syncro soft

Open4Tech Summer School 2025

Web Dev Made Easy: Django, MVT vs MVC, REST, Bootstrap & More

Vlad Greaca
Software Developer, Syncro Soft

Alexandru Smarandache
Software Developer, Syncro Soft

Top 8 Frameworks for Web Applications in 2025

- 1. Ruby on Rails
- 2. Django
- 3. Angular
- 4. ASP.NET
- 5. METEOR
- 6. Laravel
- 7. Express
- 8. Spring

Why Django?

- One of its defining philosophies is that it's "**batteries-included**," meaning that Django comes with almost everything a developer needs out of the box. This reduces the need for external libraries and simplifies the development lifecycle.
- Ruby on Rails was also developed under this philosophy, however **Python** is the more popular language with a lot more packages available, using **pip**.



Some of the built-in features include:

- ORM (Object-Relational Mapping)
- **Authentication System**
- **Admin Interface**
- URL Routing
- Templating Engine
- Forms and Validation
- Internationalization (i18n)
- **Security Features (CSRF, XSS, SQL injection protection)**
- Caching
- Middleware
- Testing Framework
- Sitemap Framework
- Session Management
- File Upload Handling

Let's Get Started!

- To use the framework you need to install it, using:
  - *pip install Django*
- To start a new project use:
  - *django-admin startproject <name for properties> <name of the project>*
- To add the primary structure call:
  - *python manage.py startapp <name of the site>*
- To start the newly created app just call:
  - *python manage.py runserver*

Files Created

- These files are:
- **manage.py**: A command-line utility that lets you interact with this Django project in various ways.
- **settings.py**: Settings/configuration for this Django project.
- **urls.py**: The URL declarations for this Django project.

Files Created

- *asgi.py*: An entry-point for ASGI(Asynchronous Server Gateway Interface)-compatible web servers to serve your project.
  Is a spiritual successor to WSGI, intended to provide a standard interface between async-capable Python web servers
- *wsgi.py*: An entry-point for WSGI(Web Server Gateway Interface)-compatible web servers to serve your project. See How to deploy with WSGI for more details.
  It is a specification that describes how a web server communicates with web applications, and how web applications can be chained together to process one request.

```
69    WSGI_APPLICATION = 'settings.wsgi.application'
```

MVC VS MVT

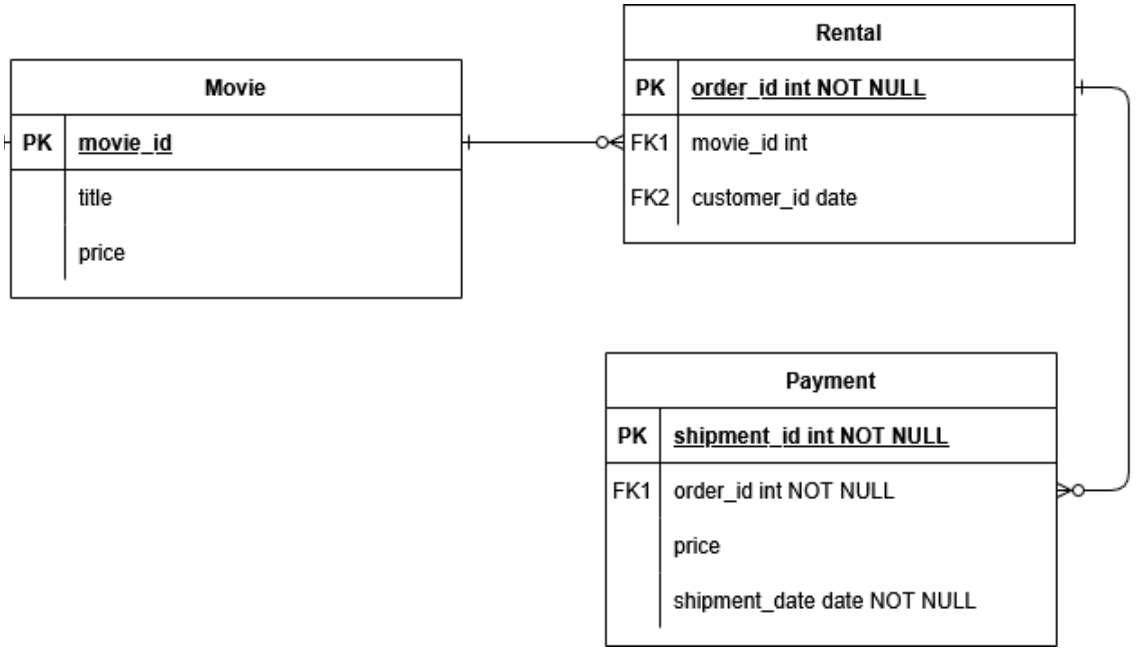
- Model: Represents the data or business logic of the application. It manages the data and logic, and updates the View when the data changes.
- View: Represents the user interface (UI) elements, typically in the form of HTML.
- Controller: Acts as the intermediary between the Model and the View. It handles user input, updates the Model and the View.

- Model: Similar to MVC
- View: Is responsible for receiving user input, processing it, and returning an appropriate response.
- Template: Is responsible for rendering the HTML or UI elements. It's a file that defines the structure and layout of the output.

Files Created

- **models.py**: Defines the data models (i.e., database schema) for the app. Each model maps to a single database table.
- **admin.py**: This file is used to register your models with the Django admin site. Once registered, your models can be managed via Django's built-in admin interface.
- **views.py**: Contains the view functions or classes that control the logic for handling web requests and returning responses (e.g., HTML pages).
- **urls.py**: Defines the URL patterns for the polls app. This connects views to specific URLs, often included in the project's main.
- *apps.py*: Contains the configuration for the web app. This includes metadata like the app name and can be customized for advanced app configurations.

Entity-Relation Diagram



Models

- A model is the single, definitive source of information about your data. It contains the essential fields and behaviors of the data you're storing. Generally, each model maps to a single database table.
- The basics:
  - Each model is a Python class that subclasses django.db.models.Model.
  - Each attribute of the model represents a database field.
- https://docs.djangoproject.com/en/5.2/topics/db/models/

Models Fields

- A few field Types and arguments:
  - CharField : maxLength = number
  - DateField/DateTimeField : auto_now=True/False
  - DecimalField : max_digits=number, decimal_places=number
  - BooleanField
  - AutoField/BigAutoField.
  - ForeignKey/OneToOneField/ManyToManyField = class, on_delete=models.CASCADE/SET_NULL/SET_DEFAULT
- https://docs.djangoproject.com/en/5.2/ref/models/fields/

Models Options

- **Meta -** Model metadata is “anything that’s not a field”, such as ordering options (ordering), database table name (db_table), or human-readable singular and plural names (verbose_name and verbose_name_plural).
- ex.

```
class Ox(models.Model):
    horn_length = models.IntegerField()

    class Meta:
        ordering = ["horn_length"]
        verbose_name_plural = "oxen"
```

Models Option:

- **__str__()** - A Python “magic method” that returns a string representation of any object. This is what Python and Django will use whenever a model instance needs to be coerced and displayed as a plain string. Most notably, this happens when you display an object in an interactive console or in the admin.
- You’ll always want to define this method; the default isn’t very helpful at all.

Database

- By default Django comes with SQLite, however it has support, by default, for PostgreSQL and MySQL. It can be modified in settings.py

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

- To migrate our models to the database we need to add our *apps* configuration in sources, as the following:
  - *<name_of_the_site>.apps.<name_of_the_class_created>*

Database

- Django can create migrations for you. Make changes to your models - say, add a field and remove a model - and then run
  - *python manage.py makemigrations*
- Once you have your new migration files, you should apply them to your database to make sure they work as expected - run:
  - *python manage.py migrate*

Admin

- Creating an admin user:
  - First we'll need to create a user who can login to the admin site. Run the following command:
    - *python manage.py createsuperuser*
  - The Django admin site is activated by default. Let's start the development server and explore it.
- To add the models to the admin site we need register the Model, in admins.py

```
3   from .models import Question
4
5   admin.site.register(Question)
```

Views

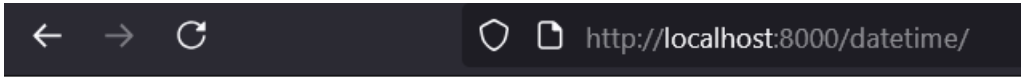
- A view function, or view for short, **is a Python function that takes a web request and returns a web response**. This response can be the HTML contents of a web page, or a *redirect*, or a 404 error, or an XML document, or an image . . . etc.
- ex.

```
from django.http import HttpResponse
from datetime import datetime


def current_datetime(request):
    now = datetime.now()
    html = f'<html lang="en"><body>It is {now}.</body></html>'
    return HttpResponse(html)
```

Mapping a URL

- To display this view at a particular URL, it needs to be added to urls.py

```python
from movie.views import *

urlpatterns = [
    path('admin/', admin.site.urls),
    path('datetime/', current_datetime, name='current_datetime'),
]
```



It is now 2025-07-01 10:46:30.059757.

Template

- A template contains the static parts of the desired HTML output as well as some special syntax describing how dynamic content will be inserted.
- To initialize a Template location add a URL, in settings.py/TEMPLATES/DIRS:
-

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': ['templates'],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Template

- In Django, we use render(request, template_name, context) in views.py to return an HTML template filled with data to the browser.
- *ex.*

index.html

example > templates > index.html

```
1   <head>
2       <meta charset="UTF-8">
3       <meta name="viewport" content="width=device-width, initial-scale=1.0">
4       <title>Movie</title>
5   </head>
6
7   <body>
8       <h1>Welcome to the Movie App</h1>
9       <p>Check the current date and time by visiting <a href="{% url 'current_datetime' %}">this link</a>.</p>
10  </body>
```

```
11  def index(request):
12      return render(request, 'index.html')
```

Templates Language Syntax

- Variables:
  - {{ <name_of_variable> }}
    ex.
    My first name is {{ name }}. My last name is {{ lastname }}.
  - In order to use a variable we must use a dictionary.
  - ex.

```
def index(request):
    name = "Greaca"
    lastname = "Vlad"
    context = {
        'name': name,
        'lastname': lastname
    }
    return render(request, 'index.html', context)
```

Templates Language Syntax

- {% if <condition> %}
  ...
  {% else %}
  ...
  {% endif %}
- {% for <item> in <list> %}
  {{ <item> }}
  {% endfor %}

Templates Static Files

- Websites generally need to serve additional files such as images, JavaScript, or CSS. In Django, we refer to these files as “static files”. Django provides django.contrib.staticfiles to help you manage them:
  - In settings.py/INSTALLED_APPS include django.contrib.staticfile is included in your .
  - Define STATIC_URL, for example:

```
130    STATIC_URL = "static/"
131    STATICFILES_DIRS = ["static"]
```

  - For the template that uses static files, include {% load static %}

Templates Static Files

- ex.

```
{% load static %}
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Movie</title>
</head>

<body>
    <h1>Welcome to the Movie App</h1>
    <p>Check the current date and time by visiting <a href="{% url 'current_datetime' %}">this link</a>.</p>

    <img src="{% static 'image.png' %}" alt="My image">
</body>
```

Queries

- Once you've created your data models, Django automatically gives you a database-abstraction API that lets you create, retrieve, update and delete objects.
- ex.
-

```
def query(request):
    entry = Entry.objects.get(pk=1)
    cheese_blog = Blog.objects.get(name="Cheddar Talk")
    entry.blog = cheese_blog
    entry.save()
```

Queries operations

- save()
- objects.all()
- objects.filter(<condition>)
  ex. Entry.objects.filter(blog_id=4)
- objects.get(<condition>)
  ex. Entry.objects.get(headline__exact="Cat bites dog")
  Entry.objects.get(headline__contains=”Cat”)

CSRF Token

- CSRF, or Cross-Site Request Forgery, is a type of web security vulnerability where an attacker tricks a user's browser into making unwanted actions on a website the user is currently logged into.
- The CSRF middleware is activated by default in DJango in the settings.py/MIDDLEWARE
- In any template that uses a POST form, use the csrf_token tag inside the <form> element if the form is for an internal URL, e.g.:
<form method="post">{% csrf_token %}

REST(Representational State Transfer)

- It is a type of API (Application Programming Interface) that allows communication between different systems over the internet. REST APIs work by sending requests and receiving responses, typically in JSON format, between the client and server.
- REST APIs use HTTP methods (such as GET, POST, PUT, DELETE) to define actions that can be performed on resources. These methods align with CRUD (Create, Read, Update, Delete) operations, which are used to manipulate resources over the web.

Serializer

- A serializer is a tool or process that converts data structures (like objects or complex data types) into a format suitable for storage or transmission, such as JSON or XML.
- A deserializer is a tool or process that converts serialized data (like from a file, network, or database) back into its original object or data structure in memory

Django REST Framework

- REST functionality is not added by default from Django, however there is a toolkit available thorough pip
  - pip install djangorestframework
- Then, it settings.py we need configure the REST API
- ex.

```
REST_FRAMEWORK = {
    'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.PageNumberPagination',
    'PAGE_SIZE': 10
}
```

Django REST Framework

- After, we need to serialize the information

ex.

```
21  from django.contrib.auth.models import User
22  from rest_framework import routers, serializers, viewsets
23
24  # Serializers define the API representation.
25  class UserSerializer(serializers.HyperlinkedModelSerializer):
26      class Meta:
27          model = User
28          fields = ['url', 'username', 'email', 'is_staff']
```

- Then, create a View

ex.

```
30  # ViewSets define the view behavior.
31  class UserViewSet(viewsets.ModelViewSet):
32      queryset = User.objects.all()
33      serializer_class = UserSerializer
```

Django REST Framework

- Some Web frameworks such as Rails provide functionality for automatically determining how the URLs for an application should be mapped to the logic that deals with handling incoming requests.
- REST framework adds support for automatic URL routing to Django, and provides you with a simple, quick and consistent way of wiring your view logic to a set of URLs.

```
# Routers provide an easy way of automatically determining the URL conf.
router = routers.DefaultRouter()
router.register(r'users', UserViewSet)

urlpatterns = [
    path('admin/', admin.site.urls),
    path('datetime/', current_datetime, name='current_datetime'),
    path('', index, name='index'),
    path('api-auth/', include('rest_framework.urls', namespace='rest_framework'))
]
```

RECAP

- Django as a a concept
- python manage.py runserver
- REST

Biblography:

- https://www.geeksforgeeks.org/blogs/top-frameworks-for-web-applications/
- https://medium.com/@naeem.ahmed.bdn/why-django-is-the-ultimate-batteries-included-framework-for-scalable-web-development-746ffcebaa7d
- https://docs.djangoproject.com/en/5.2/intro/tutorial01/
- https://www.django-rest-framework.org/