HINTS

PRÁCTICAS RECOMENDADAS DE DJANGO: PROYECTOS FRENTE A APLICACIONES.

La definición de Django de una "aplicación", a menudo es confusa para los recién llegados. Examinaremos los cuatro conceptos principales de la arquitectura Django mediante la creación de una aplicación web de blog básica.

PROYECTO

Es una aplicación web que utiliza Django. Solo hay un proyecto, y muchas "aplicaciones" dentro de él. Entonces, para nuestra aplicación web de blog, debemos crearla y asignarle un nombre como `config`.

BOARDS/ADMIN.PY

```
$ django-admin startproject config .
$ tree
.
├── Pipfile
├── Pipfile.lock
├── config
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
└── manage.py
```

Tenga en cuenta que se agregó al final un punto (`.`) del comando para que los archivos se incluyan en el directorio actual. De lo contrario, Django crearía automáticamente un directorio adicional con el nombre de nuestro proyecto, y luego agregaría los archivos de inicio dentro de ese directorio. Esto pareciera redundante, pero algunos desarrolladores prefieren ese enfoque.

APLICACIONES_INSTALADAS

Dentro del archivo recién creado settings.py hay una configuración llamada `INSTALLED_APPS`, que es una lista de aplicaciones de Django dentro de un proyecto. Django viene con seis aplicaciones integradas que podemos examinar.

CONFIG/SETTINGS.PY

```
# config/settings.py
INSTALLED_APPS = [
```

```
    'django.contrib.admin' ,
    'django.contrib.auth' ,
    'django.contrib.contenttypes' ,
    'django.contrib.sessions' ,
    'django.contrib.messages' ,
    'django.contrib.staticfiles' ,
]
```

APLICACIONES

Una aplicación Django es una pequeña biblioteca que representa una parte discreta de un proyecto más grande. Por ejemplo: nuestra aplicación web de blog podría tener una aplicación para posts, una para páginas estáticas como una página “Acerca de” llamada `pages`, y otra aplicación llamada `payments` para cobrar a los suscriptores registrados.

Podemos agregar una aplicación usando el comando `startapp`, así que agreguemos la aplicación post.

CONFIG/SETTINGS.PY

```
(code) $ python manage.py startapp posts
(code) tree
.
├── Pipfile
├── Pipfile.lock
├── config
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── manage.py
└── posts
    ├── __init__.py
    ├── admin.py
    ├── apps.py
    ├── migrations
    │   └── __init__.py
    ├── models.py
    ├── tests.py
    └── views.py
```

CONVENCIONES DE NOMENCLATURA DE APLICACIONES

El nombre de una aplicación debe seguir las pautas de Pep 8, es decir, debe ser breve, todo en minúsculas y no incluir números, guiones, puntos, espacios ni caracteres especiales. También, en

general, debe ser el plural del modelo principal de una aplicación, por lo que nuestra aplicación posts tendría un modelo principal llamado Post.

PRÁCTICAS RECOMENDADAS DE DJANGO: PERMISOS DE USUARIO

Por lo general, los permisos se establecen en el archivo `views.py`. La vista actual para actualizar una publicación de blog existente `BlogUpdateView`, tiene el siguiente aspecto:

```
# blog/views.py
clase  BlogUpdateView ( UpdateView ):
    modelo  =  Publicar
    template_name  =  'post_edit.html'
    campos  =  [ 'título' ,  'cuerpo' ]
```

Ahora supongamos que queremos que un usuario inicie sesión antes de que pueda acceder a `BlogUpdateView`. Hay varias formas de hacerlo, pero la más simple es usar el `LoginRequiredMixin` integrado. El `mixin` es un orden de izquierda a derecha, por lo que se debe agregar el `mixin` de inicio de sesión antes de `UpdateView`. Eso significa que si un usuario no ha iniciado sesión, le aparecerá un mensaje de error.

```
# blog/views.py
from django.contrib.auth.mixins import LoginRequiredMixin

class BlogUpdateView(LoginRequiredMixin, UpdateView):
    model = Post
    template_name = 'post_edit.html'
    fields = ['title', 'body']
```

USERPASSESTESTMIXIN

Un permiso de siguiente nivel es algo específico para el usuario. En este caso, hagamos cumplir la regla de que solo el autor de una publicación de blog puede actualizarla. Podemos usar el `UserPassesTestMixin` incorporado para esto.

```
# blog/views.py
from django.contrib.auth.mixins import LoginRequiredMixin,
UserPassesTestMixin

class BlogUpdateView(LoginRequiredMixin, UserPassesTestMixin,
UpdateView):
    model = Post
    template_name = 'post_edit.html'
```

```
 9      fields = ['title', 'body']
10
11      def test_func(self):
12          obj = self.get_object()
13          return obj.author == self.request.user
```

Tenga en cuenta que importamos `UserPassesTestMixinen` en la parte superior, y lo agregamos en segundo lugar en nuestra lista de `mixins` para `BlogUpdateView`. Eso significa que un usuario primero debe iniciar sesión, y luego debe pasar la prueba de usuario antes de acceder `UpdateView`. ¿Podemos poner `UserPassesTestMixinprimero`? Sí, pero generalmente es mejor comenzar con los permisos más generales, y luego volverse más granular a medida que avanza hacia la derecha.

El `test_func` es un método utilizado por `UserPassesTestMixinpara`. Tenemos que anularlo. En este caso, establecemos la variable `obj` en el objeto actual devuelto por la vista usando `get_object`. Luego, decimos si `author` en el objeto actual coincide con el usuario actual en la página web (quien haya iniciado sesión e intente realizar el cambio), entonces permítalo. Si es falso, se arrojará un error.

Hay otras formas de establecer permisos por usuario, incluida la anulación del método de envío, pero `UserPassesTestMixines` es elegante y está diseñado específicamente para este caso de uso.