». Comme nous n'avons pas spécifié de clé primaire dans notre modèle Billet, Django en crée une pour nous (par défaut, un champ nommé « id » contenant un nombre qui augmente pour chaque entrée, c'est-à-dire 1, 2, 3, etc.) et l'ajoute comme champ à chacun de nos billets. Nous accédons à la clé primaire en écrivant billet.pk, pareil que pour accéder aux autres champs (title, author, etc.) de notre objet Billet !

Maintenant si nous jetons un coup d'œil à http://127.0.0.1:8000/, nous rencontrons une erreur. Ceci est prévisible, puisque nous n'avons ni d'URL ni de view pour post_detail. L'erreur ressemble à ceci :



Créer une URL vers le contenu d'un billet

Allons créer notre URL dans le fichier `'urls.py` pour notre vue `post_detail` !

Nous aimerions que le contenu de notre premier billet s'affiche à cette URL : http://127.0.0.1:8000/billet/1/

Allons créer une URL dans le fichier `blog/urls.py` qui dirigera Django vers une vue appelée `post_detail`. Cela nous permettra d'afficher l'intégralité d'un billet de blog. Ouvrez le fichier `blog/urls.py` dans l'éditeur de code et ajoutez la ligne `path('billet/<int:pk>/', views.post_detail, name='post_detail')`, afin que le fichier ressemble à ceci :

`blog/urls.py`

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.post_list, name='post_list'),
    path('billet/<int:pk>/', views.post_detail, name='post_detail'),
]
```

La partie `billet/<int:pk>/` spécifie un modèle d'URL – allons voir plus dans le détail :

- billet/ signifie que l'URL doit commencer par le mot billet suivie d'un /. OK, pas très compliqué.
- <int:pk>– cette partie est plus délicate. Cela signifie que Django s'attend à une valeur entière (int), qu'en suite il transférera à une vue comme variable de nom pk.
- / – il nous faut un / à nouveau avant la fin de l'URL.

Concrètement, cela signifie que si vous entrez http://127.0.0.1:8000/billet/5/ dans votre barre d'adresse, Django va comprendre que vous cherchez à atteindre une vue appelée post_detail et qu'il doit communiquer à cette vue que pk est égal à 5.

OK, nous avons ajouté un nouveau modèle d'URL à `blog/urls.py` ! Actualisons la page : http://127.0.0.1:8000/ Boom ! Le serveur a cessé de marcher à nouveau. Jetez un oeil sur la console – comme prévu, il y a encore une autre erreur !

```
    return _bootstrap._gcd_import(name[level:], package, level)
  File "<frozen importlib._bootstrap>", line 2231, in _gcd_import
  File "<frozen importlib._bootstrap>", line 2214, in _find_and_load
  File "<frozen importlib._bootstrap>", line 2203, in _find_and_load_unlocked
  File "<frozen importlib._bootstrap>", line 1200, in _load_unlocked
  File "<frozen importlib._bootstrap>", line 1129, in _exec
  File "<frozen importlib._bootstrap>", line 1448, in exec_module
  File "<frozen importlib._bootstrap>", line 321, in _call_with_frames_removed
  File "/home/hel/code/djangogirls/workthrough/blog/urls.py", line 6, in <module>
    url(r'^post/(?P<pk>[0-9]+)/$', views.post_detail, name='post_detail'),
AttributeError: 'module' object has no attribute 'post_detail'
```

Vous souvenez-vous de ce qu'il faut faire ensuite ? Il faut ajouter une vue !

Ajouter une vue pour le contenu du billet

Cette fois-ci, nous allons donner un paramètre supplémentaire à notre vue : pk. Notre vue va avoir besoin de le récupérer, n'est ce pas ? Pour cela, nous allons définir une fonction : `def post_detail(request, pk):`. Notez que ce paramètre doit avoir exactement le même nom que celui que nous avons spécifié dans le fichier urls (pk). Notez aussi qu'oublier de mettre cette variable est incorrect et produira une erreur !

Maintenant, nous aimerions obtenir qu'un seul billet de blog. Pour cela, nous allons utiliser des QuerySets qui ressemblent à ceux-ci:

`blog/views.py`

```
Billet.objects.get(pk=pk)
```

Cependant, il y a un petit problème dans cette ligne de code. Si aucun de nos Billets ne possèdent cette primary key (clef primaire) (pk), nous allons nous retrouver avec une super erreur bien cracra!

Dans l'idéal, nous aimerions pouvoir éviter ça! Encore une fois, Django nous offre l'outil parfait pour ça : `get_object_or_404`. Dans le cas où il n'existerait pas de Billet avec le pk indiqué, une page d'erreur beaucoup plus sympathique s'affichera : c'est ce qu'on appelle une erreur 404 : page non trouvée.

La bonne nouvelle, c'est que vous pouvez créer vous-mêmes votre page Page non trouvée et en faire ce que vous voulez ! Reconnaissez que ce n'est pas le plus important pour le moment donc nous allons zapper cette partie ;).

Ok, ajoutons notre vue à notre fichier `views.py`!

Dans le fichier `blog/urls.py` nous avons crée un modèle d'URL post_detail qui fait référence à la vue views.post_detail. En conséquence, Django s'attend à qu'une fonction post_detail soit définie dans le fichier `blog/views.py`.

Ouvrez maintenant le fichier `blog/views.py` dans l'éditeur de code et ajoutez la ligne suivante après les autres lignes du type from qui existent déjà :

`blog/views.py`

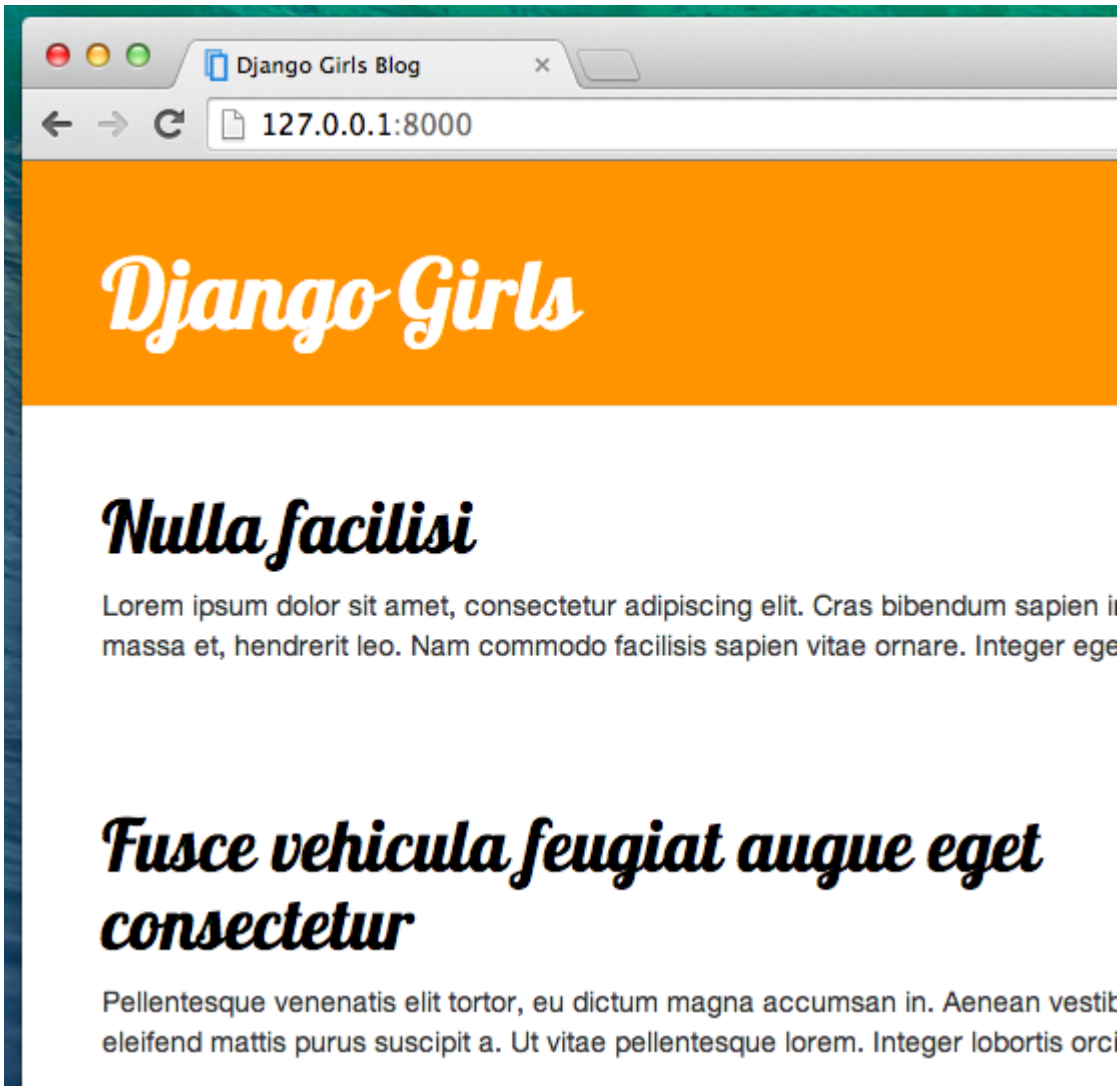
```
from django.shortcuts import render, get_object_or_404
```

A la fin du fichier nous ajoutons finalement notre view :

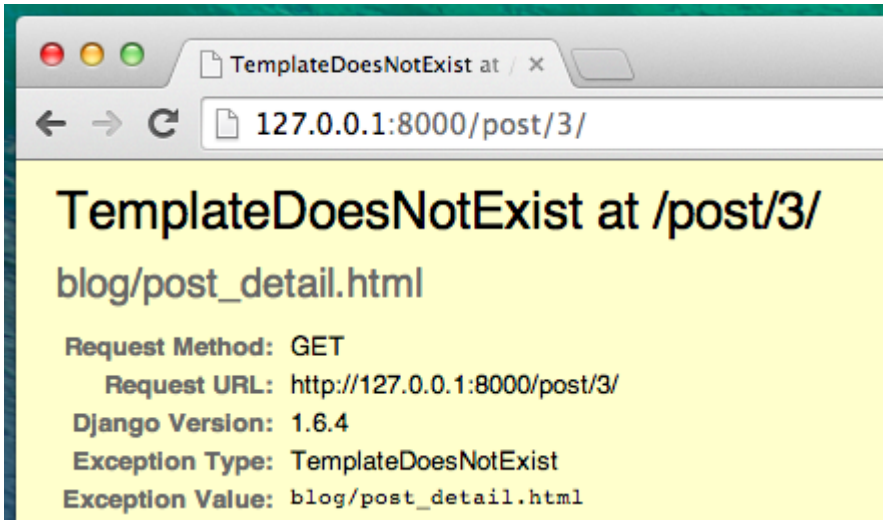
`blog/views.py`

```
def post_detail(request, pk):
    billet = get_object_or_404(Billet, pk=pk)
    return render(request, 'blog/post_detail.html', {'billet': billet})
```

Hop, réactualisons la page http://127.0.0.1:8000/



C'est bon, ça a marché ! Mais que ce passe-t-il lorsque nous cliquons sur un lien dans un titre de billet de blog ?



Oh non ! Encore une erreur ! Mais cette fois, vous savez quoi faire : nous avons besoin d'un template ! Créer un template pour le contenu du billet

Nous allons créer un fichier `post_detail.html` dans le dossier `blog/templates/blog` et nous l'ouvrons ensuite avec notre éditeur de code.

Entrez le code suivant :

`blog/templates/blog/post_detail.html`

```
{% block content %}
    <div class="billet">
        {% if billet.published_date %}
            <div class="date">
                {{ billet.published_date }}
            </div>
        {% endif %}
        <h2>{{ billet.title }}</h2>
        <p>{{ billet.text|linebreaksbr }}</p>
    </div>
{% endblock %}
```

Une nouvelle fois, nous faisons hériter de `base.html`. Dans le content block, nous voulons afficher la date de publication d'un billet (si elle existe), son titre et son texte. Mais vous souhaitez peut-être quelques éclaircissements avant, non?

`{% if … %} … {% endif %}` est une balise de template que nous pouvons utiliser si nous voulons vérifier quelque chose. (Vous souvenez-vous de if … else .. que nous avons appris dans le chapitre Introduction à Python ?) Dans ce cas spécifique, nous voulons vérifier que published_date existe.

Ok, vous pouvez maintenant actualiser votre page et voir si TemplateDoesNotExist a enfin disparu.

Yay ! Ça marche!

Encore bravo :)