Fundamentals

Django

Free and open-source **framework** for building web apps with **python**.

- Batteries included
- admin site
- ORM
- Authentication
- Caching
- and much more

> "Performance isn't everything"

Average Django developer Salary: $117,673

How the web works

Every web application has two parts:

- Front-end (Client)
- Back-end (Server)

URL -> Uniform Resource Locator

a resource can be:

- Page
- Image
- Video
- PDF
- and more ...

Request / Response

1. Client writes a URL in the browser.
2. Hits Enter -> Request sent to the server
3. The web server processes the request and sends a response
4. client sees the page
   This process of data exchange is called Request / Response which is done with **HTTP**(Hyper Text Transfer Protocol).

How we respond to clients

- Generate and Return HTML files (Templated)
- Return only the data and have the client to generate the page (API)

What is the difference?

The backend server can focus on doing what it should do and not worry about generating front-end stuff like a html file with styling and ... .
As a result for this method our backend becomes more available and scalable.

> This approach has become the industry best practice.

Generating web pages on the Client

These days we have tools like:

- React
- Angular
- Vue
  for generating a web page on the client. These are client side tools that are used in the front-end.
  In Contrast, we have server-side tools for building backends. Django falls in this category.

API

If we push the responsibility of generating web pages to the client, the server essentially becomes a gateway to data. On the server we can provide **Endpoints** that the client can talk to, to get or save various pieces of data:

- /products
- /orders
- /customers
  All these endpoints together define the interface the client can use to talk to the server.

Creating the first Django Project

```
mkdir ProjectName
cd ProjectName
pipenv install django
```

Activate virtual Environment so you use the interpreter inside it, not the global one

```
pipenv shell
```

Start a new Django project

```
django-admin startproject projectname
```

This command does two things:

- Creates a base directory with the specified name
- Creates a subdirectory in the project directory (*Core of our application*)

```
project
|___project
```

```
                |__init__.py
                |asgi.py
                |settings.py
                |urls.py
                |wsgi.py
```

> 3 Redundant directory names
>
> use a "." at the end of `django-admin start project .` to avoid too many directories

Sub-directory files

file	description
__init__.py	defines the directory as a python package
settings.py	defines our application settings
urls.py	where we define the URLs and routes of our application
asgi.py	Used for deployment (*Asynchronous*)
wsgi.py	Web Server Gateway Interface (*Used for deployment*)

Manage.py

In the root of our project, we have a file called `manage.py`. this is a wrapper around `django-admin` but takes our custom settings into account. So after the first time we create the project (Where we don't have any settings) we use `django-admin` but after that we use `python manage.py <command>`.

Run the web server

```
python manage.py runserver <port> → optional
```

> Creating the first App
>
> In Django projects, every logical block of our application is an app. if you head over to `INSTALLED_APPS` in **settings.py** you can see some pre

installed apps that Django comes with.

app	functionality
admin	gives us an admin interface for data / user management
auth	used for authenticating users
content types	for mapping models to numbers (*I think* 🤔)
sessions	temporary memory on the server for managing users data
messages	showing one time notifications to the user
static files	for serving Static files

We can also create our own apps here.

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

After creation, you add the app to `INSTALLED_APPS`

Views

HTTP is a **Request** / **Response** protocol. that's where **views** come in.
A view is a function that takes in a **request** and returns a **response**.

> View functions need to be mapped to a URL

Mapping URLs to Views

- **/app/endpoint**

1. Create the urls.py module in the app
2. Define the `urlpatterns` list or `routers`
3. Add the app to Main URL Conf

Using Templates

Dynamic HTML Content
Located in *templates* Directory -> always name space this directory with a

sub-directory that specifies the app name.

---

we use a **template** and the **render** function to show templates in views.
keywords:

- Context {{ }}
- {% if condition %}
- {% else %}
- {% endif %}
  Django template language can be changed.

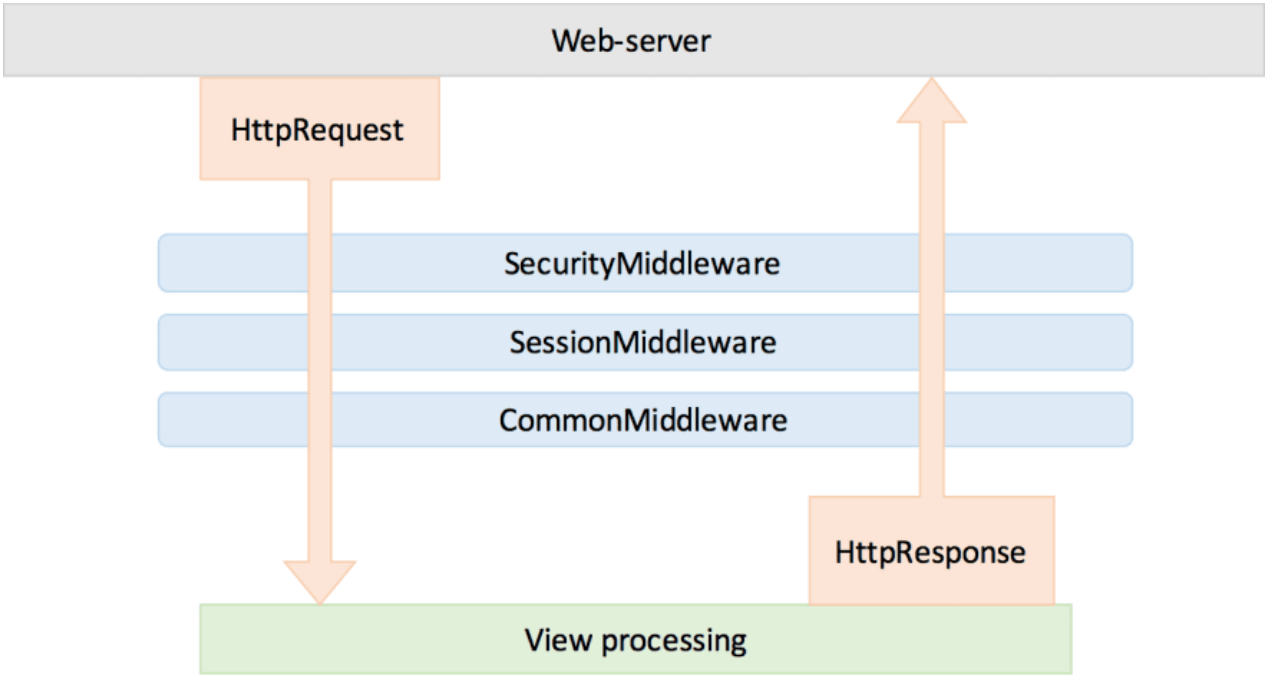
Using Django Debug Toolbar

Powerful debugging tool

- Info about:
  - SQL queries and redundant queries
  - Request
  - Execution time
  - settings
  - Headers
  - Django Version
  - Templates
  - Cache
  - ...

Install using documentation

Middleware is **a framework of hooks into Django's request/response processing**. It's a light, low-level “plugin” system for globally altering Django's input or output. Each middleware component is responsible for doing some specific function.


Web-server
HttpRequest
SecurityMiddleware
SessionMiddleware
CommonMiddleware
HttpResponse
View processing