El framework web para perfeccionistas con plazos

- Reyes González Agustín Óscar
- Vázquez Zaragoza Jesús Arturo

Framework

Un framework es un entorno o marco de trabajo, que está conformado por un conjunto estandarizado de herramientas, criterios y buenas prácticas que toman una problemática en particular y la utilizan como referencia para enfrentar y resolver futuros problemas de índole similar.

¿Qué es Django?

Es un framework web escrito en **Python** de alto nivel que fomenta un desarrollo rápido con un diseño limpio y pragmático. Creado por desarrolladores experimentados, se ocupa de gran parte de las molestias del desarrollo web, por lo que puede concentrarse en escribir su aplicación sin necesidad de reinventar la rueda. Es gratis y de código abierto.

Algunas tareas que agiliza

- Creación de formularios
- Autenticación y permisos de usuarios
- Cacheo
- Crea un panel de administración
- Serialización de objetos

Historia

Django nació de aplicaciones de la vida real escritas por un equipo de desarrolladores Web en Lawrence, Kansas. Nació cuando los programadores Web del diario Lawrence Journal-World, Adrian Holovaty y Simon Willison, comenzaron a usar Python para crear sus aplicaciones.

Para los sitios, los periodistas exigían que se agregaran nuevas características y que aplicaciones enteras se crearan a una velocidad vertiginosa, a menudo con sólo días u horas de preaviso. Es así que Adrian y Simon desarrollaron por necesidad un framework de desarrollo Web que les ahorrara tiempo.

Luego de haber desarrollado este framework hasta el punto en que estaba haciendo funcionar la mayoría de los sitios World Online, el equipo de World Online, que ahora incluía a Jacob Kaplan-Moss, decidió liberar el framework como software de código abierto en julio de 2005.

Dato curioso

Lo llamaron Django, por el guitarrista de jazz Django Reinhardt.

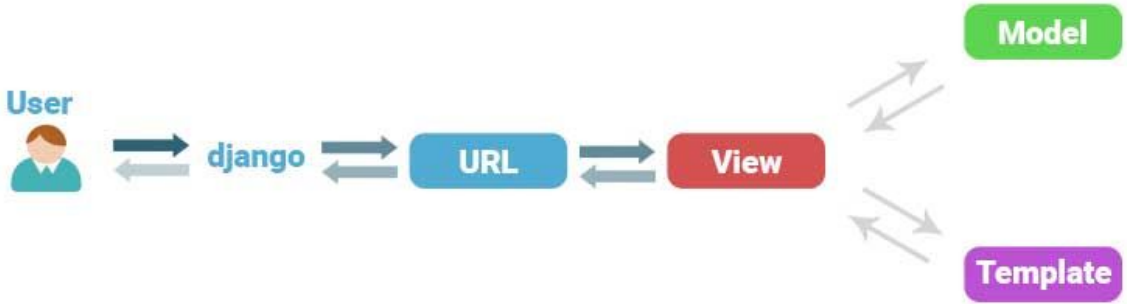
Ventajas

Con Django, puede llevar aplicaciones web desde el concepto hasta el lanzamiento en cuestión de horas. Django cumple con las siguientes características:

- Rápido
- Completamente cargado
- Seguro
- Escalable
- Versátil
- Portable

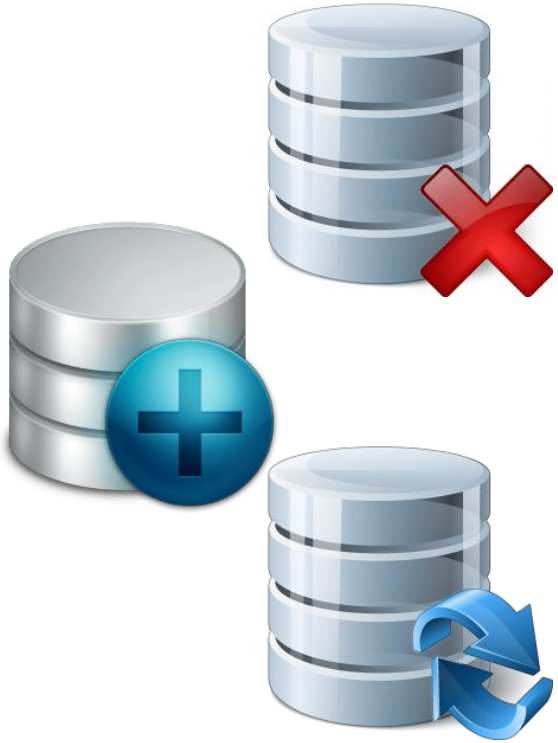
¿Cómo trabaja?

Django trabaja bajo el modelo MTV (Model Template View) el cual es equivalente al modelo vista controlador pero con ciertas diferencias en cuanto a funcionalidades en cada uno de los componentes que la conforman



Model

Los Modelos son objetos de Python que definen la estructura de los datos de una aplicación y proporcionan mecanismos para gestionar (añadir, modificar y borrar) y consultar registros en la base de datos.



Template

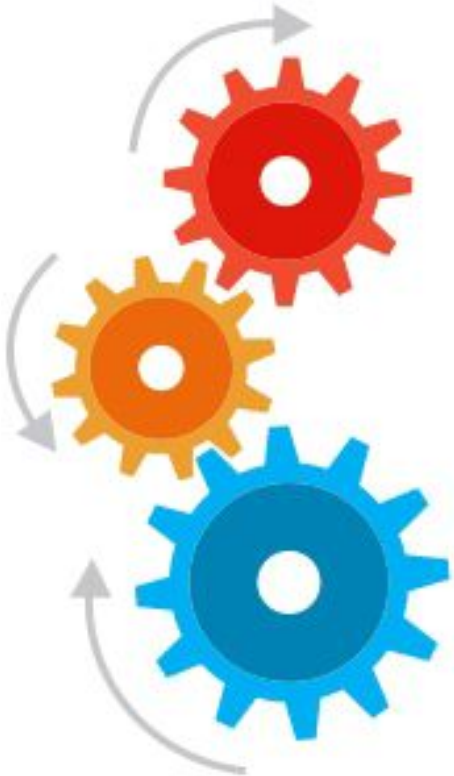
Un template es un fichero de texto que define la estructura o diagrama de otro fichero (tal como una página HTML), con marcadores de posición que se utilizan para representar el contenido real.

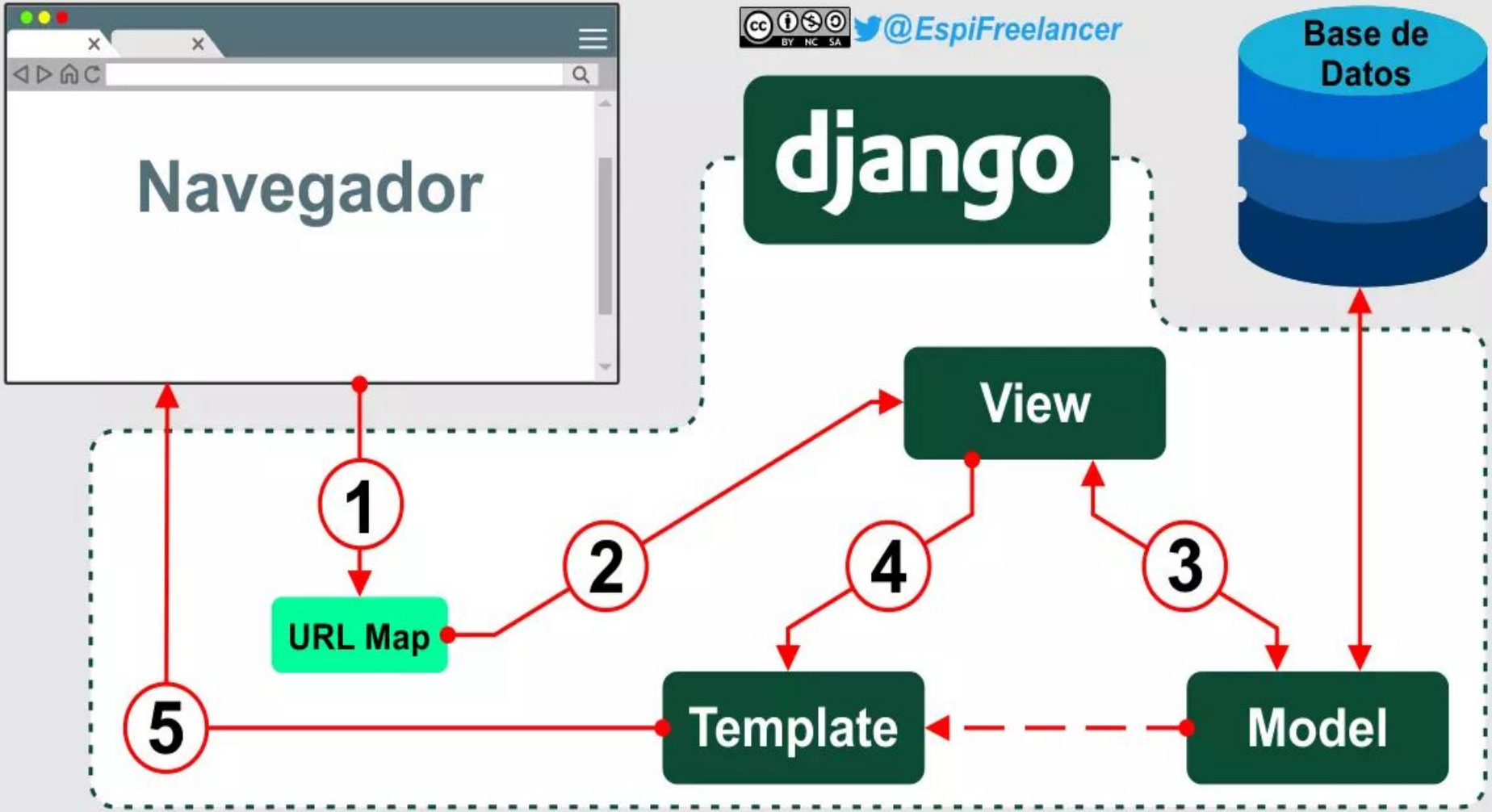
Una vista puede crear dinámicamente una página usando una plantilla, rellenandola con datos de un modelo.

Una plantilla se puede usar para definir la estructura de cualquier tipo de fichero.

View

Una vista es una función de gestión de peticiones que recibe peticiones HTTP y devuelve respuestas HTTP. Las vistas acceden a los datos que necesitan para satisfacer las peticiones por medio de modelos, y delegan el formateo de la respuesta a las plantillas ("templates").




Navegador
@EspiFreelancer
Base de Datos
django
View
1
2
3
4
5
URL Map
Template
Model

Resumen

- **Model**

Maneja todo lo relacionado con la información, esto incluye cómo acceder a esta, la validación, relación entre los datos y su comportamiento. (Cuenta con su propio ORM)

- **View**

Es un enlace entre el modelo y el template. Decide qué información será mostrada y por cual template.

- **Template**

Decide cómo será mostrada la información.

View

```
def index(request):
    latest_question_list = Question.objects.order_by('-pub_date')[:]
    context = {'latest_question_list': latest_question_list}
    return render(request, 'polls/index.html', context)
```

Model

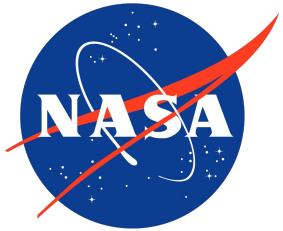
```python
class Question(models.Model):
    question_text = models.CharField(max_length=200, verbose_name='Pregunta')
    pub_date = models.DateTimeField(verbose_name='Fecha publicación', blank=True, null=True)

    def __str__(self):
        return self.question_text
```

Template

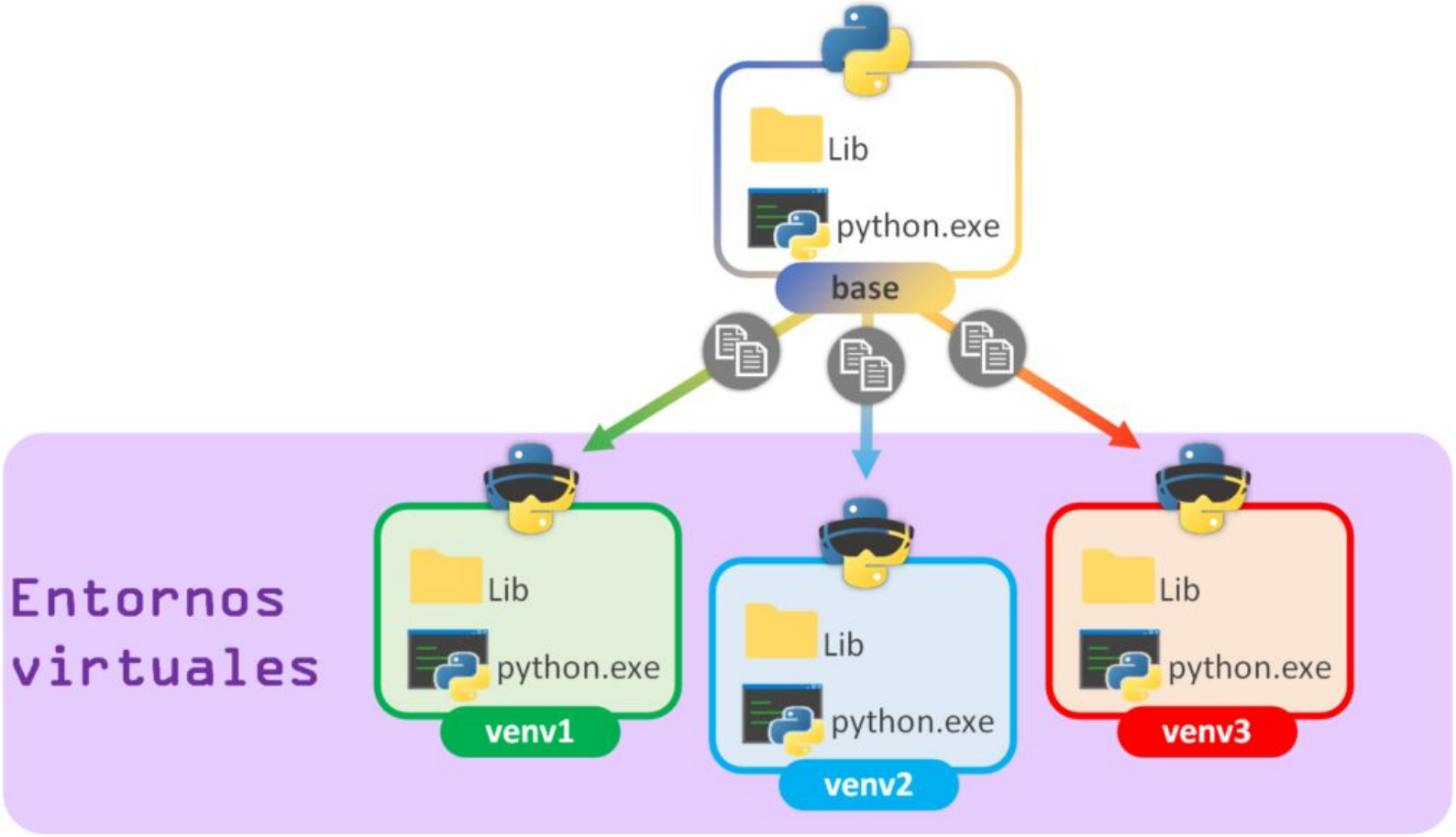
```
<body>
  <h1>Primer Proyecto Django</h1>
  <form id="form">
    {% if latest_question_list %}
    <div class="form-control">
      <h2>Selecciona la pregunta que deseas constestar:</h2>
      {% for question in latest_question_list %}
        <label><p><a href="{% url 'polls:detail' question.id %}">{{ question.question_text }}</a></p></label>
      {% endfor %}
    </div>
    {% else %}
        <p>No hay encuestas disponibles.</p>
    {% endif %}
  </form>
</body>
</html>
```

Empresas que lo utilizan actualmente



Entornos virtuales

Un entorno virtual es un entorno de Python parcialmente aislado que permite instalar paquetes para que los use una aplicación en particular en lugar de instalarlos en todo el sistema ya que hay aplicaciones que trabajan con librerías o módulos específicos que en ocasiones están descontinuados o se vuelven obsoletos.


Lib
python.exe
base
Entornos
virtuales
Lib
python.exe
venv1
Lib
python.exe
venv2
Lib
python.exe
venv3

Pasos para trabajar con un entorno virtual

1) Verificar que se tiene instalado Python en el ordenador con el siguiente comando desde la terminal:

```
python --version
```

2) Instalar el paquete de virtualenv con el siguiente comando:

```
pip install virtualenv
```

3) Crear el entorno con:

```
python -m venv <nombre_entorno>
```

Pasos para trabajar con un entorno virtual

4) Activar el entorno virtual:

```
nombre_entorno/scripts/activate
```

5) Instalar Django y paquetes necesarios con el siguiente comando:

```
pip install <nombre_paquete>
```

Comandos indispensables con Django

- Crear proyecto

```
django-admin startproject <nombre_proyecto>
```

- Crear aplicaciones

```
python manage.py startapp <nombre_aplicacion>
```

- Crear usuarios para el panel de administración

```
python manage.py createsuperuser
```

Comandos indispensables con Django

- Migrar modelos a una base de datos.

```
python manage.py migrate
```

- Actualizar los cambios de los modelos en la base de datos.

```
python manage.py makemigrations
```

- Actualizar los cambios de los modelos en la base de datos.

```
python manage.py runserver
```

Fuentes recomendadas

- **Django Documentation**

https://docs.djangoproject.com/en/4.0/

- **Youtube**

- Introducción a Django | Curso Completo (Solo Python)

https://www.youtube.com/watch?v=mQ0IUy_18mI

- Curso Django 2 (Developer. pe)

https://www.youtube.com/playlist?list=PLMbRqrU_kvbTGg_oUKXyWi63Mo9Yoot9K