# Gpul:
# Django Framework + Raspberries


UXÍO FUENTEFRÍA ORÓNS

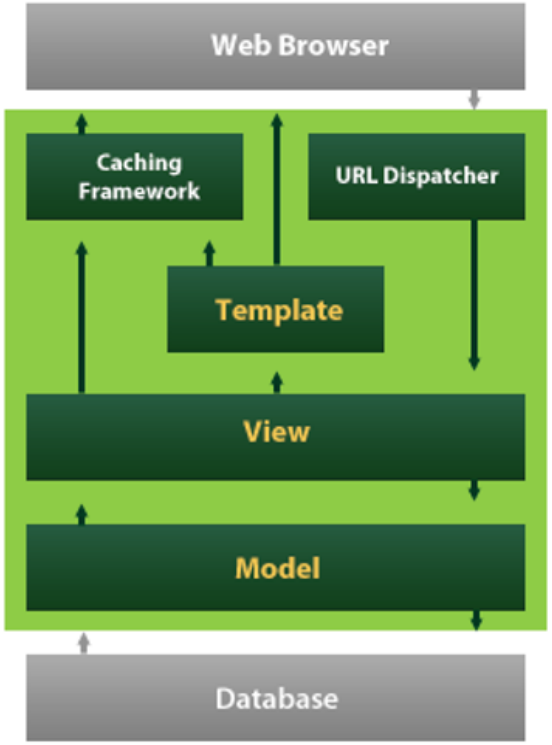
# Django: La D es muda

# Django: Introducción

- Framework que ha traído el desarrollo web a *Python*, rivaliza a menudo con *Ruby on Rails*.
- Desarrollo rápido, limpio y con la filosofía ***DRY*** (Don´t repeat yourself).
- Incluye un ***ORM*** de serie y puede ser usado con las bases de datos más comunes (***MariaDB***, ***Sqlite***, ***PostgreSQL***, ***Oracle***…).
- Incluye directamente mecanismos que nos protegen de los ataques más importantes a nivel web (inyección de ***SQL***, ***CSRF***…).
- Escalable.
- Extensible mediante *plugins*.
- Sigue un sistema parecido a Modelo-Vista-Controlador, en el que tenemos Vistas, Plantillas y Modelos.

# Django: Introducción

- “Son los usuarios los que eligen Django, y es Django el que quiere que sean los usuarios Django”



# Django: Capa modelo

- Corresponde a los datos gestionados por la aplicación.
- Los modelos se definen como clases, y se generarán automáticamente métodos para modificar datos y hacer consultas sobre ellos.
- Se conecta a bases de datos. La recomendada es ***PostgreSQL***, aunque es común usar ***SQLite*** durante el desarrollo y luego pasar a ***PostgreSQL*** para producción. No requiere más cambio en el código que cambiar la configuración de la base de datos.
- Django implementa un ***ORM*** que hace muy sencillo trabajar con los datos de la aplicación.
- https://docs.djangoproject.com/en/1.9/topics/db/models/

# Django: Capa modelo

```
class Band(models.Model):
    name = models.CharField(max_length=200)
    can_rock = models.BooleanField(default=True)
class Member(models.Model):
    name = models.CharField("Member's name", max_length=200)
    instrument = models.CharField(choices=(
            ('g', "Guitar"),
            ('b', "Bass"),
            ('d', "Drums"),
        ),
        max_length=1
    )
    band = models.ForeignKey("Band")
```

# Django: Capa modelo. Así de fácil

# Django: Capa vista

- Prepara los datos (usando los modelos o no) que serán renderizados en las plantillas para mostrar al usuario.
- Las vistas pueden ser diseñadas usando funciones o clases. Se recomienda el uso de clases, aunque a veces puede ser mejor emplear funciones.
- https://docs.djangoproject.com/en/1.9/topics/http/views/
- https://docs.djangoproject.com/en/1.9/topics/class-based-views/

# Django: Capa vista (funciones)

```
from django.shortcuts import render

import .models



def band_listing(request):
    """A view of all bands."""
    bands = models.Band.objects.all()
    return render(request, 'bands/band_listing.html', {'bands': bands})
```

# Django: Capa vista (clases)

```
from django.shortcuts import render

from django.views.generic import View

import .models


class BandListing(View):
    """A view of all bands."""
    def get(self, request, *args, **kwargs):
            bands = models.Band.objects.all()
            return render(request, 'bands/band_listing.html', {'bands': bands})
```

# Django: Capa vista (clases) (mejorado)

```
from django.views.generic.base import TemplateView
import .models


class BandListing(TemplateView):
    template_name = "bands/band_listing.html"


    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['bands'] = models.Band.objects.all()
        return context
```

# Django: Capa vista (clases) (mejorado)

# Django: Capa vista (clases) (aún mejor)

```
from django.views.generic import ListView

import .models


class BandListing(ListView):

    template_name = "bands/band_listing.html"

    model = models.Band
```

# Django: Capa vista (clases) (aún mejor)

# Django: Urls

- Permite definir qué patrones en la Url serán atendidos por unas vistas o otras.

```
from django.conf.urls import url


from . import views


urlpatterns = [
    url(r'^bands/$', views.band_listing, name='band-list'),
    url(r'^bands/(\d+)/$', views.band_detail, name='band-detail'),
    url(r'^bands/search/$', views.band_search, name='band-search'),
]
```

# Django: Plantillas

- Las plantillas de Django sirven para componer *HTML5* a partir de los datos preparados por las vistas.
- Son fáciles de usar para cualquiera que ya sepa *HTML*.
- Al tener separada la lógica (vistas) de lo que mostramos (plantillas) es mucho más fácil trabajar con diseñadores y desarrolladores de *front-end*.
- Las plantillas pueden extenderse y seguir un sistema de herencia.
- Soporta por defecto ***DTL*** (*Django Template Language*) y ***Jinja2***. También hay soportes para otras librerías usando el *backend* de terceros.
  - El soporte para múltiples sistemas de plantillas está disponible a partir de **Django 1.8**.
  - https://docs.djangoproject.com/en/1.9/topics/templates/#configuration

# Django: Plantillas

```
<html>
 <head>
  <title>Band Listing</title>
 </head>
 <body>
  <h1>All Bands</h1>
  <ul>
  {% for band in bands %}
   <li>
    <h2><a href="{{ band.get_absolute_url }}">{{ band.name }}</a></h2>
    {% if band.can_rock %}<p>This band can rock!</p>{% endif %}
   </li>
  {% endfor %}
  </ul>
 </body>
</html>
```

# Django: Creando nuestra primera web

- `sudo pip3 install django`
- `python3 -c "import django; print(django.get_version())"`
- `django-admin startproject raspberry`
- raspberry/
  - manage.py
  - raspberry/
    - __init__.py
    - settings.py
    - urls.py
    - wsgi.py
- `python3 manage.py runserver  # Arranca el servidor para hacer las pruebas`

# Django: Creando modelos

- Primero creamos la aplicación ( proyecto -> aplicaciones ) que contendrá los modelos:
  - `python3 manage.py startapp core`
- core/
  - __init__.py
  - admin.py
  - migrations/
    - __init__.py
    - models.py
    - tests.py
    - views.py
- Editamos *models.py*
- También tenemos que editar el archivo *settings.py* del proyecto para añadir *core* a las aplicaciones instaladas.

# Django: Creando modelos

```
from django.db import models


class Raspberry(models.Model):
    ip = models.CharField(max_length=20)
    name = models.CharField(max_length=30)
```

# Django: Creando modelos

- Le decimos al proyecto que hemos cambiado modelos.
  - python3 manage.py makemigrations
- Para depurar, podemos ver qué SQL ha generado:
  - python3 manage.py sqlmigrate core 0001
- Creamos las tablas en la base de datos:
  - python3 manage.py migrate
- Podemos ahora hacer pruebas en modo interactivo:
  - python3 manage.py shell

# Django: Creando modelos

- Las migraciones se detectan y realizan automáticamente!

# Django: Probando modelos

```
In [1]: from core.models import Raspberry

In [2]: Raspberry.objects.all()

Out[2]: []

In [3]: m = Raspberry(name='Una', ip='localhost')

In [4]: m.save()

In [5]: Raspberry.objects.all()

Out[5]: [<Raspberry: Raspberry object>]

In [6]: m.ip

Out[6]: 'localhost'

In [7]: m.name

Out[7]: 'Una'
```

# Django: Vistas

- Editamos el archivo *views.py* dentro del directorio core.

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("Hello, world. You're at the Raspberry index.")

def detail(request, raspberry_id):
    return HttpResponse("You're looking at Raspberry {}.".format(raspberry_id))
```

# Django: Urls

- Dentro de la carpeta *core/* creamos un fichero *urls.py*, y añadimos lo siguiente:

```
from django.conf.urls import url
from . import views

urlpatterns = [
    # ex: /core/
    url(r'^$', views.index, name='index'),
    # ex: /core/
    url(r'^(?P<raspberry_id>[0-9]+)/$', views.detail, name='detail'),
]
```

- Ayuda sobre expresiones regulares: https://docs.python.org/3/library/re.html#module-re

# Django: Urls

- Añadimos las urls de la aplicación core a *raspberry/urls.py*

```
from django.conf.urls import include, url
from django.contrib import admin
urlpatterns = [
    url(r'^core/', include('core.urls')),
    url(r'^admin/', include(admin.site.urls))]
```

# Django: Vistas con modelos

- Comprobamos que funciona, y ahora vamos a recuperar datos de verdad y usar una plantilla:

```
from django.shortcuts import render
from .models import Raspberry


def index(request):
    raspberry_list = Raspberry.objects.order_by('name')
    context = {'raspberry_list': raspberry_list}
    return render(request, 'index.html', context)
```

# Django: Vistas con modelos

- Comprobamos que funciona, y ahora vamos a recuperar datos de verdad y usar una plantilla:

```
from django.views.generic import ListView

from .models import Raspberry


class RaspberryList(ListView):

    template_name = 'core/index.html'

    model = Raspberry
```

# Django: Plantillas

- Dentro de la carpeta *core/templates/ creamos un fichero index.html:*

```
{% if raspberry_list %}
    <ul>
    {% for raspberry in raspberry_list %}
        <li><a href="/core/{{ Raspberry.id }}/">{{ raspberry.ip}} – {{ raspberry.name }}</a></li>
    {% endfor %}
    </ul>
{% else %}
    <p>No raspberries.</p>
{% endif %}
```

# Django: Plantillas con url dinámica

- Dentro de la carpeta *core/templates/ creamos un fichero index.html:*

```
{% if raspberry_list %}
    <ul>
    {% for raspberry in raspberry_list %}
        <li><a href="/core/{% url 'core:detail' Raspberry.id %}/">
            {{ raspberry.ip}} – {{ raspberry.name }}</a>
        </li>
    {% endfor %}
    </ul>
{% else %}
    <p>No raspberries.</p>
{% endif %}
```

# Django: Vistas lanzando excepciones

```
from django.http import Http404
from django.shortcuts import render
from .models import Raspberry


def detail(request, raspberry_id):
    try:
        raspberry = Raspberry.objects.get(pk=raspberry_id)
    except Raspberry.DoesNotExist:
        raise Http404("Raspberry does not exist")
    return render(request, 'details.html', {‘raspberry': raspberry})
```

# Django: Vistas lanzando excepciones (Mejorado)

```
from django.shortcuts import get_object_or_404, render

from .models import Raspberry


def detail(request, raspberry_id):
    raspberry = get_object_or_404(raspberry, pk=raspberry_id)
    return render(request, 'details.html', {‘raspberry': raspberry})
```

# Django: Plantillas

- Dentro de la carpeta *core/templates/ creamos un fichero details.html:*

```
<h1>{{ raspberry.name}}</h1>
<ul>
{% for report in raspberry.report_set.all %}
    <li>{{ report.ip }} – {{report.created_at}}</li>
{% endfor %}
</ul>
```

# Django: Extender plantillas

```
<!DOCTYPE html>
<html lang="en">
<head><link rel="stylesheet" href="style.css" />
<title>{% block title %}My amazing site{% endblock %}</title></head>
<body><div id="sidebar">{% block sidebar %}
        <ul><li><a href="/">Home</a></li><li><a href="/blog/">Blog</a></li>
        </ul>{% endblock %}
    </div>
    <div id="content">
        {% block content %}{% endblock %}
    </div></body></html>
```

# Django: Extender plantillas

```
{% extends "base.html" %}

{% block title %}My amazing blog{% endblock %}

{% block content %}
{% for entry in blog_entries %}
    <h2>{{ entry.title }}</h2>
    <p>{{ entry.body }}</p>
{% endfor %}
{% endblock %}
```

# Django: Cargar ficheros estáticos en plantillas

```
{% load staticfiles %}

<img src="{% static "my_app/myexample.jpg" %}" alt="My image"/>
```

# Django: Tareas de línea de comandos

- core/
- __init__.py
- models.py
- management/
- __init__.py
- commands/
- __init__.py
- _private.py
- check_reports.py
- tests.py
- views.py

# Django: check_reports.py

```
from django.core.management.base import BaseCommand, CommandError
from core.models import Raspberry


class Command(BaseCommand):
    help = 'Show Raspberrys'


    def handle(self, *args, **options):
        for Raspberry in Raspberry.objects.all():
            self.stdout.write('Raspberry {} - {}'.format(Raspberry.name, Raspberry.ip))
```

# Django: Probando interfaz admin

- Crear superusuario
  - python3 manage.py createsuperuser
  - Iniciar el servidor e ir a http://127.0.0.1:8000/admin/

# Django: Probando interfaz admin

- Registrar nuestros modelos
  - Editamos *core/admin.py*

```
from django.contrib import admin

from .models import Raspberry, Report


admin.site.register(Raspberry)

admin.site.register(Report)
```

# Django: Ejercicio

# Django-redis: Caché

- pip install django-redis

```
CACHES = {
    "default": {
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1:6379/1",
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
        }
    }
}
SESSION_ENGINE = "django.contrib.sessions.backends.cache"
SESSION_CACHE_ALIAS = "default"
```

# Django: Caché

```
>>> from django.core.cache import cache
>>> cache.set("foo", "value", timeout=25)
>>> cache.ttl("foo")
25
>>> cache.ttl("not-existent")
0
>>> cache.set("foo", "bar", timeout=22)
>>> cache.ttl("foo")
22
>>> cache.persist("foo")
>>> cache.ttl("foo")
None
```

# Django: Bloqueos usando caché

```
with cache.lock("somekey"):
    do_some_thing()
```

# Django: Caché en bloques

```
>>> from django.core.cache import cache
>>> cache.keys("foo_*")
["foo_1", "foo_2"]
>>> from django.core.cache import cache
>>> cache.iter_keys("foo_*")
<generator object algo at 0x7ffa9c2713a8>
>>> next(cache.iter_keys("foo_*"))
"foo_1"
```

# Django: Ruegos y preguntas