Django

Sergio Paniego Blanco
@sergiopaniego
sergiopaniegoblanco@gmail.com
https://sergiopaniego.github.io/

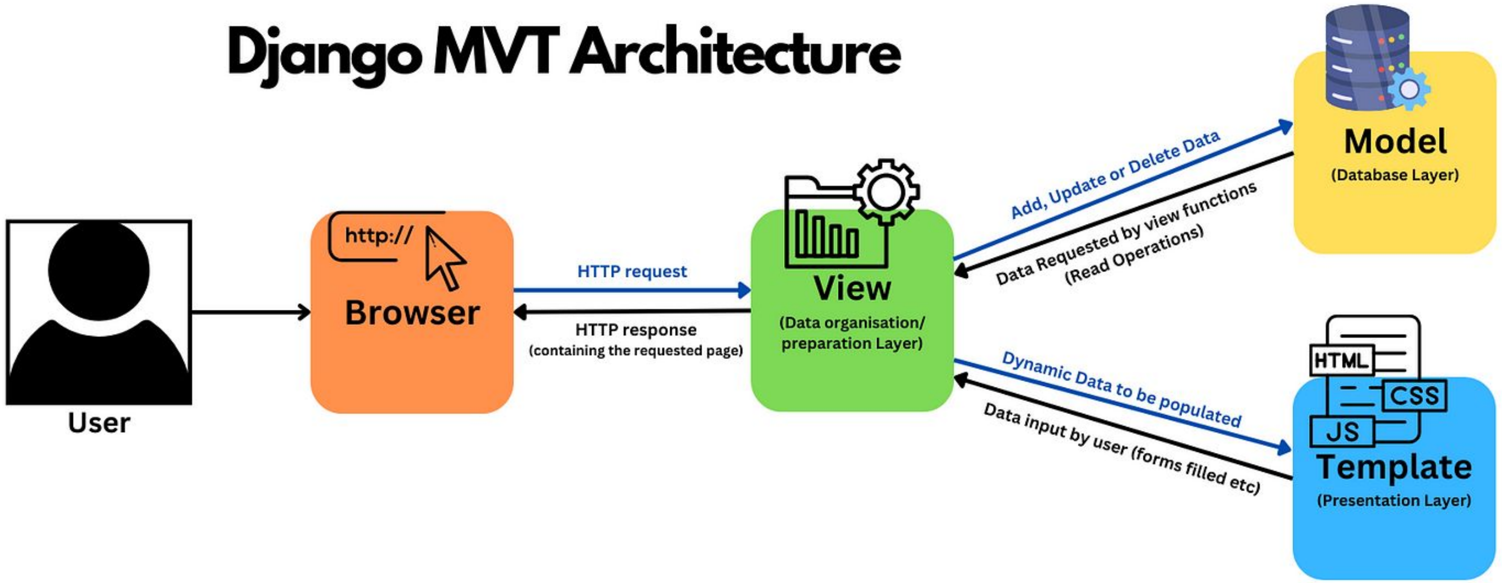
Django

- Introduction to Django
- Models and Databases
- Views and URLs
- Templates and static files
- Authentication and administration.
- APIs and REST framework.

Introduction to Django

- Django is a high-level backend Python framework focused on developing fast and clean web applications.
- Advantages: scalability, security, easy development, active community.
- Use cases: commercial applications, APIs, internal systems…

Introduction to Django


Django MVT Architecture
User
http://
Browser
HTTP request
HTTP response
(containing the requested page)
View
(Data organisation/ preparation Layer)
Add, Update or Delete Data
Data Requested by view functions (Read Operations)
Model
(Database Layer)
Dynamic Data to be populated
Data input by user (forms filled etc)
HTML
CSS
JS
Template
(Presentation Layer)

Installation and configuration of a new project

- Setup and requirements

```
pip install django
```

- Start a new Django project

```
django-admin startproject my_project
```

- Create a new `Django` app and check that it works!

```
cd my_project
python manage.py runserver
```

django

Introduction to Django

- Django project structure:
  - Project folder
    - `settings.py`: config file (databases, installed apps, security configurations…)
    - `urls.py`: app routes configuration file.
    - `asgi.py` y `wsgi.py`: web server communication configuration.
  - `manage.py`: main command for project management (migrations, start server, create applications…)

Introduction to Django

- Let’s change the language. Check the changes in localhost

```
LANGUAGE_CODE = ‘es’
```

- Let’s change the time zone.

```
TIME_ZONE = 'Europe/Madrid'
```

- Let’s check the changes

```
python manage.py shell
```

- Inside the shell

```
from django.utils import timezone
print("Current time zone:", timezone.get_current_timezone_name())
current_time = timezone.now()
print("Current time with timezone:", current_time)
```

Introduction to Django

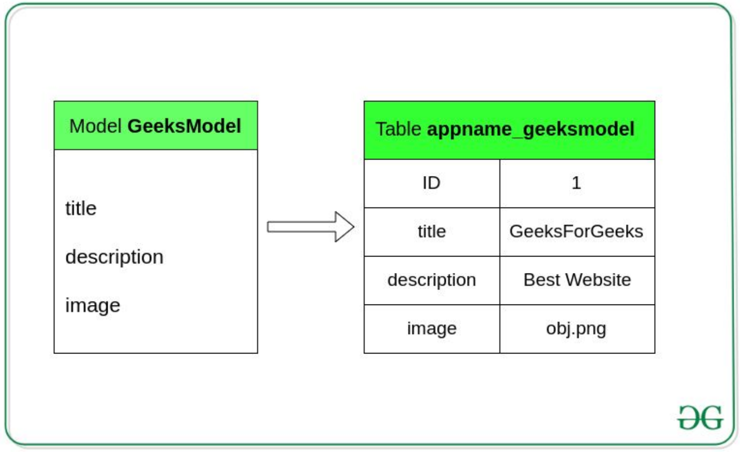
- There is a default database already configured.

Introduction to Django

- Let's investigate the structure!

Models and databases

- Models are a fundamental part of the design pattern model-view-controller (MVC) or as it is known in Django model-view-template (MVT).
- Model: representation of data in the database and responsible for defining the structure and relationship between data.



ID	1
title	GeeksForGeeks
description	Best Website
image	obj.png

Models and databases

- Common data types: CharFields, IntegerField, DateField, EmailField…

```
from django.db import models

class Author(models.Model):
    name = models.CharField(max_length=100)
    email = models.EmailField()
    date_of_birth = models.DateField()

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

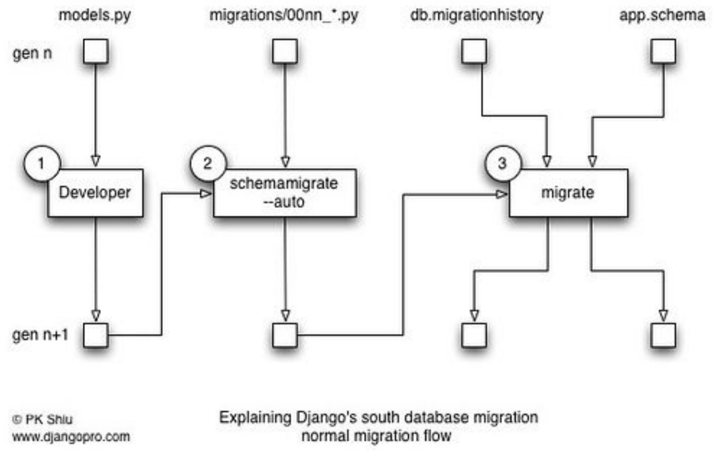
Models and databases

- ORM (Object-Relational Mapping): technique to interact with relational databases using a object oriented programming instead of using SQL queries.
- Features
  - Abstraction from the database.
  - Models as Classes.
  - Simplified queries.
  - Automatic migrations.
  - Relationships between models are easier to define and manage.
  - Compatibility between databases.
  - Queries execution: the ORM generates the SQL queries from the operations made with the objects.

Models and databases

- Migrations: when changing a model, Django can automatically generate the migrations for creating or modifying the tables in the database.

```
python manage.py makemigrations
python manage.py migrate
```



Models and databases

- Adding some objects to the database:

```
python manage.py shell
```

- Executing:

```
from my_app.models import Author

author = Author(name='Jane Doe', email='jane@example.com',
date_of_birth='1980-01-01')
author.save()

authors = Author.objects.all()

author.email = 'jane.doe@example.com'
author.save()

author.delete()
```

ACTIVITY

- Add a new model, migrate the database and add/delete some objects.

Routes and views configuration

- urls.py file is responsible for managing routing.
- They match routes with specific views.
- Django divides URLs in two levels:
  - `urls.py` at project level: configures main routes and connects to urls.py of each application.
  - `urls.py` at application level: defines routes for each view inside an specific application.
- Let's see urls.py

Routes and views configuration

- Create a new `Django` app

```
cd library_project
python manage.py startapp my_app
```

- Configure `my_project/urls.py`

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('my_app.urls')),  # Links to URLs in 'my_app'
]
```

Routes and views configuration

- Configure `my_app/urls.py`

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.home, name='home'),
    path('about/', views.about, name='about'),
]
```

Routes and views configuration

- Configure `my_project/settings.py`

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'my_app',  # new app
]
```

- We have mapped the URLs to the views.
- We still can’t see anything. We need to configure the views!

Routes and views configuration

- In Django, views are the components responsible for receiving HTTP requests and return responses.
- Two possibilities:
  - Function-based views: python functions that receive a request object and return an `HttpResponse`/`JsonResponse`.

```
def home(request):
    return HttpResponse("Bienvenido a la página de inicio")

def api_data(request):
    data = { "name": "Sergio", "age": 30, "city": "Madrid" }
    return JsonResponse(data)
```

Routes and views configuration

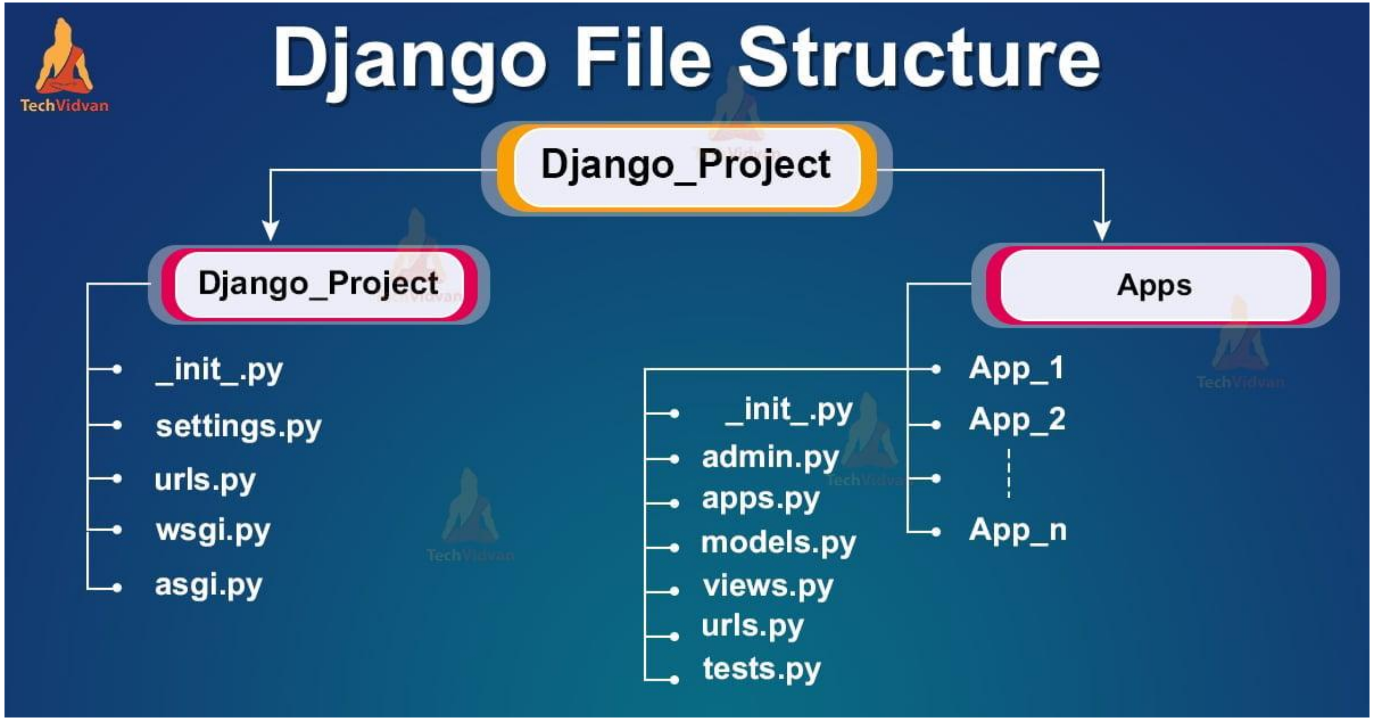
- Two possibilities:
  - Class-based views: python classes used to build more structured views. Django provides predefined views for common tasks (`ListView` for lists of elements, `DetailView` for details of an object or `TemplateView` for using templates.)

```
# views.py
class AboutPageView(TemplateView):
    template_name = "about.html"

#urls.py
from .views import AboutPageView

urlpatterns = [
    path('about/', AboutPageView.as_view(), name='about'),
]
```

Let’s configure the views

```
from django.http import HttpResponse
from django.shortcuts import render

def home(request):
    return HttpResponse("Welcome to the Home Page")
def about(request):
    return render(request, 'my_app/about.html')
```

Routes and views configuration



ACTIVITY

- Add a new url and view to the application.
  - Play with `JsonResponse`, `HttpResponse...`
- Check that you can access the new view!

Routes and views configuration

- For the render option, we need to
  - Generate a folder for the templates: `my_app/templates/my_app/about.html`
  - Update TEMPLATES in my_project/settings.py
  - Add the html

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Routes and views configuration

```html
<!-- mi_app/templates/mi_app/about.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>About Us</title>
</head>
<body>
    <h1>About Us</h1>
    <p>Welcome to the "About" page. Here you can learn more
about us.</p>
</body>
</html>
```

ACTIVITY

- Add a new url and view to the application that returns a rendered HTML.

Models and databases

- Let’s add a form to update/see the data

```
# views.py
from django.shortcuts import render, redirect
from .models import Author
from .forms import AuthorForm
def author_view(request):
    if request.method == 'POST':
        form = AuthorForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('author_view')
        else:
            form = AuthorForm()
    authors = Author.objects.all()
    return render(request, 'my_app/authors.html', {'form': form,
    'authors': authors})
```

Models and databases

- Let’s add a HTML form to update/see the data

```
# forms.py
from django import forms
from .models import Author

class AuthorForm(forms.ModelForm):
    class Meta:
        model = Author
        fields = ['name', 'email', 'date_of_birth']
```

```
# urls.py
urlpatterns = [
    …
    path('authors/', views.author_view, name='author_view')
]
```

Models and databases

```
# authors.html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Authors</title>
</head>
<body>
    <h1>Authors</h1>
    <h2>Add New Author</h2>
    <form method="post">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit">Add Author</button>
    </form>
    <h2>Author List</h2>
    <ul>
        {% for author in authors %}
            <li>{{ author.name }} - {{ author.email }} - {{ author.date_of_birth }}</li>
        {% empty %}
            <li>No authors found.</li>
        {% endfor %}
    </ul>
</body>
</html>
```

ACTIVITY

- Add a new form and view to manage the data directly from the web.

Templates and static files

- Templates allow to generate dynamic HTML content (Django Template Language).
- With them, we can:
  - Insert data
  - Insert control structures (conditionals, loops)
  - Apply inheritance to generate a base template for extension.
- `/templates` folder

Templates and static files

- Basic syntax:
  - `{{variable}}`
  - `{% for %} … {% endfor %}`
  - `{% if %} … {% endif %}`
  - `{% static ‘…‘ %}`
- Let’s see an example `profile.html` and `views/profile`

Templates and static files

- Inheritance in templates:
    - Base template
    - Extended template.
- Let’s check base.html and `about_2.html`.
- Let’s add some style using CSS (`style.css`)
- Check that in `settings.py STATIC_URL = '/static/'`
- We can also add images (`base_with_logo.html`) os JS (`base_with_js.html`)

ACTIVITY

- Add a new view combining
  - Django Template Language
  - Static content

Authentication and administration

- User authentication
  - User Registration: we can create a custom view for managing this pipeline
    - register in `views.py`
    - `UserCreationForm`
    - `register.html`
  - Login and Logout: Django includes built-in views for this tasks
    - `login/` and `registration/login.html`
    - `logout/`

Authentication and administration

- Protecting views with authentication decorators:
- Django provides decorators for different functionalities. For example:
  - `@login_required`: view only visible for logged users

Authentication and administration

- Django administration: powerful tool for managing the content of the application. To access, create a superuser:

```
python manage.py createsuperuser
```

- Register the models in admin.py with a customized view

ACTIVITY

- Register and tests user.
- Customize administration with different `list_display`, `search_fields`, `list_filter`.
- Create a user form for updating user’s basic information.

API and REST framework

- We’ve already seen this part but let’s recap!

```
pip install djangorestframework
```

- Add to installed apps

```
INSTALLED_APPS = [
    …
    ‘rest_framework’,
]
```

API and REST framework

- Django can manage the API endpoints.
- We can create using `views.py`, `serializers.py` and `urls.py`

```
# views.py
from rest_framework import viewsets
from .models import Book
from .serializers import BookSerializer
class BookViewSet(viewsets.ModelViewSet):
      queryset = Book.objects.all()
      serializer_class = BookSerializer
```

```
# serializers.py
from rest_framework import serializers
from .models import Book
class BookSerializer(serializers.ModelSerializer):
      class Meta:
            model = Book
            fields = '__all__'
```

```
# urls.py
from django.urls import path, include
from rest_framework.routers import DefaultRouter
from .views import BookViewSet
router = DefaultRouter()
router.register(r'books', BookViewSet)
urlpatterns = [ path('', include(router.urls)), ]
```

ACTIVITY

- Add a new API call for authors