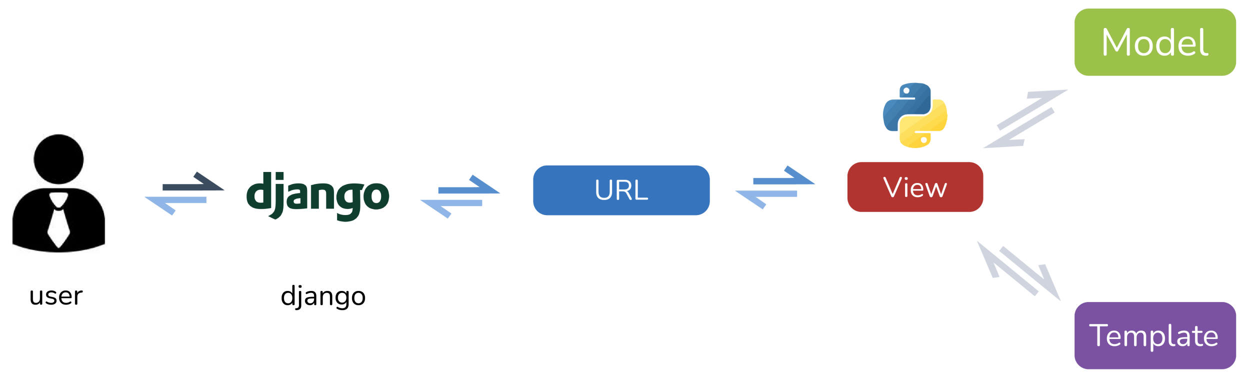
Django Views

Overview



Function-based views vs Class-based views

Django has two types of views; function-based views (FBVs), and class-based views (CBVs). Django originally started out with only FBVs, but then added CBVs as a way to templatize functionality so that you didn't have to write boilerplate (i.e. the same code) code over and over again.

A simple view

Here's a view that returns a simple HTML document:

```
# First, we import the class HttpResponse from the django.http module.
from django.http import HttpResponse

# Next, we define a function called home_view. This is the view function.
# Each view function takes an HttpRequest object as its first parameter, which is
typically named "request".
def home_view(request):
    html = "<html><body>Hello World!</body></html>"
    return HttpResponse(html)
    # The view returns an HttpResponse object that contains the generated
response.
    # Each view function is responsible for returning an HttpResponse object.
```

Another example with rendering an html page:

```
# To render an html page, import render.
from django.shortcuts import render
```

```
def home_view(request):
    # There will be a context, we will use it on the html page.
    context = {
        'first_name': 'Rafe',
        'last_name': 'Stefano',
    }
    return render(request, "app/home.html", context)
```

- More information about Request and response objects

Note that the name of the view function doesn't matter; it doesn't have to be named in a certain way in order for Django to recognize it. Give a meaningfull name which makes sense, clearly indicates what it does.

Django Templates

Being a web framework, Django needs a convenient way to generate HTML dynamically. The most common approach relies on templates. A template contains the static parts of the desired HTML output as well as some special syntax describing how dynamic content will be inserted.

A Django project can be configured with one or several template engines (or even zero if you don't use templates). Django ships built-in backends for its own template system, creatively called the Django template language (DTL).

The Django template language is Django's own template system.

The Django template language

A Django template is a text document or a Python string marked-up using the Django template language. Some constructs are recognized and interpreted by the template engine.

The syntax of the Django template language involves four constructs.

- Variables
- Tags
- Filters
- Comments

Variables

A variable outputs a value from the context, which is a dict-like object mapping keys to values.

Variables are surrounded by {{ and }} like this:

```
My first name is {{ first_name }}. My last name is {{ last_name }}.
```

With a context of `{'first_name': 'Rafe', 'last_name': 'Stefano'}`, this template renders to:

```
My first name is Rafe. My last name is Stefano.
```

Dictionary lookup, attribute lookup and list-index lookups are implemented with a dot notation:

```
{{ my_dict.key }}
{{ my_object.attribute }}
{{ my_list.0 }}
```

Tags

Tags reference

Tags provide arbitrary logic in the rendering process.

This definition is deliberately vague. For example, a tag can output content, serve as a control structure e.g. an “if” statement or a “for” loop, grab content from a database, or even enable access to other template tags.

Tags are surrounded by {% and %} like this:

```
{% csrf_token %}
```

csrf token reference

Most tags accept arguments:

```
{% cycle 'odd' 'even' %}
```

Some tags require beginning and ending tags:

```
{% if user.is_authenticated %}
    Hello, {{ user.username }}.
{% endif %}
```

A reference of built-in tags is available as well as instructions for writing custom tags.

Filters

Filters reference

Filters transform the values of variables and tag arguments.

They look like this:

```
{{ django|title }}
```

With a context of {'django': 'the web framework for perfectionists with deadlines'}, this template renders to:

```
The Web Framework For Perfectionists With Deadlines
```

Some filters take an argument:

```
{{ my_date|date:"Y-m-d" }}
```

Note that you don't have to memorize this filters. Always look at the documentation, and create your hands-on notes to easy access your frequently used filters. For `date` here is the documentation:

Drango filter: date

A reference of built-in filters is available as well as instructions for writing custom filters.

Comments

Comments look like this:

```
{# this won't be rendered #}
```

A {% comment %} tag provides multi-line comments.