Digital Career Institute

Python Course - Django Web Framework - Basics

Digital Career Institute

Goal of the Submodule

The goal of this submodule is to introduce the learners to the Django framework. By the end of this submodule, the learners will be able to:

- Explain what is the MVC design pattern.
- Understand how Django works and how does it implement the MVC.
- Set up a new Django project.
- Define URL endpoints using a variety of techniques and strategies.

Topics

- Web Frameworks: what are they and why do we use them.
- The MVC design pattern and Django's implementation.
- Django setup.
- URL Routing.
  - Paths, endpoints and views.
  - Using arguments in the URL.
  - Using and defining path converters.
  - Naming and namespacing.
  - URL utility functions: reverse & redirect.

Django Web Framework

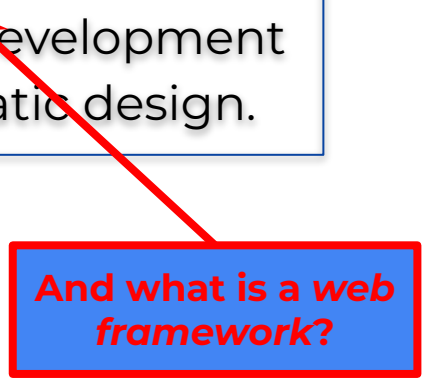
What is Django?

Digital Career Institute

Django is a high-level Python **web framework** that encourages rapid development and clean, pragmatic design.

What is Django?

Django is a high-level Python **web framework** that encourages rapid development and clean, pragmatic design.



What is a Web Framework?

Django is a high-level **Python** web framework that **encourages rapid development and clean, pragmatic design**.

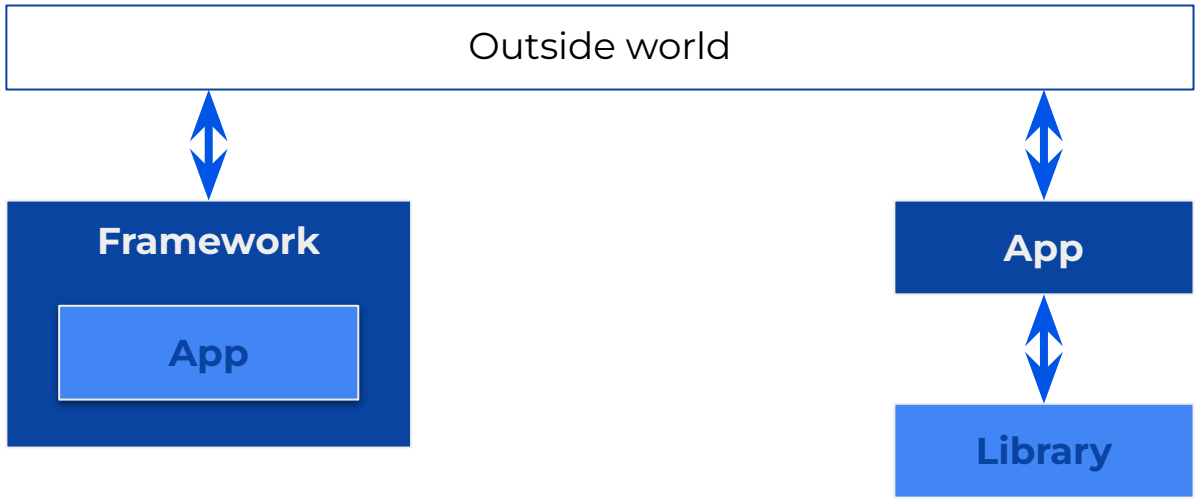
A web framework:

- Improves productivity.
- Is language specific.
- Provides a higher-level of abstraction.

What is a Web Framework?

Photo by Patrick on Unsplash

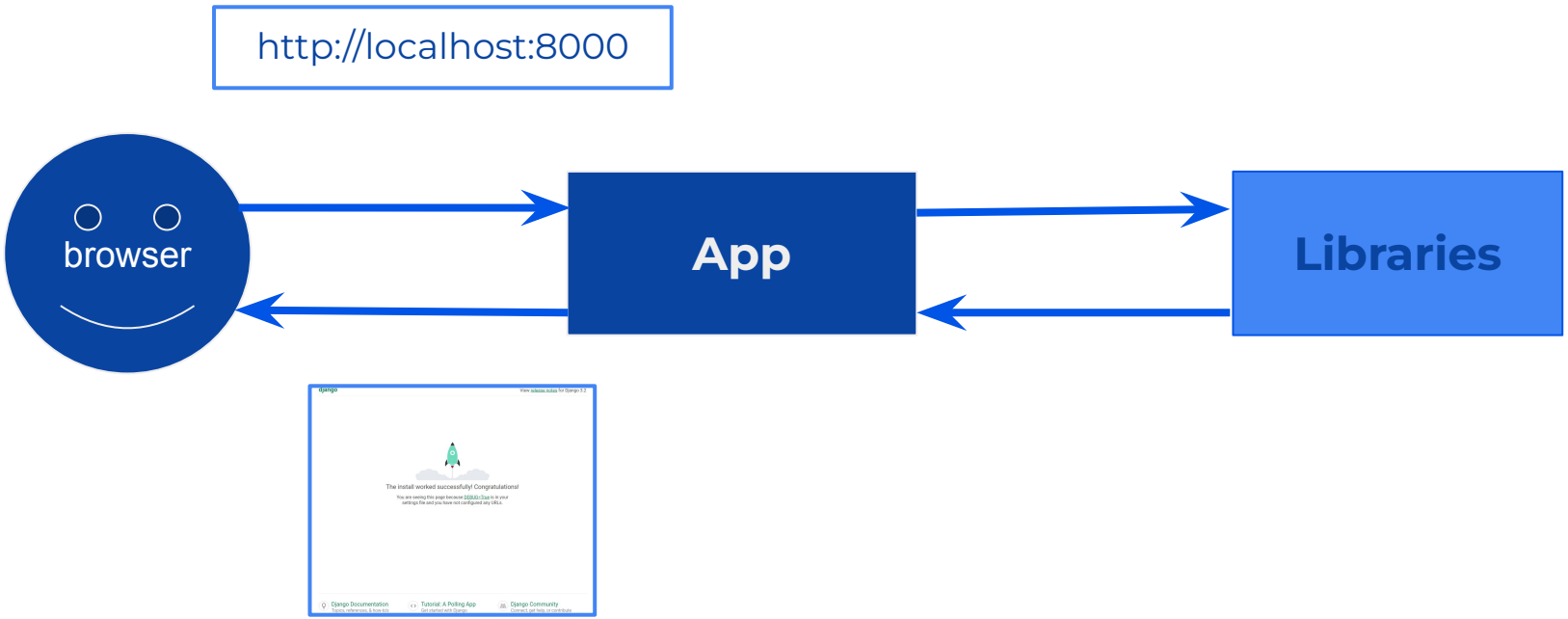
Framework vs. Library



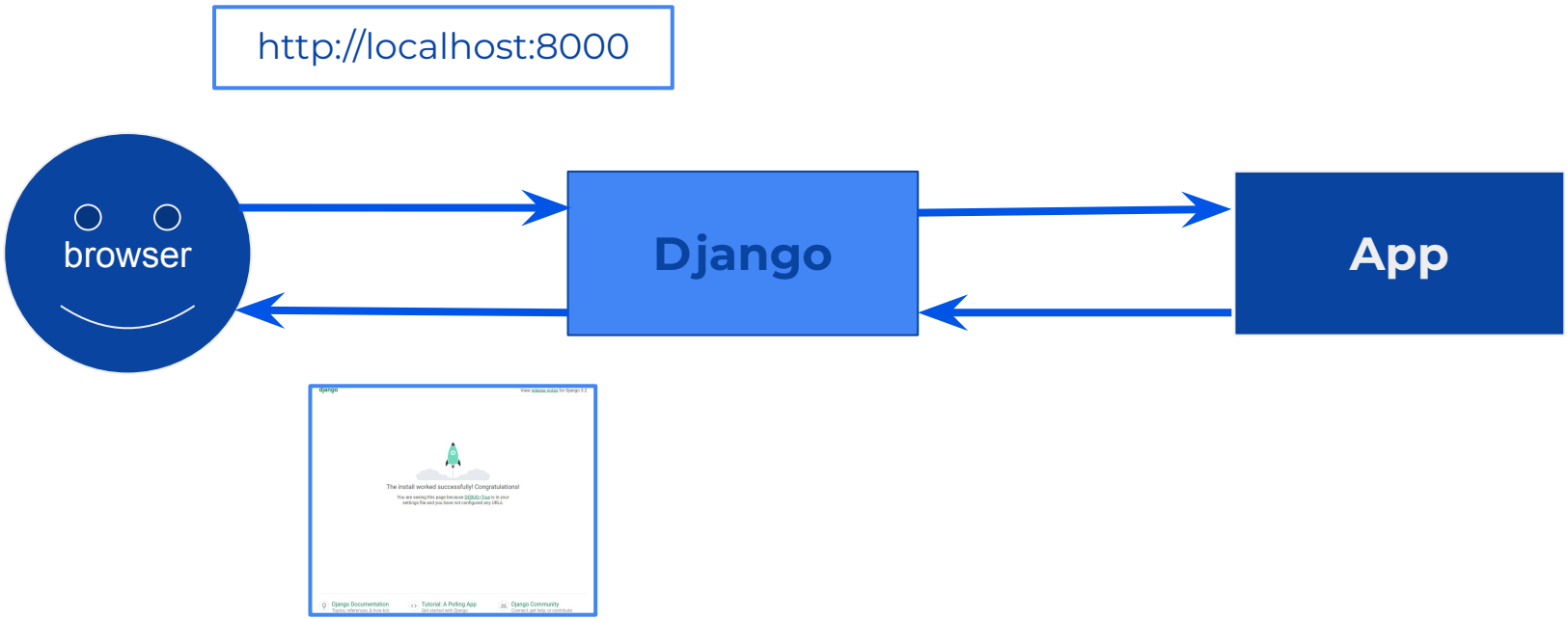
When we use a framework much of the **control flow** happens on the **framework code**.

When we use a library the **control flow** happens on **our code**.

General Control Flow: No Framework



General Control Flow: Framework



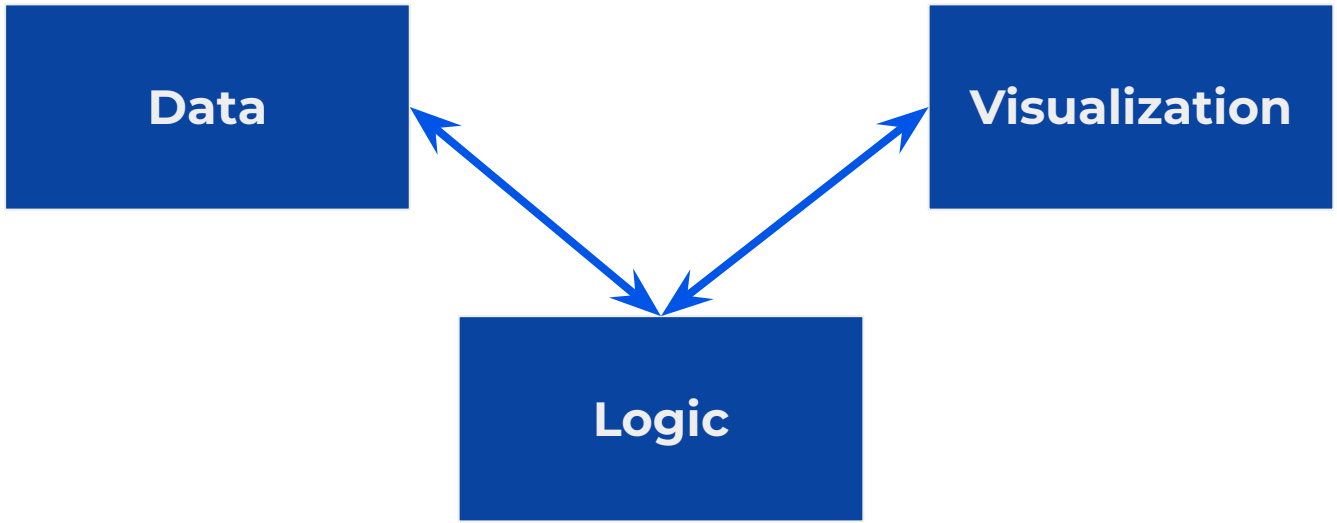
Web Framework Rules

"With great power comes great responsibility"

A web framework gives us power, but also defines some **rules** concerning the way we type our code and the way we organize it.

Every web framework has its own set of rules, but most of them share a **common design pattern**.

Common Design Pattern



Common Design Pattern

Defines the data used in our application (users, shopping list, ...).

Defines ways to communicate with the **database** to retrieve and store data and pass it on to our logic.

The part of our code where we define the data is a bit like a mirror of the **database tables**.

Often called **Model**.

Common Design Pattern

Defines how the data should be presented to the user.

Example: Should there be a dropdown? A simple list?

Visualization

The part of our code where we define the visualization will be our **User Interface**.

Often called **View**.

Common Design Pattern

There must be a point that integrates our code with the framework control flow and triggers the usage of the data and visualization on request.

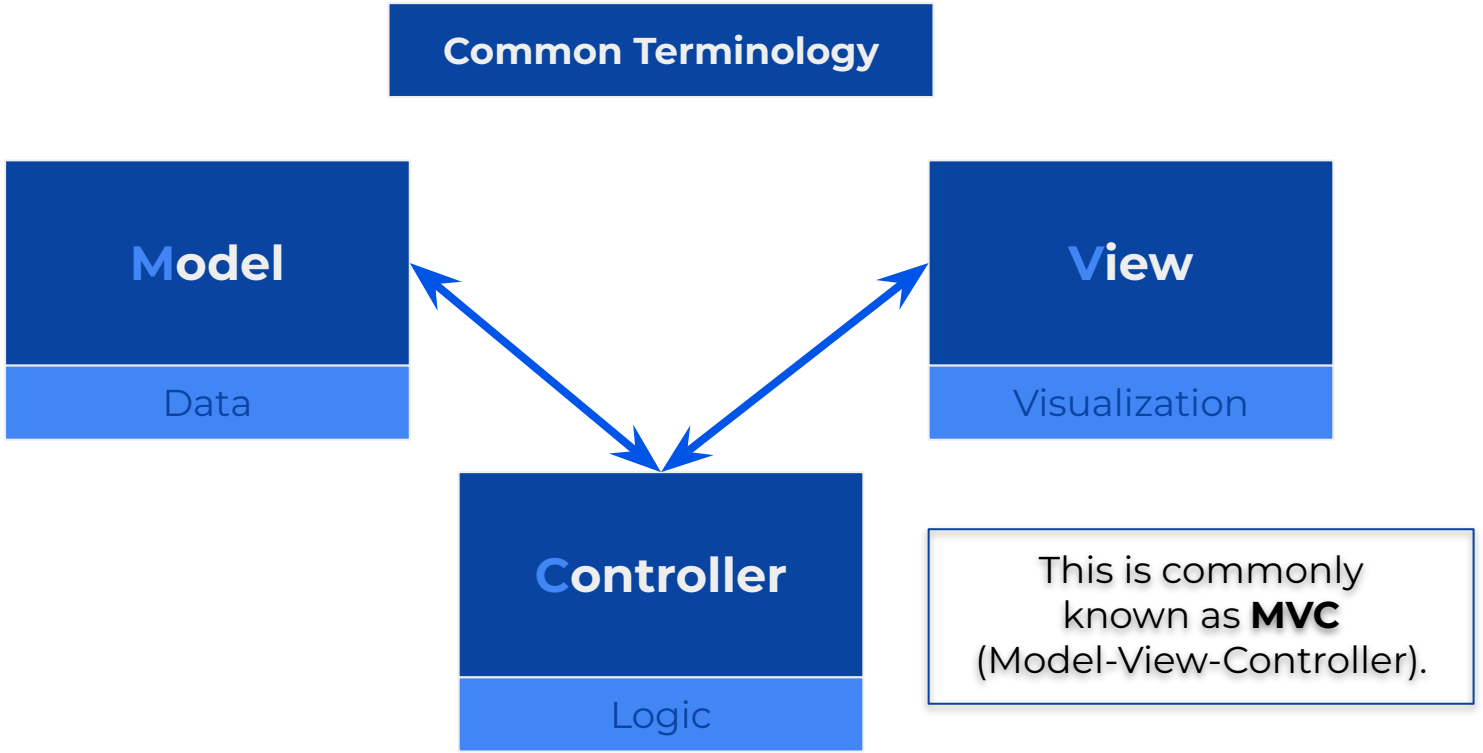
This is what we call the logic of the application.

The part of our code where we define our control flow.

Often called **Controller**.

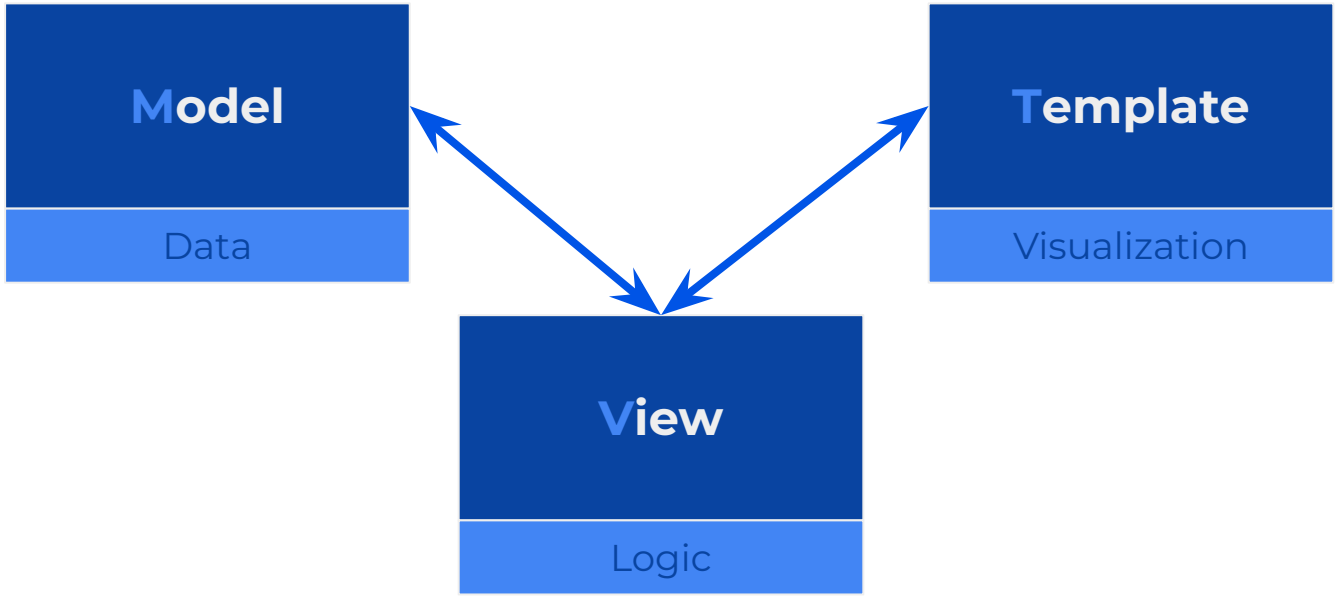
Logic

The MVC Design Pattern



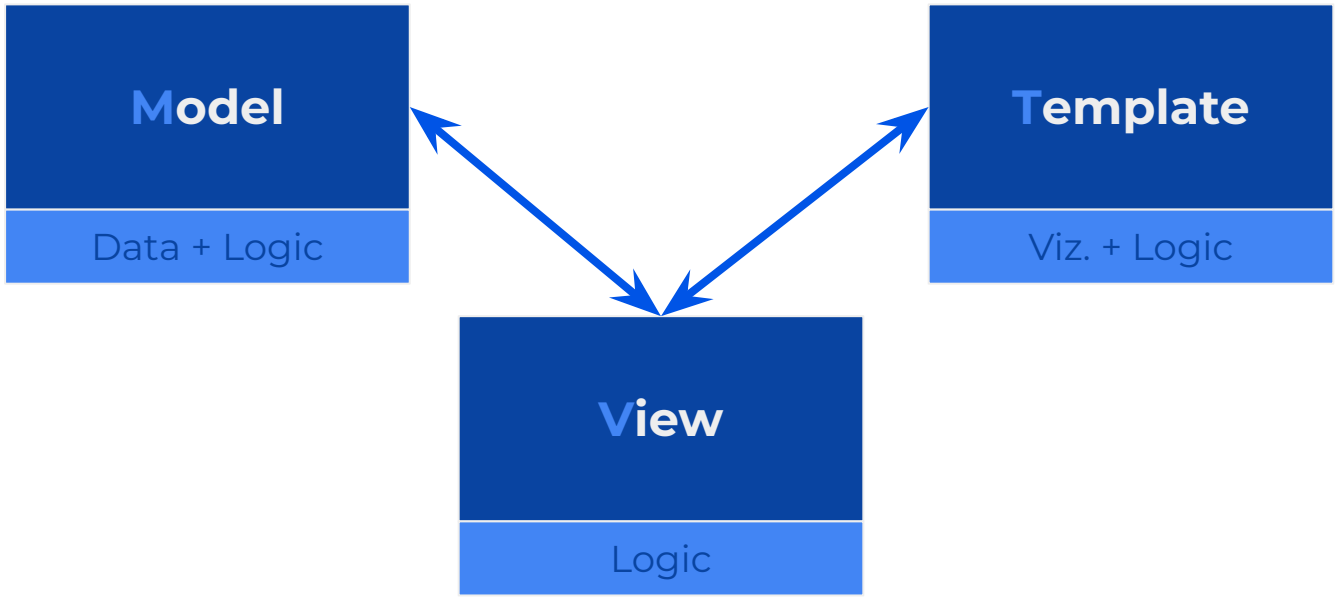
Django's "MVC" vision

This alternative terminology is exclusive to Django and is sometimes known as **MTV**.

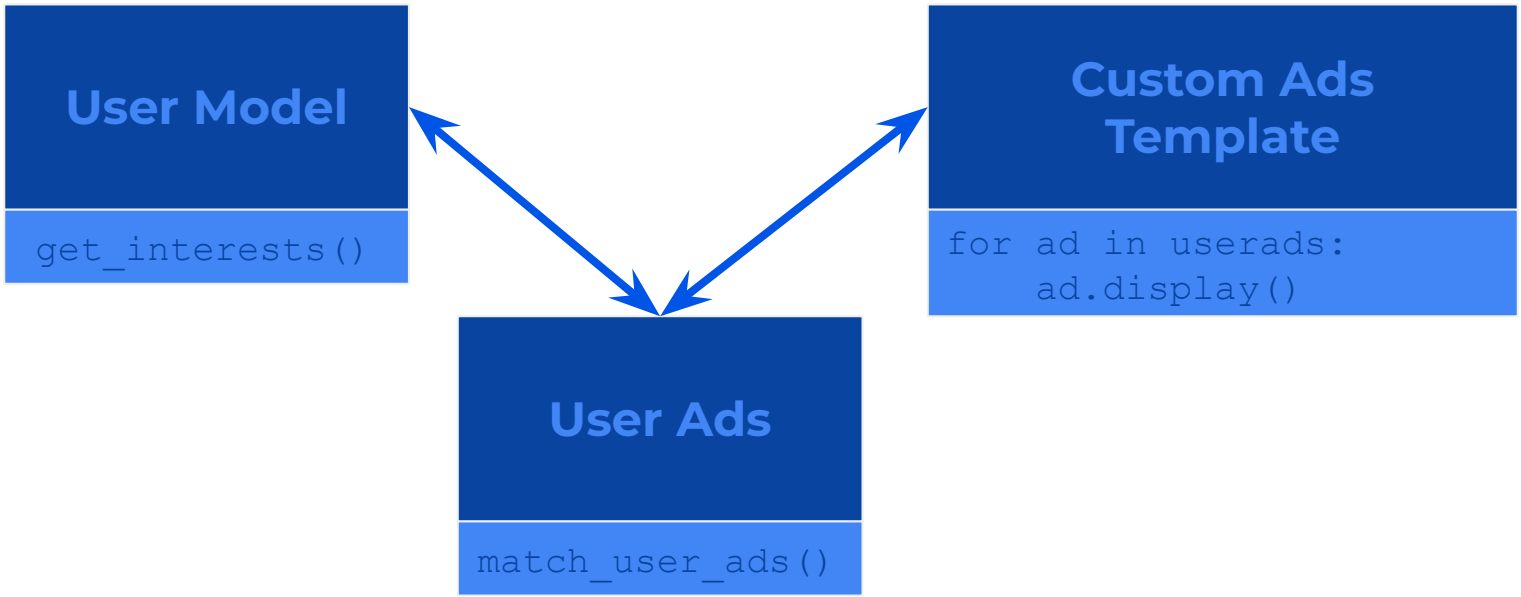


Django's "MVC" vision

In many MVC frameworks, like Django, the logic is actually split.



The Logic in the MVC



Web Framework Examples

Front-end

JavaScript

ReactJS
VueJS
Angular
...

CSS

Bootstrap
Materialize
Tailwind
...

Back-end

Python

Django
Flask

JavaScript

Next.js
Express
...

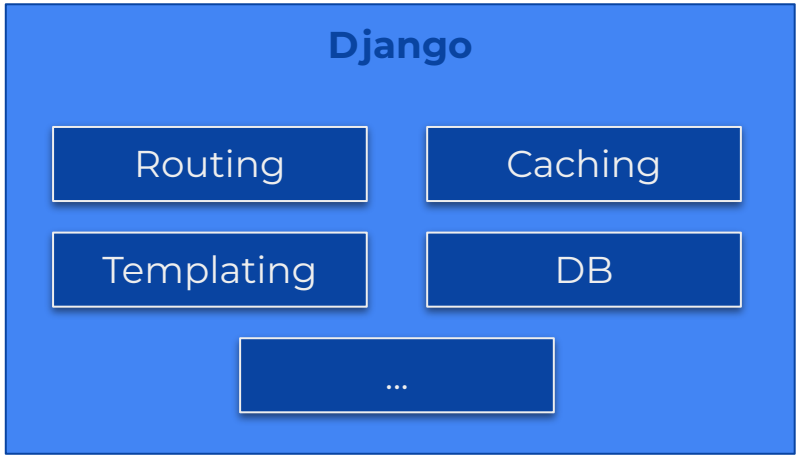
PHP

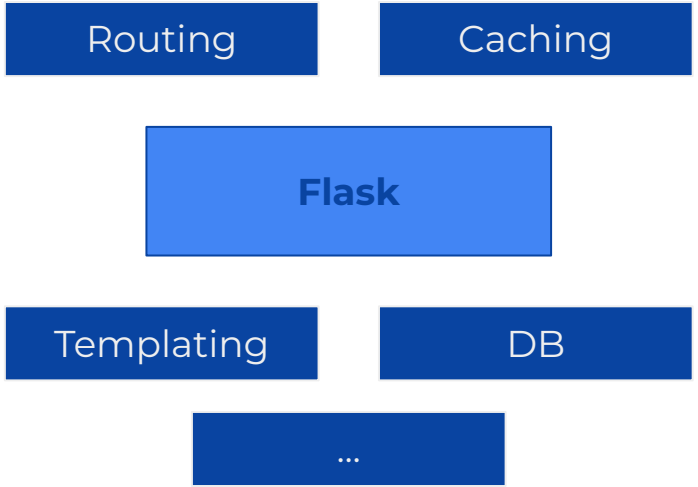
Laravel
CakePHP
...

Ruby

Ruby on Rails

Python Frameworks: Django vs. Flask





We learned ...

- What role plays Django for web development.
- What is a Web Framework and why it is useful.
- That the MVC design pattern is a common architectural pattern among web frameworks.
- That Django uses the same pattern with a different perspective and terminology.
- That frameworks are language-specific and there are two main Python web frameworks: Django and Flask.

Django Setup

Digital Career Institute

Django Setup

The first step is to install the Django framework in our virtual environment.

```
(env)$ pip install django
```

Django-admin

Installing Django provides a command named `django-admin`.

```
(env)$ django-admin command parameters
```

`django-admin` is a utility program that we can use to perform various operations related to Django.

Django Projects

`django-admin` can be used to start a new Django project.

```
(env)$ django-admin startproject hello
```

This will create a directory named `hello` with some initial files.

Django Projects: Directory Tree

```
+  hello
   + hello
       - __init__.py
       - asgi.py
       - settings.py
       - urls.py
       - wsgi.py
   - manage.py
```

Important initial files:

- **manage.py**
  Is a utility module to execute various Django commands.
- **hello/settings.py**
  Django's configuration file.
- **hello/urls.py**
  Main initial point of entry when someone sends a request to our Django application.

Django Projects: Directory Tree

```
+  hello
   + project
        - __init__.py
        - asgi.py
        - settings.py
        - urls.py
        - wsgi.py
   - manage.py
```

A version of the previous command sometimes is used to reduce the name redundancy:

```
(env)$ mkdir hello
(env)$ cd hello
(env)$ django-admin startproject project .
```

The directory where the project should be created.

Manage.py: Running a Django Shell

```
(env)$ python manage.py shell
Python 3.6.9 (default, Jul  3 2019,
15:36:16)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits"
or "license" for more information.
(InteractiveConsole)
>>> from django.conf import settings
>>>
```

Django comes with a custom Python shell interface where we can do all sorts of ad-hoc testing.

Manage.py: Running a Web Server

```
(env)$ python manage.py runserver
...
Django version 3.2.6, using settings
'hello.settings'
Starting development server at
http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Django comes with a web server meant to be used for development purposes.

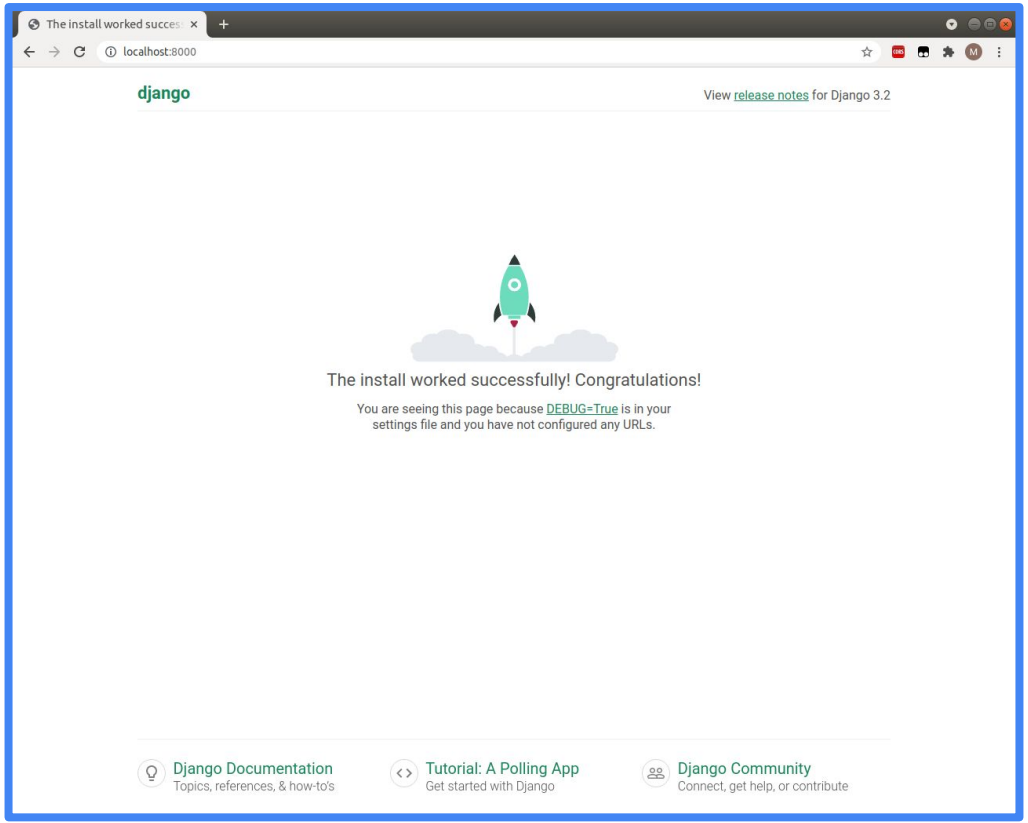
It is run by using Django's command `runserver`.

By default, the web server listens to localhost requests on port 8000, although this can be changed:

```
manage.py runserver myhost:3000
```

We must leave this running while we work. When we want to finish, we can do it with `Ctrl+C`.

Manage.py: Running a Web Server



Django Apps

A Django app is a component of our website that packs a set of common features (a shop, a user list, a blog, ...).

```
(env)$ cd hello
(env)$ django-admin startapp shop
```

This will create a directory named **`shop`** inside our main **`hello`** directory.

Django Apps: Directory Tree

```
+ shop
  + migrations
    - __init__.py
  - __init__.py
  - admin.py
  - apps.py
  - models.py
  - tests.py
  - views.py
```

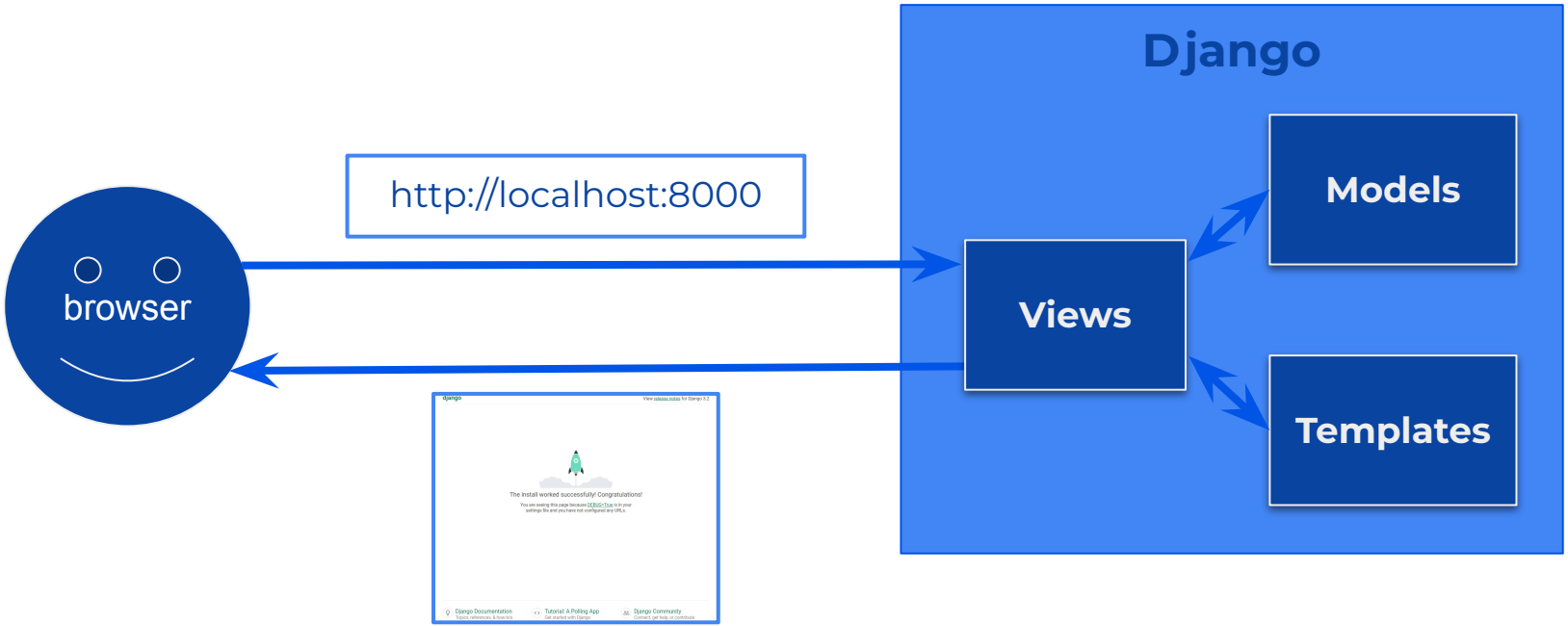
Some first important files:

- **models.py**
  Defines the models of the app.
- **views.py**
  Defines the views of the app.

Project vs. App

Project	App
The initial directory does not contain any `models.py` or `views.py` file.	The initial directory contains a `models.py` and a `views.py` files.
The directory contains the **settings** and serves as **main entry point** of the control flow.	The directory contains the code of each app, structured in **models**, **views** and **templates**.

Control Flow Reviewed



We learned ...

- How to install Django.
- How to run a development web server and access it with the browser.
- What are Django projects and apps and how to create them.
- Where are located some of the most important files of our code: models and views.
- That there is a configuration file named `settings.py`, and a file named `urls.py` that is the entry point for every request.

URL Mapping

URL Mapping

URL Endpoints

http://localhost:8000

http://localhost:8000/users

http://localhost:8000/blog

http://localhost:8000/shop

Views

```
def home:
    ...
```

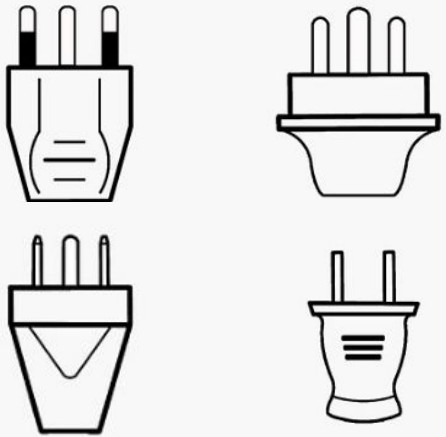
```
def users:
    ...
```

```
def blog:
    ...
```

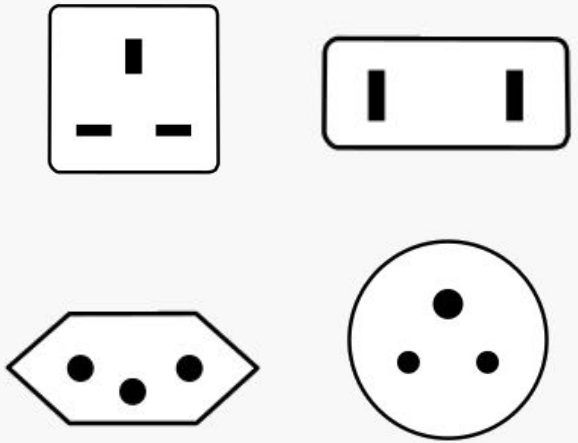
```
def shop:
    ...
```

URL Mapping

URL Endpoints

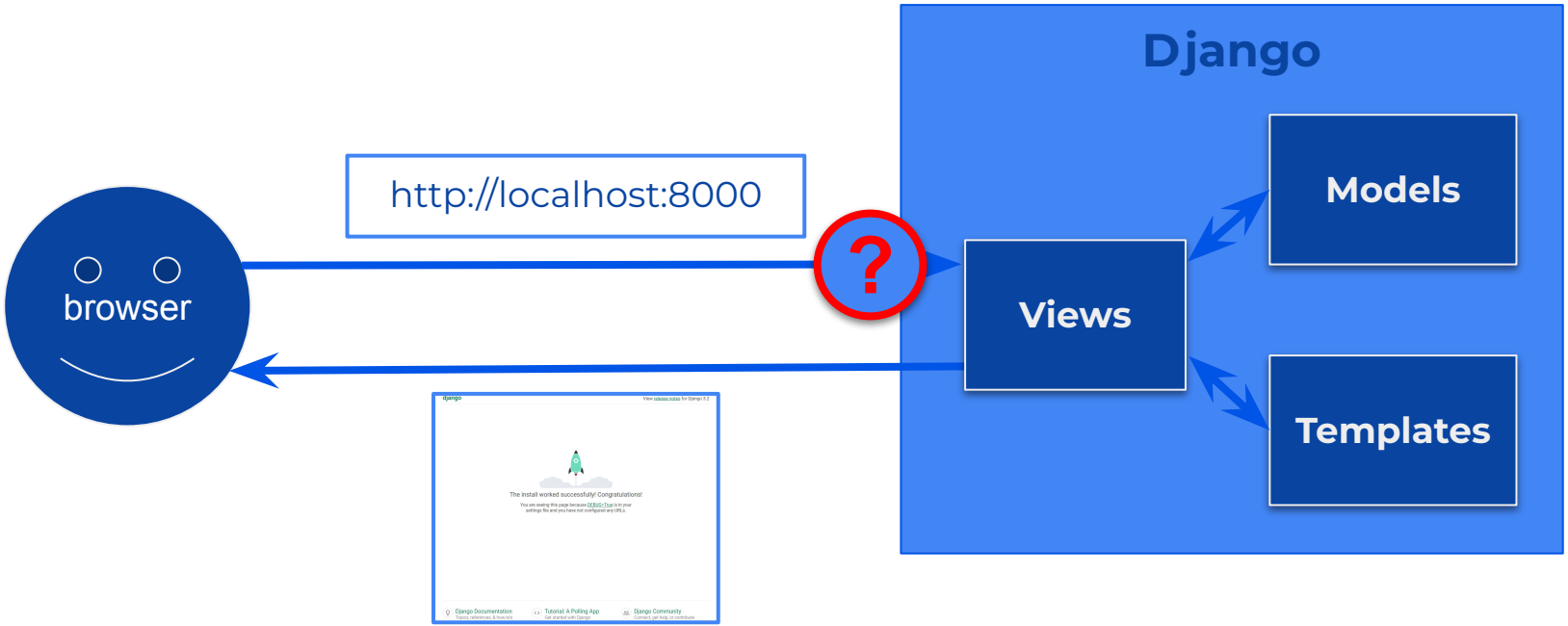


Views



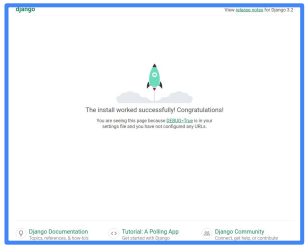
Which one goes where?

Mapping URL to Views

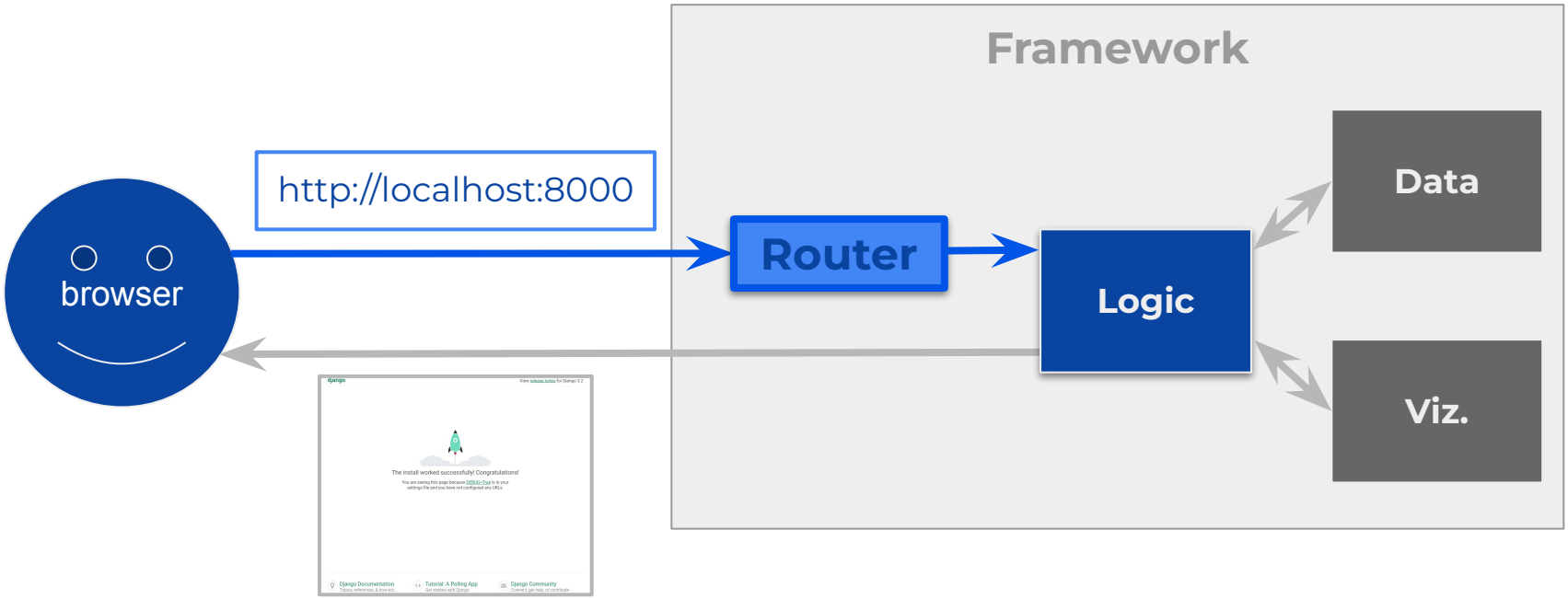


URL Mapping Without Framework

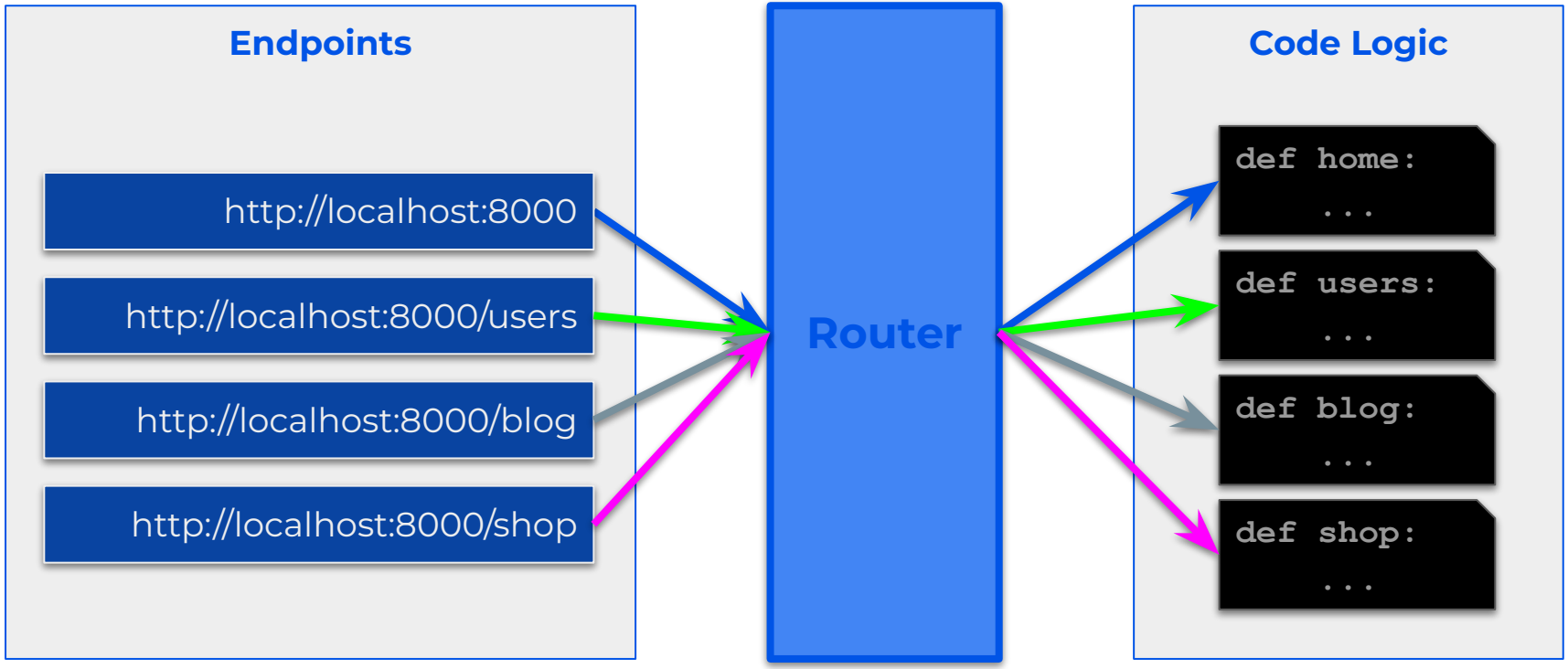
http://localhost:8000/**directory/subdirectory/file.extension**



URL Mapping With a Framework



The Web Router



Django's Router: URL Dispatcher

hello/urls.py

```
from django.contrib import admin
from django.urls import path
from shop.views import home as shop_home

urlpatterns = [
    path('admin/', admin.site.urls),
    path('shop/', shop_home)
]
```

When an HTTP request is received pointing at ***`http://localhost:8000/shop/`*** *Django will execute the content of the view named* ***`shop_home`***.

Django's configuration of the URL dispatcher is called **URLconf** and takes the form of files where paths are defined.

Django's Views

shop/views.py

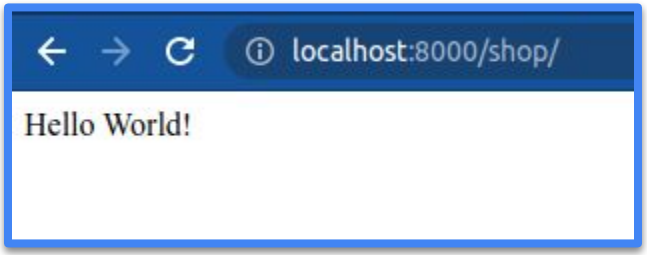
```
from django.http import HttpResponse


def home(request):
    """The shop home view."""
    return HttpResponse("Hello World!")
```

A view is a function.

All views will receive, at least, an object with the HTTP **request**.

All views must return an HTTP **response**.



Django's Web Router

hello/urls.py

```
from django.contrib import admin
from django.urls import path
from shop.views import (
    home as shop_home,
    listing as shop_list
)

urlpatterns = [
    path('admin/', admin.site.urls),
    path('shop/', shop_home),
    path('shop/browse/', shop_list),
]
```

When an HTTP request is received pointing at ***`http://localhost:8000/shop/browse/`*** *Django will execute the content of the view named* ***`shop_list`***.

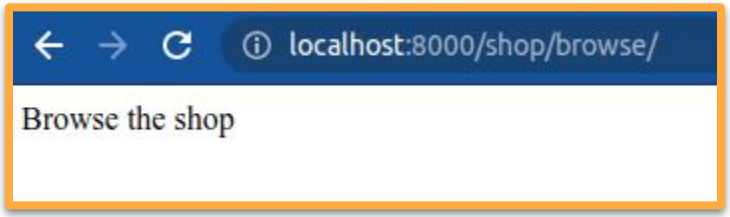
There may be as many paths as needed in the **`urlpatterns`** sequence.

Django's Views

shop/views.py

```
...


def listing(request):
    """The shop list."""
    return HttpResponse("Browse the shop")
```



URLconf

Keyword Parameters

hello/urls.py

```
from django.contrib import admin
from django.urls import path
from shop.views import (
    item as shop_item
)

urlpatterns = [
    path('admin/', admin.site.urls),
    path('shop/', shop_home),
    path('shop/<item_id>/', shop_item),
]
```

If we add **<variable name>** to a path, the view will receive its value as a keyword argument with that name.

The third path in this example will match anything starting with **shop/** followed by any character, and that string will be passed as an argument named **item_id**.

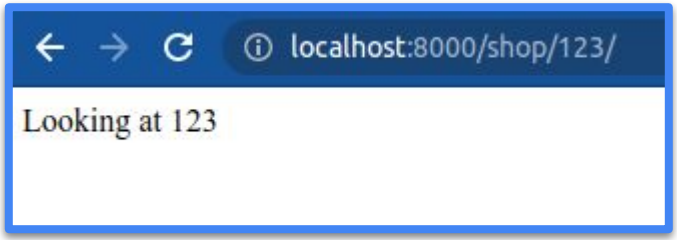
Matching examples:
/shop/123/
/shop/anything/

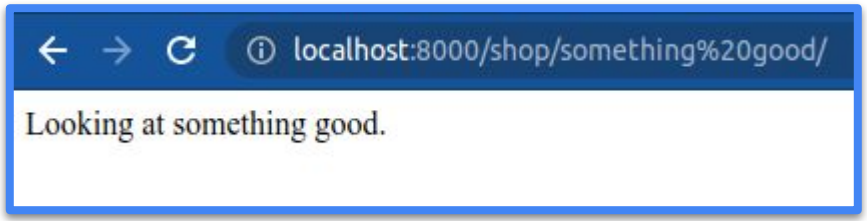
Keyword Parameters

shop/views.py

```
...


def item(request, item_id):
    """An item of the shop."""
    return HttpResponse(f"Looking at {item_id}")
```





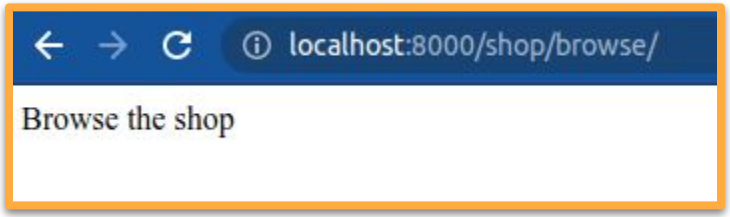
Path Collisions

hello/urls.py

```
...

urlpatterns = [
    path('shop/browse/', shop_list),
    path('shop/<item_id>/', shop_item),
]
```

When using endpoint parameters, more than one path may match the same request.

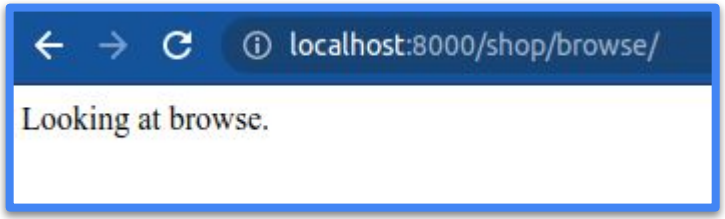


Paths Order

hello/urls.py

```
...

urlpatterns = [
    path('shop/<item_id>/', shop_item),
    path('shop/browse/', shop_list),
]
```



The endpoint **`shop/browse/`** matches both paths.

Django takes the first one in the sequence.

In this case, tries to show an item called **`browse`**.

Parameter Types

hello/urls.py

```
...

urlpatterns = [
    path('admin/', admin.site.urls),
    path('shop/', shop_home),
    path('shop/<int:item_id>/', shop_item),
    path('shop/<str:term>/', shop_search),
]
```

These are called **path converters** and they are used to match a particular type against the received request URL.

The type of the argument can be made explicit in the endpoint definition.

Django matches the path including the type.

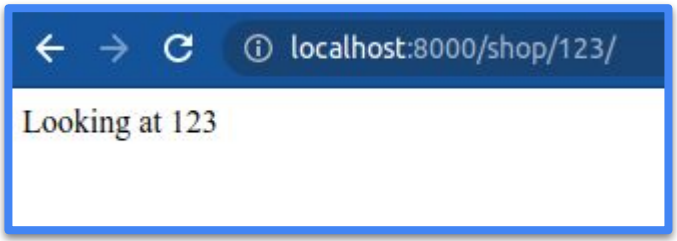
The endpoint ***`shop/shirt/`*** *only matches the last path.*

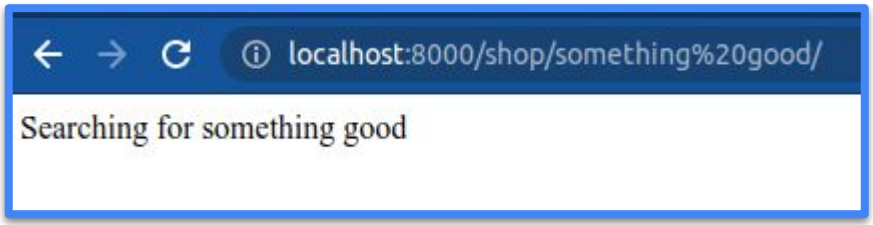
Path Converters

shop/views.py

```
def item(request, item_id):
    """An item of the shop."""
    return HttpResponse(f"Looking at {item_id}")

def search(request, term):
    """Search the shop."""
    return HttpResponse(f"Searching for {term}")
```





Path Converters

	Matches	*Example*
str	Any non-empty string, excluding `/`.	*söméthiñg%20good!*
int	0 or a positive integer.	*1342*
slug	ASCII letters/numbers, `-` and `_`.	*something-good*
uuid	A formatted UUID.	*6c92aa43-4947-4c2e-a819-3da247eae93e*
path	Any non-empty string, including `/`.	*söméthiñg/good!*

Regular Expression Paths

hello/urls.py

```
from django.contrib import admin
from django.urls import path,  re_path
from shop.views import (
    item as shop_item
)

urlpatterns = [
    path('admin/', admin.site.urls),
    re_path('shop/(.*)/', shop_item),
    path('shop/<item_id>/', shop_item)
]
```

Positional parameters can be defined with **regular expressions**.

The last two paths in this example are similar.

The first one calls the view with positional arguments and the second one with keyword arguments.

The `re_path` function allows the use of *regular expressions* in endpoint match definition.

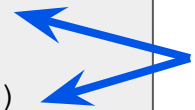
Regular Expression Paths

hello/urls.py

```
from django.contrib import admin
from django.urls import path, re_path
from shop.views import (
    item as shop_item
)

urlpatterns = [
    path('admin/', admin.site.urls),
    re_path('shop/(?P<item_id>.*)/',
            shop_item),
    path('shop/<item_id>/', shop_item)
]
```

Keyword parameters can also be defined with **regular expressions**.



The last two paths in this example are equivalent, and both call the view using keyword arguments.

Regular Expression Paths

hello/urls.py

```
...

urlpatterns = [
    path('admin/', admin.site.urls),
    re_path('articles/year/([0-9]{4})/',
            article_list)
]
```

With regular expressions we can include the type as a matching rule, but also a custom format.

In this example, the `re_path` *will only match the endpoints having a 4 digit numeric parameter.*

If it is a 3 digit or 5 digit number, for instance, it will not match the path.

Path Extra Options

hello/urls.py

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('articles/<int:month>/',
         article_list,
         {'is_int': True}),
    path('articles/<str:month>/',
         article_list,
         {'is_int': False})
]
```

The view function now must be redefined to accept a keyword parameter called **`is_int`**.

The **`path`** function accepts a third parameter as a dictionary with additional options to pass on to the view as **keyword arguments**.

Naming Paths

hello/urls.py

```
from django.contrib import admin
from django.urls import path
from shop.views import (
    shop_home, shop_item
)

urlpatterns = [
    path('shop/', shop_home,
         name="shop"),
    path('shop/<item_id>/', shop_item,
         name="shop-item"),
]
```

Paths may have a **`name`**.

Names are usually written in kebab-case.

This will be useful when writing the links in our HTML output.

Digital Career Institute

Python Course - Django Web Framework - Basics

Digital Career Institute