Grado en Ingeniero en Informática
Programación Web
Enero 2016

Examen de preguntas de respuesta corta

Responda de forma breve y razonada a las siguientes cuestiones:

1. Explique qué hace y cómo funciona el siguiente segmento de código de Django:

```
def funcion(request, distillery_name):
    distillery = get_object_or_404(Distillery, pk = distillery_name)
    if request.method == 'POST':
        form = DistilleryForm(request.POST, request.FILES, instance = distillery)
        if form.is_valid():
            form.save()
            return redirect(reverse_lazy('distillerys:distillerys_list'))
    else:
        form = DistilleryForm(instance = distillery)
    context = {'form':form}

    return render(request, 'bottles/page.html', context)
```

RESPUESTA: Se trata del código típico para editar un registro. Primero se lee el registro de la base de datos, en caso de que la petición sea GET se rellena el formulario con el registro y se presente en la plantilla. Una vez se completa el formulario la petición será de tipo POST. Entonces se obtiene la información introducida, se valida y en caso de ser correcta se salva en la base de datos y se redirige a una vista que presenta una lista de registros.

2. Indique cómo puede proteger una vista en Django para evitar que un usuario no autenticado acceda a ella, tanto para el caso de vistas como funciones como para el caso de vistas basadas en clases.

RESPUESTA: En el caso de vistas como funciones podemos usar un decorador:

```
@login_required(login_url = '/users/login')
def distillery_new(request):
```

Para el caso de vistas basadas en clases podemos decorar el resultado del resultado del método `as_view()`:

```
urlpatterns = patterns('',
    (r'^about/', login_required(TemplateView.as_view(template_name="secret.html"))),
    (r'^vote/', permission_required('polls.can_vote')(VoteView.as_view())),
)
```

En este caso solo se protege esta instancia. Si queremos proteger cada instancia del método debemos decorar el método `dispatch()` de la clase:

```
class ProtectedView(TemplateView):
    template_name = 'secret.html'
    @method_decorator(login_required)
    def dispatch(self, *args, **kwargs):
        return super(ProtectedView, self).dispatch(*args, **kwargs)
```

3. Indique qué ventajas poseen los protocolos texto, como por ejemplo HTTP.

   RESPUESTA: La principal ventaja es poder ignorar la diferencia de codificación entre equipos al usar solo la parte estándar del código ASCII.

4. ¿Qué diferencias hay entre los métodos POST y GET en el protocolo HTTP? ¿Para qué peticiones es adecuado cada uno de ellos?

   RESPUESTA: El método GET permite pasar información del cliente al servidor en forma de texto que se añade a la dirección URL. El método POST envía información pero de forma codificada. GET debe usarse para enviar información reducida y sin problemas de seguridad, como por ejemplo un término a buscar. POST debe usarse para todo los demás. En particular no debe usarse GET para enviar información que pueda modificar la base de datos.

5. Los formularios en HTML5 permiten la validación por el navegador de la información introducida antes de enviar la información al servidor. ¿Qué utilidad tiene dicha opción?

   RESPUESTA: Tiene dos ventajas principales. Por un lado permite descargar al servidor de una operación trivial de validación que puede ser llevada a cabo de forma eficiente por el cliente. También evita la sobrecarga de enviar información no válida al servidor que habrá de ser reenviada una vez corregida.

6. Suponga que se desea presentar en una plantilla en Django el resultado de una lista de objetos ordenada. Esta lista se puede ordenar en el acceso a la base de datos, en la vista en Python o en la plantilla en HTML. ¿Dónde sería más eficiente la ordenación y por qué?

   RESPUESTA: La ordenación, y en general cualquier operación sobre los datos, es siempre más eficiente a más bajo nivel, porque los métodos están más optimizados. Así será mejor que se haga la ordenación en el acceso a la base de datos.

7. Considere la siguiente clase en Django que modela un partido de fútbol entre dos equipos:

   ```
   class Partido(models.Model):
       lugar = models.CharField(max_length = 100)
       resultado = models.CharField(max_length = 50)
       equipo1 = models.ForeignKey(Equipo)
       equipo2 = models.ForeignKey(Equipo)
   ```

   Cuando intentamos generar el modelo correspondiente en la base de datos obtendríamos un error. ¿Cuál es el error y cómo lo podríamos solucionar?

   RESPUESTA: La clase `Partido` tiene dos relaciones 1:N con la clase `Equipo`. Esto es perfectamente válido pero genera un problema en Django. La relación inversa se crea por defecto cada vez que creamos una relación así, de esta forma, una campo con el nombre `partido` sería creado dos veces en la clase `Equipo`. Podemos evitar el problema añadiendo la opción:

   ```
   class Partido(models.Model):
       lugar = models.CharField(max_length = 100)
   ```

```
    resultado = models.CharField(max_length = 50)
    equipo1 = models.ForeignKey(Equipo, related_name="equipo1")
    equipo2 = models.ForeignKey(Equipo, related_name="equipo2")
```

O podemos elegir no crear la relación inversa con la opción: `related_name='+'`.

8. ¿Qué ventajas tiene el nombrar los patrones URL en Django como en el ejemplo siguiente?

```
from django.conf.urls import patterns, url
from mysite.views import archive
urlpatterns = patterns('',
    url(r'^archive/(\d{4})/$', archive, name="full-archive"),
    url(r'^archive-summary/(\d{4})/$', archive, name="arch-summary"),
)
```

RESPUESTA: Podemos usar el nombre de la dirección correspondiente en nuestras plantillas lo que hace los proyectos muchos más sencillamente mantenibles: Por ejemplo:

```
<a href="{% url 'bottles:bottle' bottle.id %}" alt="">{{ bottle.name }}</a>
```

9. ¿Qué diferencias de funcionamiento existirían entre las dos implementaciones de la función `detail()` siguientes en Django?

```
def detail(request, question_id):
    question = Question.objects.get(pk=question_id)
    return render(request, 'polls/detail.html', {'question': question})

def detail(request, question_id):
    question = get_object_or_404(Question, pk=question_id)
    return render(request, 'polls/detail.html', {'question': question})
```

RESPUESTA: En el primer caso si el objeto accedido no existe en la base de datos habrá un error que no está considerando y el servicio web se suspenderá. En el segundo caso el error será capturado y se generará una excepción 404.

10. ¿Qué ventajas presentan en Django las vistas basadas en clases con respecto a las vistas como funciones?

RESPUESTA: La principal ventaja es la posibilidad de usar la herencia para reusar el código programado. Esto unido al uso de vistas genéricas permite desarrollar proyectos con muy poco esfuerzo.

Tiempo de realización: **1** hora. Todas las cuestiones tienen la misma calificación de 1 punto.