Autonomie 1 : Une galerie en Django

En vous basant sur l'ensemble des TD précédents, vous devez à présent programmer un site web en Django qui reproduit les fonctionnalités de l'animalerie vue dans le cadre du modèle MVC. Attention, vous devez cette fois-ci personnaliser le modèle de données pour l'adapter à un univers de votre choix.

Exemples d'univers :

- Jeu de rôle
- Tamagotchi
- Pet shop
- Pocket Monster
- Aquarium
- Elevage d'escargots
- Course hippique
- Equipe de foot
- Athlètes
- etc.

Vous pouvez bien sûr conserver la structure de base qui est d'avoir un ensemble de créatures et/ou de personnages, et un ensemble de lieux avec des fonctions différentes (par exemple, pour un centre d'entrainement : terrain de foot, cantine, salle de muscu, dortoir). Il peut y avoir des variantes : Pour une maison des poupées il peut y avoir plusieurs chambres, une pièce commune, un jardin et une terrasse. Laissez libre cours à votre imagination.

Les créatures passent par différents états au cours de la journée en fonction des lieux qu'ils visitent.



Indications

Créez un projet 'playground' et reprenez toutes les étapes 1 à 15 du tutoriel précédent avec le modèle suivant, en remplaçcant "Animal" par "Character" pour être plus général.

Last update: 2023/11/04 public:appro-s7:td_web_hamsters https://wiki.centrale-med.fr/informatique/public:appro-s7:td_web_hamsters?rev=1699113122 16:52

model.py

```
from django.db import models
class Equipement(models.Model):
    id equip = models.CharField(max length=100, primary key=True)
    disponibilite = models.CharField(max_length=20)
    photo = models.CharField(max length=200)
    def __str_(self):
        return self.id_equip
class Character(models.Model):
    id_char = models.CharField(max_length=100, primary_key=True)
    etat = models.CharField(max length=20)
    type = models.CharField(max length=20)
    race = models.CharField(max length=20)
    photo = models.CharField(max length=200)
    lieu = models.ForeignKey(Equipement, on delete=models.CASCADE)
    def str (self):
        return self.id char
```

Vous deviez obtenir une interface simple de ce type:

| Bienvenue à l'animalerie | - Mozilla Firefox | _ = = | × |
|--|----------------------|---------------|---------|
| < bli 🔵 Jardii 🛛 CSS Zen 🛛 🛄 Form 🔹 https://ti 🖉 E | mm Nouvel o 🔵 Ho | Bienv × > + | ~ |
| \leftarrow \rightarrow C $\textcircled{1}$ $\rule{1}{1}$ | 😭 🔍 Rechercher | 👱 III\ 😇 >> 🗄 | |
| Bienvenue à l'animalerie! | | | |
| Tic | repus | mangeoire | |
| Тас | affamé | litiere | |
| Patrick | affamé | litiere | |
| Totoro | endormi | nid | |
| Pocahontas | affamé | litiere | |

Il est bien sûr possible d'améliorer les choses en utilisant des fonctions de mise en page plus évoluées

La mise en page du site repose sur l'utilisation de fichiers de style (css) ainsi que de composants javascript de type :

- Web Front, option S7 (F. Brucker)
 - bootstrap

 et : w3school
 - materialize
 - (voir aussi ce tutoriel).

Vous pouvez exploiter les photos qui sont définies dans les attributs du modèle pour obtenir par exemple:



Formulaires Django

(repris du tutoriel django_forms)

La dernière chose que nous voulons faire sur notre site web, c'est créer une manière de mettre a jour l'etat des personnages. Les formulaires (forms) vont nous permettre de rendre le site dynamique !

Comme toutes les choses importantes dans Django, les formulaires ont leur propre fichier : forms.py.

Nous allons devoir créer un fichier avec ce nom dans notre dossier blog.

```
playground
└── forms.py
```

Ouvrez maintenant ce fichier dans l'éditeur de code et tapez le code suivant :

```
from django import forms
from .models import Character
class MoveForm(forms.ModelForm):
    class Meta:
        model = Character
        fields = ('lieu',)
```

Tout d'abord, nous avons besoin d'importer les formulaires Django (from django import forms), puis notre modèle Character (from .models import Character).

Comme vous l'avez probablement deviné, MoveForm est le nom de notre formulaire. Nous avons besoin d'indiquer à Django que ce formulaire est un ModelForm (pour que Django fasse certaines choses automatiquement pour nous). Pour cela, nous utilisons forms.ModelForm.

Ensuite, nous avons la class Meta qui nous permet de dire à Django quel modèle il doit utiliser pour créer ce formulaire (model = Character).

Enfin, nous précisions quel·s sont le·s champ·s qui doivent figurer dans notre formulaire. Dans notre cas, nous souhaitons que seul le lieu apparaisse dans notre formulaire.

Et voilà, c'est tout ! Tout ce qu'il nous reste à faire, c'est d'utiliser ce formulaire dans une vue et de l'afficher dans un template.

Nous allons donc une nouvelle fois suivre le processus suivant et créer : un lien vers la page, une URL, une vue et un template.

Lien vers une page contenant le formulaire

Il est temps d'ouvrir playground/templates/playground/base.html dans l'éditeur de code et ajouter un lien vers la vue character_detail.

```
<a href="{% url 'character_detail' id_character=character.id_character
%}">{{ character.id_character }}</a>
```

URL

Ouvrez le fichier

playground/urls.py

dans l'éditeur de code et mettez ceci:

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

```
urlpatterns = [
    path('', views.character_list, name='character_list'),
    path('character/<str:id_character>/', views.character_detail,
name='character_detail'),
    path('character/<str:id_character>/?<str:message>',
views.character_detail, name='character_detail_mes'),
]
```

La vue character_detail

Ouvrez maintenant le fichier playground/views.py dans l'éditeur de code et ajoutez les lignes suivantes avec celles du from qui existent déjà :

Afin de pouvoir créer un nouveau formulaire Move, nous avons besoin d'appeler la fonction MoveForm() et de la passer au template. Nous reviendrons modifier cette *vue* plus tard, mais pour l'instant, créons rapidement un template pour ce formulaire.

Template

Nous avons à présent besoin de créer un fichier character_detail.html dans le dossier playground/templates/playground et de l'ouvrir dans l'éditeur de code. Afin que notre formulaire fonctionne, nous avons besoin de plusieurs choses :

- Nous avons besoin d'afficher le formulaire. Pour cela, nous n'avons qu'à utiliser form.as_uk
- La ligne précédente va avoir besoin d'être entourée des balises HTML < form method="POST">...</form>.
- 3. Nous avons besoin d'un bouton Valider. Nous allons le créer à l'aide d'un bouton HTML :

Valider.

4. Enfin, nous devons ajouter {% csrf_token %} juste après <form ...>. C'est très important car c'est ce qui va permettre de sécuriser votre formulaire ! Si vous oubliez ce détail, Django se plaindra lorsque vous essaierez de sauvegarder le formulaire:

×

Ok, voyons maintenant à quoi devrait ressembler le HTML contenu dans le fichier character_detail.html :

{% endblock %}

Rafraîchissons la page ! Et voilà : le formulaire s'affiche sous la forme d'une liste d'options!

```
Mais attendez une minute! Lorsque vous sélectionnez une option, que se passera-t-il? Rien!
Retournons à notre /vue. ==== Sauvegarder le contenu du formulaire ==== <html> Ouvrez à
nouveau <code>blog/views.py</code> dans l&apos;éditeur de code. Actuellement,
<code>post new</code> n&apos;est composé que des lignes de code suivantes : </html>
<code python> def character detail(request, id character): character = get object or 404(Character,
id_character=id_character) form=MoveForm() return render(request,
'playground/character_detail.html', {'character': character, 'lieu': lieu, 'form': form}) </code> <html>
Lorsque nous envoyons notre formulaire, nous revenons à la même vue. Cependant, nous
récupérons les données dans <code>request</code>, et plus particulièrement dans
<code>request.POST</code>. Vous rappelez-vous comment dans le fichier HTML, notre définition de
la variable <code>&lt;form&gt;</code> avait la méthode
<code>method=&quot;POST&quot;</code>? Tous les champs du formulaire se trouvent maintenant
dans <code>request.POST</code>. Veillez à ne pas renommer <code>POST</code> en quoi que ce
soit d'autre : la seule autre valeur autorisée pour <code>method</code> est
<code>GET</code>. Malheureusement, nous n&apos;avons pas le temps de rentrer dans les détails
aujourd'hui. Donc dans notre <em>vue</em> nous avons deux situations différentes
à gérer : la première quand on accède à la page pour la première fois et nous voulons un formulaire
vide, et la seconde quand on revient à la <em>vue</em> avec les données que l&apos;on a saisies
dans le formulaire. Pour gérer ces deux cas, nous allons utiliser une condition </html> <code
python> if request.method == "POST": [...] else: form = MoveForm() </code> <html> Il faut
maintenant remplir à l'endroit des pointillés <code>[...]</code>. Si <code>method</code>
```

contient <code>POST</code> alors on veut construire le <code>MoveForm</code> avec les données du formulaire, n'est-ce pas ? Nous allons le faire comme cela : </html> <code python> form = MoveForm(request.POST, instance=character) </code> <html> La prochaine étape est de vérifier que le formulaire a été rempli correctement (tous les champs obligatoires ont été remplis et aucune valeur incorrecte n'a été envoyée). Nous allons faire ça en utilisant <code>form.is_valid()</code>. Testons donc si notre formulaire est valide et, si c'est le cas, sauvegardons-le ! </html> <code python> if form.is valid(): ancien lieu = get_object_or_404(Equipement, id_equip=character.lieu.id_equip) ancien_lieu.disponibilite = "libre" ancien lieu.save() form.save() nouveau lieu = get object or 404(Equipement, id equip=character.lieu.id equip) nouveau lieu.disponibilite = "occupé" nouveau lieu.save() </code> <html> En gros, nous effectuons deux choses ici : nous sauvegardons le nouvel état du personnage grâce à <code>form.save</code> et nous mettons à jour l'occupation des lieux. Rappelez vous, tout déplacement du personnage s'acccompagne d'un changement d'occupation. Nous devons également modifier les lieux. <code>ancien_lieu.save()</code> et <code>nouveau lieu.save()</code> sauvegarderont les changements. Et voilà, la mise à jour est enregistrée ! Enfin, ce serait génial si nous pouvions tout de suite aller à la page <code>character detail</code> avec le contenu que nous venons de créer. Pour cela, nous avons besoin d' importer une dernière chose : Maintenant, nous allons ajouter la ligne qui signifie "aller à la page <code>character detail</code> pour le changement qui vient d'être enregistré </html> <code python> return redirect('character detail', id character=id character) </code> <html> <code>character detail</code> est le nom de la vue où nous voulons aller. Voyons à quoi ressemble maintenant notre vue ? </html> <code python> def character_detail(request, id_character): character = get_object_or_404(Character, id character=id character) form=MoveForm() if form.is_valid(): ancien_lieu = get object or 404(Equipement, id equip=character.lieu.id equip) ancien lieu.disponibilite = "libre" ancien lieu.save() form.save() nouveau lieu = get object or 404(Equipement, id equip=character.lieu.id equip) nouveau lieu.disponibilite = "occupé" nouveau lieu.save() return redirect('character detail', id character=id character) else: form = MoveForm() return render(request, 'playground/character_detail.html', {'character': character, 'lieu': lieu, 'form': form}) </code> <html> Voyons si ça marche. Allez à l'adresse http://127.0.0.1:8000/character/Tic/, selectionnez un nouveau lieu, sauvegardez ... et voilà ! La mise a jour est prise en compte ! </html> ==== Modèle complet ==== Il ne reste plus qu'à raffiner le modèle afin qu'il corresponde aux contraintes définies en début d'énoncé. * Si le lieu de destination n'est pas libre, alors le changement ne doit pas être accepté. Il ne faut donc pas sauvegarder l'état du personnage immédiatement. Cela est possible grâce à l'option : <code python> form.save(commit=False) </code> * Avec cette option, le lieu est mis à jour dans l'objet character mais pas dans la base de données! Il est alors possible de tester si le nouveau lieu est bien libre en regardant si character.lieu.disponibilite est "libre". * Si c'est bon, alors effectuez les changements et n'oubliez pas de sauvegarder avec character.save(). * Si ce n'est pas le cas, alors les changements ne doivent pas être enregistrés. * N'oubliez pas non plus: le changement de lieu a pour conséquence un changement d'état, qui passe par exemple d' "affamé" à "repus" lorsqu'on le déplace sur la mangeoire. Reprenez toutes les conditions et tenez en compte pour la mise à jour du personnage. * Il est bien sûr possible de reprendre en l'adaptant la structure de contrôleur vue au TD1. ==== Les messages ==== Lorsqu'un changement n'est pas autorisé, il est préférable d'afficher un message d'avertissement. Le message transmis comme paramètre au template character_detail.html. Le cadre sera activé uniquement si le message est non vide. Pensez à modifier les paramètres d'appel du template dans views.py. <code html> {% if message != "%}

```
{{message}}
</div>
{% endif %}
```

</code>

Encore un petit effort : déployons !

Nos modifications fonctionnent-elles sur PythonAnywhere ? Pour le savoir, déployons à nouveau ! Tout d' abord, commitez votre nouveau code et pushez le à nouveau sur GitHub: label">command-line <code>\$ git status \$ git add . \$ git status \$ git commit -m " Added views to create/edit blog post inside the site." \$ git pus </code> Puis, dans la console bash de <a</pre> href="https://www.pythonanywhere.com/consoles/" target=" blank">PythonAnywhere: label">Ligne de commande PythonAnywhere <code>\$ cd ~/<your-pythonanywhere-domain>.pythonanywhere.com \$ git pull [...] </code>(N'oubliez pas de remplacer <code><your-pythonanywheredomain&qt;</code> avec votre propre sous-domaine PythonAnywhere, sans les chevrons.) Enfin, allez sur "Web" page (utilisez le bouton de menu en haut à droite de la console) et cliquez Reload. Actualisez votre blog <a href="https://subdomain.pythonanywhere.com"</pre> target=" blank">https://subdomain.pythonanywhere.com pour voir les changements. Et normalement c'est tout ! Félicitations ! :) </section>

From: https://wiki.centrale-med.fr/informatique/ - **WiKi informatique**

Permanent link: https://wiki.centrale-med.fr/informatique/public:appro-s7:td_web_hamsters?rev=169911312



Last update: 2023/11/04 16:52