**Trabajo Práctico “Proyecto de implementación de sitios web dinámicos” 26/08/2019**

Fecha de entrega: 04/09/2019

Apellido y Nombre: Gerardo Ezequiel Valdez

1) ¿Cuál es la última versión de Django? ¿Cuál es la más antigua que aún cuenta con soporte? ¿hasta cuándo será soportada?

2) ¿En qué archivo se definen los Modelos de Django?

3) ¿De qué clase heredan?

4) ¿Que es un campo dentro de un modelo?

5) ¿Cuantos tipos de campos incluye Django? Mencionar en detalle al menos 3 tipos.

6) ¿Que son los argumentos de los campos? Explicar al menos 2.

7) ¿Qué podemos definir con los metadatos?

8) ¿Qué es un método? ¿Cuáles son los más usados en proyectos Django?

9) ¿Cómo se relacionan los modelos con la base de datos? ¿Qué es un ORM?

10) ¿Cómo especifico que campo será la Primary Key de una tabla? ¿Qué pasa si no lo defino explícitamente?

11) ¿Que son las relaciones entre modelos? Definir y explicar cada tipo con ejemplos.

12) ¿Qué es una API? ¿Cuáles son sus ventajas? ¿Cómo puedo ofrecer una API en mi proyecto Django?

Respuestas

1. La última versión es: 2.1.12, la versión más vieja con soporte es la: 1.11.24 y será soportada hasta abril de 2020.
2. Los modelos se van a definir en models.py el cual está situado dentro de la carpeta blog
3. Hereda la clase Post la cual muestra el autor, el título, el texto, fecha de creación y de publicación.
4. Los campos en el modelo funcionan como un intermediario entre los valores de base de datos y las estructuras nativas de Python, con ello cuando accedemos a un atributo de un modelo, este es una representación de los datos que tenemos en base de datos.
5. Tiene más de 20 campos.

+ Charfield: es un campo de cadena, para cadenas de tamaño pequeño a grandes, para cadenas de grandes cantidades de texto es mejor usar TextField.

+ AutoField: es una IntegerField que se incrementa automáticamente según las ID disponibles. Por lo general, no necesitará usar esto directamente; un campo de clave primaria se agregará automáticamente a su modelo si no especifica lo contrario.

+ FloatField: es un número de coma flotante representado en Python por una instancia float.

1. Un argumento es un conjunto de premisas al que sigue una conclusión. Cuando esta conclusión se sigue necesariamente de las premisas, se habla de un argumento deductivamente válido.

+ verbose\_name: Nombre de fácil lectura que se usa en etiquetas para el campo. Si no se especifica, Django inferirá el valor por defecto del verbose name a partir del nombre del campo.

+ choices: Un grupo de valores de selección para este campo. Si se proporciona, el widget correspondiente por defecto del formulario será una caja de selección con estos valores de selección en vez del campo de texto estándar.

1. "cualquier cosa que no sea un campo", como las opciones de pedido (ordering), el nombre de la tabla de la base de datos (db\_table) o los nombres en singular y plural legibles por humanos (verbose\_namey verbose\_name\_plural). No se requiere ninguno, y agregar a un modelo es completamente opcional. class Meta
2. -clean (value, instance)

-db\_type(connection)

-formfield ()

-get\_internal\_type ()

-pre\_save (model\_instance, add)

-save\_form\_data (instance, data)

-validate (value, instance)

1. Claramente, el poder de las bases de datos relacionales reside en relacionar tablas entre sí. Django ofrece formas de definir los tres tipos más comunes de relaciones de bases de datos: muchos a uno, muchos a muchos y uno a uno.

-Relaciones de muchos a uno

Para definir una relación de muchos a uno, use django.db.models.ForeignKey. Lo usa como cualquier otro Field tipo: incluyéndolo como un atributo de clase de su modelo.

ForeignKey requiere un argumento posicional: la clase con la que se relaciona el modelo.

-Relaciones de muchos a muchos

Para definir una relación de muchos a muchos, usa ManyToManyField. Lo usa como cualquier otro tipo Field: incluyéndolo como un atributo de clase de su modelo.

ManyToManyField requiere un argumento posicional: la clase con la que se relaciona el modelo.

-Relaciones uno a uno

Para definir una relación uno a uno, usa OneToOneField. Lo usa como cualquier otro Fieldtipo: incluyéndolo como un atributo de clase de su modelo.

Esto es más útil en la clave primaria de un objeto cuando ese objeto "extiende" otro objeto de alguna manera.

OneToOneField requiere un argumento posicional: la clase con la que se relaciona el modelo.

\*El ORM (Object-relational mapping) de Django es una herramienta poderosa y tal vez uno de los puntos más fuertes del framework. Nos facilita todo lo relacionado a consultas, actualización, eliminar, listar todo lo almacenado en nuestra base de datos de una manera fácil de entender.

1. Se define poniendo primary\_key=True en uno de los campos.

Si Django ve que ha establecido explícitamente Field.primary\_key, no agregará la id columna automática.

Si no especifica primary\_key=True ningún campo en su modelo, Django agregará automáticamente una IntegerField para mantener la clave principal, por lo que no necesita configurar primary\_key=True ninguno de sus campos a menos que desee anular el comportamiento predeterminado de la clave principal.

11)- Los modelos definen la estructura de los datos almacenados, incluidos los tipos de campo y los atributos de cada campo, como su tamaño máximo, valores predeterminados, lista de selección de opciones, texto de ayuda para la documentación, texto de etiqueta para formularios, etc. La definición del modelo es independiente de la base de datos subyacente. Puede elegir una de entre varias como parte de la configuración de su proyecto. Una vez que haya elegido la base de datos que desea usar, no necesita hablar directamente con ella. Simplemente escriba la estructura de su modelo y algo de código, y Django se encargará de todo el trabajo sucio, al comunicarse con la base de datos por usted.

**Relación de muchos a uno**

from django.db import models

class Author(models.Model):

name = models.CharField(max\_length=50)

#Book has a foreignkey (many to one) relationship with author

class Book(models.Model):

author = models.ForeignKey(Author, on\_delete=models.CASCADE)

publish\_date = models.DateField()

La opción más genérica. Se puede utilizar en cualquier lugar que desee representar una relación.

**Relación de muchos a muchos**

class Topping(models.Model):

name = models.CharField(max\_length=50)

# One pizza can have many toppings and same topping can be on many pizzas

class Pizza(models.Model):

name = models.CharField(max\_length=50)

toppings = models.ManyToManyField(Topping)

**Internamente esto se representa a través de otra tabla. Y ManyToManyField debe colocar en modelos que se ManyToManyField en un formulario. Por ejemplo: la Appointment tendrá un ManyToManyField llamado Customer , Pizza tiene Toppings y así sucesivamente.**

**Relación de muchos a muchos usando clases directas**

class Service(models.Model):

name = models.CharField(max\_length=35)

class Client(models.Model):

name = models.CharField(max\_length=35)

age = models.IntegerField()

services = models.ManyToManyField(Service, through='Subscription')

class Subscription(models.Model):

client = models.ForeignKey(Client)

service = models.ForeignKey(Service)

subscription\_type = models.CharField(max\_length=1, choices=SUBSCRIPTION\_TYPES)

created\_at = models.DateTimeField(default=timezone.now)

De esta manera, podemos mantener más metadatos sobre una relación entre dos entidades. Como puede verse, un cliente puede suscribirse a varios servicios a través de varios tipos de suscripción. La única diferencia en este caso es que para agregar nuevas instancias a la relación M2M, no se puede usar el método abreviado pizza.toppings.add(topping) , sino que se debe crear un nuevo objeto de la clase through , Subscription.objects.create(client=client, service=service, subscription\_type='p')

En otros idiomas, las through tables también se conocen como JoinColumn , Intersection table o mapping table

**Relación uno a uno**

class Employee(models.Model):

name = models.CharField(max\_length=50)

age = models.IntegerField()

spouse = models.OneToOneField(Spouse)

class Spouse(models.Model):

name = models.CharField(max\_length=50)

Utilice estos campos cuando solo tendrá una relación de composición entre los dos modelos.

12) Una API es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones. API significa interfaz de programación de aplicaciones.

Las API permiten que sus productos y servicios se comuniquen con otros, sin necesidad de saber cómo están implementados. Las API le otorgan flexibilidad; simplifican el diseño, la administración y el uso de las aplicaciones, y proporcionan oportunidades de innovación, lo cual es ideal al momento de diseñar herramientas y productos nuevos (o de gestionar los actuales).

Las API son un medio simplificado para conectar su propia infraestructura a través del desarrollo de aplicaciones nativas de la nube, pero también le permiten compartir sus datos con clientes y otros usuarios externos. Las API públicas representan un valor comercial único porque simplifican y amplían la forma en que se conecta con sus partners y, además, pueden rentabilizar sus datos.

Ventajas

1. separación cliente/servidor.
2. Independencia de tecnologías / lenguajes.
3. Fiabilidad, escalabilidad, flexibilidad.
4. Experiencia de usuario.
5. requiere menos recursos del servidor.