Web Programming

Django I

Prof. Josué Obregón
Department of Industrial Engineering- ITM
Seoul National University of Science and Technology

Objectives

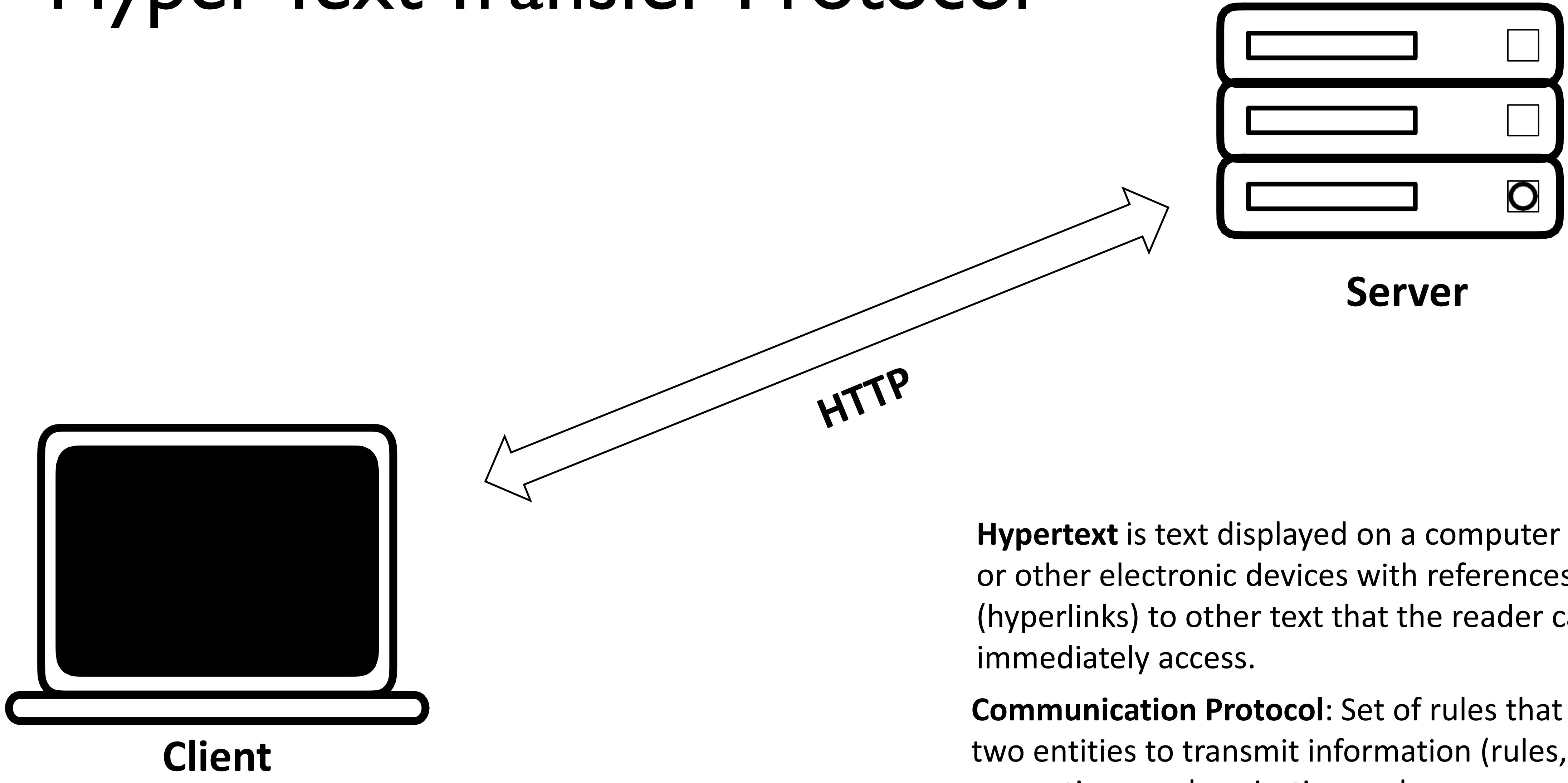
- Understand how different components of web applications communicate with each other.
- Learn one of the most typical architectural design patterns for modern web applications, the MVC framework.
- Learn what Django is and how it is used to help us easily create web applications.
- Use the Django framework to write a simple but dynamic web application.

Agenda

- Web applications
    - http
    - MVC
- Django
    - Django architecture
    - Django project structures
    - Example project
        - Views
        - URLs
        - Templates

Web applications

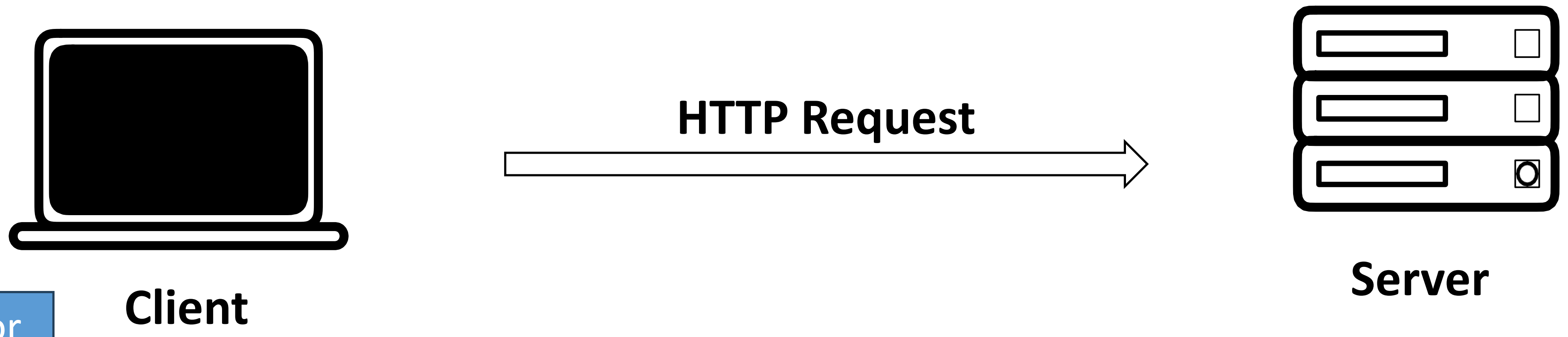
Hyper Text Transfer Protocol



Hypertext is text displayed on a computer display or other electronic devices with references (hyperlinks) to other text that the reader can immediately access.

Communication Protocol: Set of rules that allows two entities to transmit information (rules, syntax, semantics, synchronization and error recovery)

HTTP Request



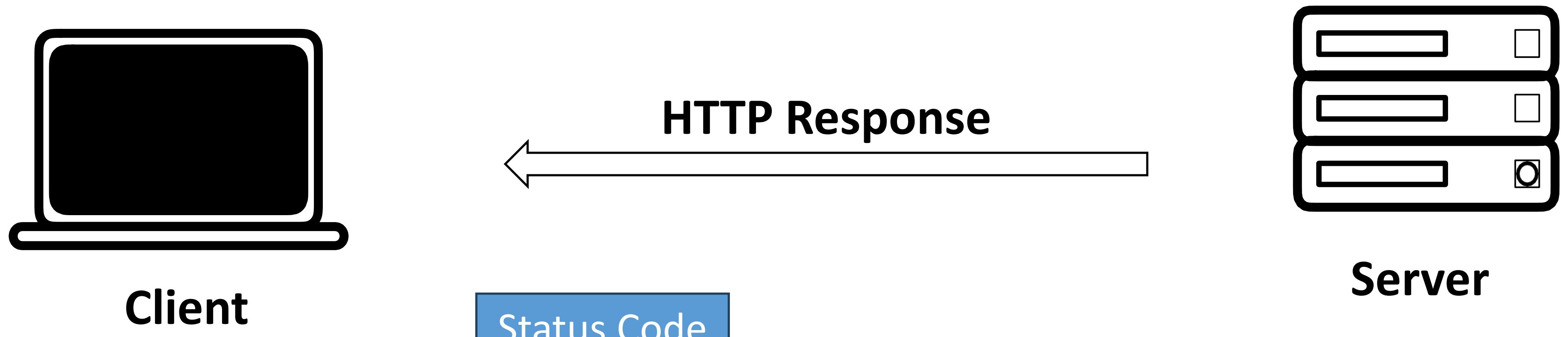
Method or Verb

```
GET / HTTP/1.1
Host: www.example.com
Accept-Language: en
...
```

Other methods

Method	Action
POST	Create
GET	Read
PUT	Update
DELETE	Delete

HTTP Response



Status Code

```
HTTP/1.1 200 OK
Content-Type: text/html
...
```

Other code types

Status Code	Description
1xx	informational
2xx	Successful
3xx	Redirection
4xx	Client error
5xx	Server error

HTTP Status Codes

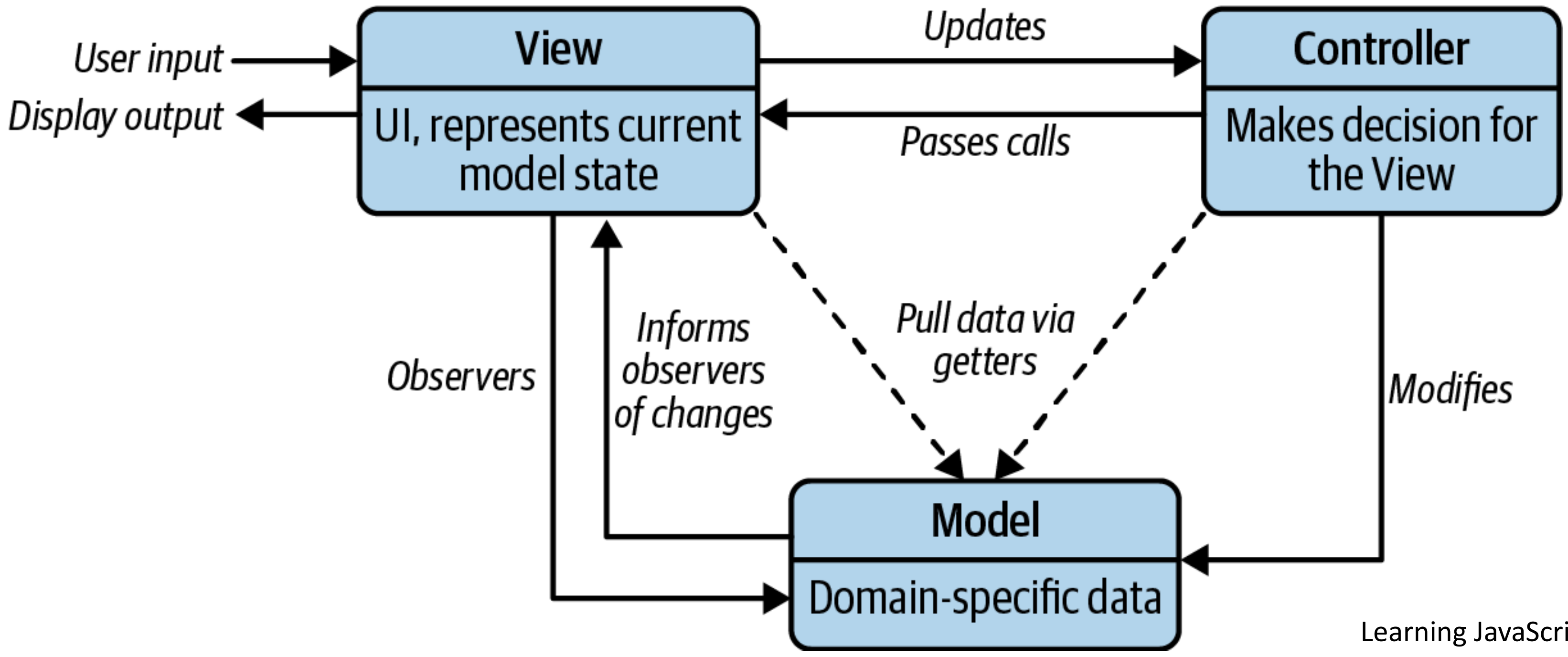
Status Code	Description
200	OK
301	Moved Permanently
403	Forbidden
404	Not Found
500	Internal Server Error

Full list: https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

Web applications Architectural Design

MVC

- **Models** – primarily concerned with business data
- **Views** – visual representation of Models that present a filtered view of their current state.
- **Controllers** – intermediaries between Models and Views, which are classically responsible for updating the Model when the user manipulates the View



Learning JavaScript Design Patterns, 2nd edition, Addy Osmany, O'reilly Media

Django

Django

- Web framework written in Python

- Build features fast
  - we can focus on logic

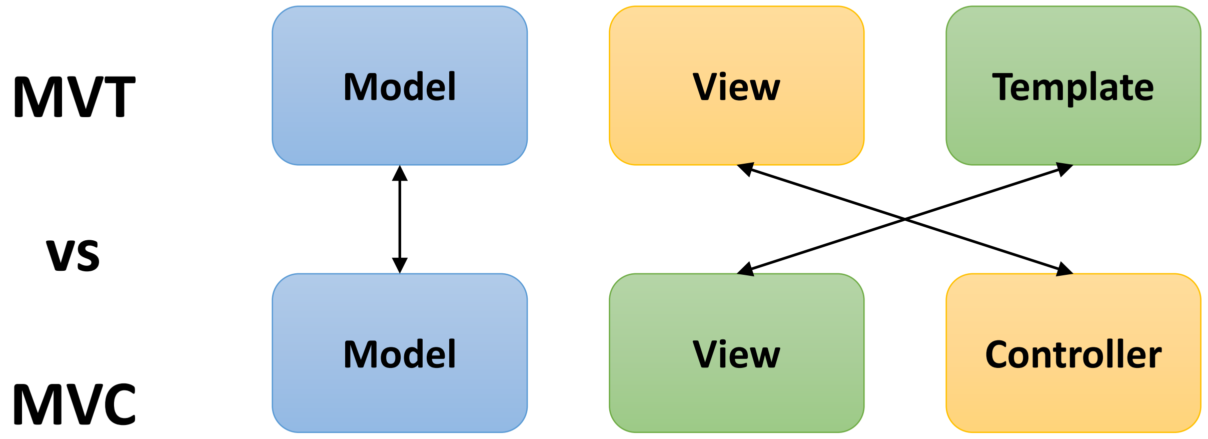
- Security built-in

- Scales in size well

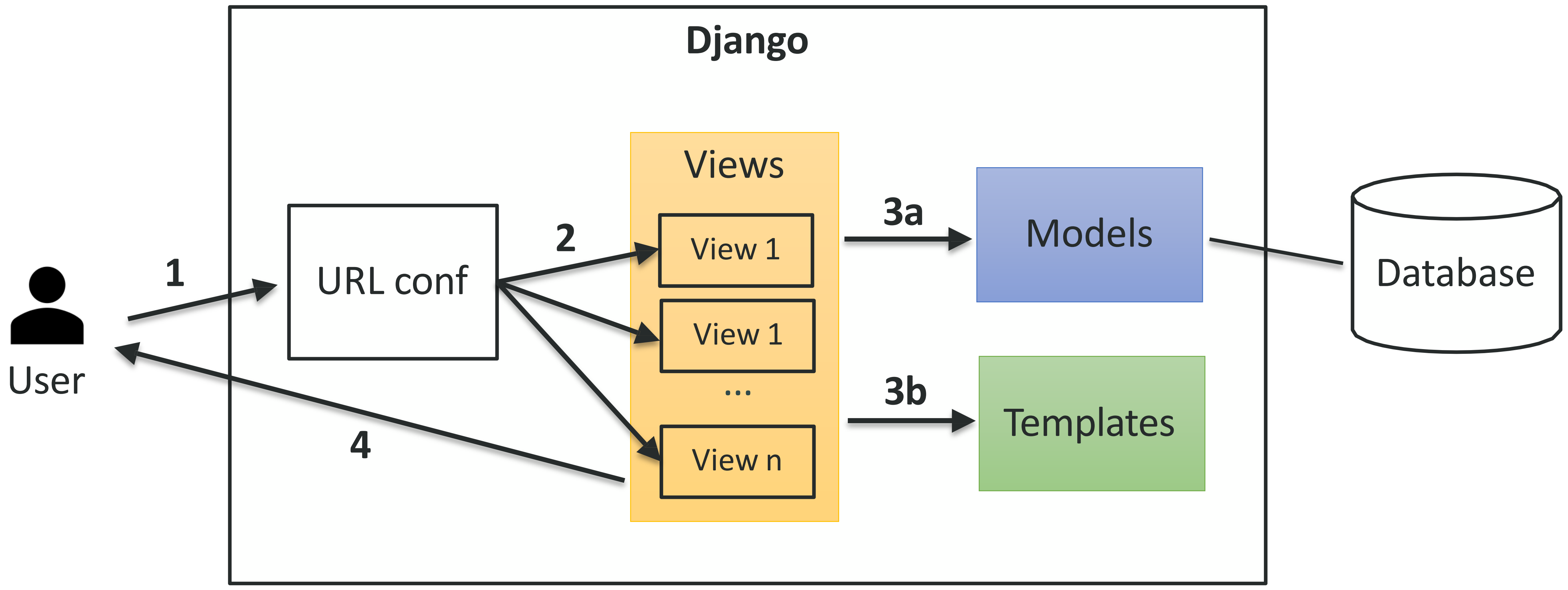
https://www.djangoproject.com/

Overall architecture

- Uses an MVT pattern:
  - **Models** – Interact with the database via an ORM (Object Relational Mapper)
  - **Views** – Handle HTTP requests and return responses
  - **Templates** – Create dynamic HTML pages from Python data



Django architecture overview



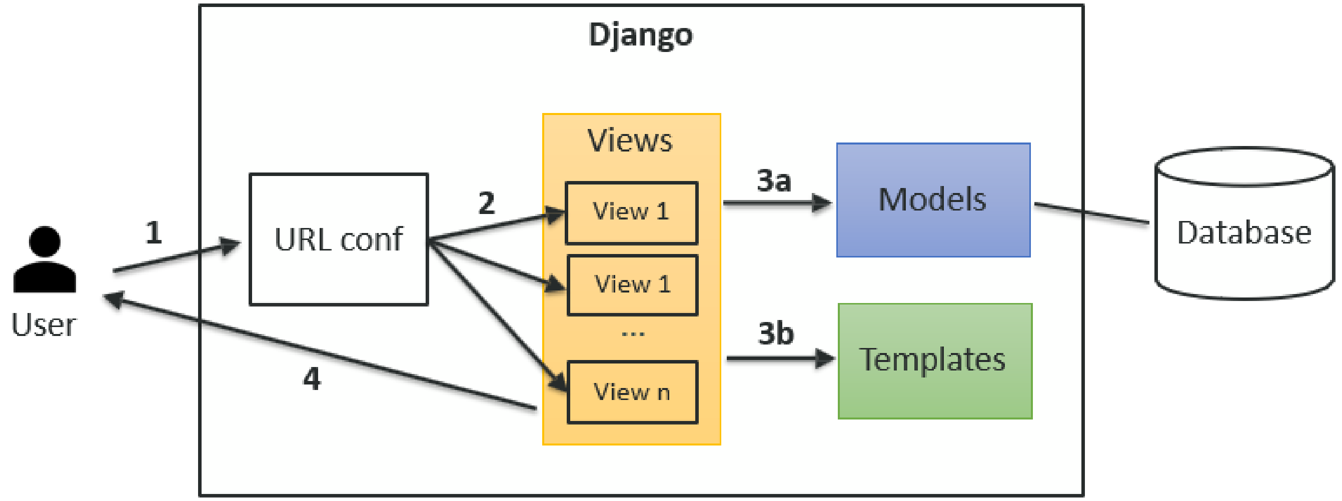
Django architecture

1. User sends an HTTP **request** to Django

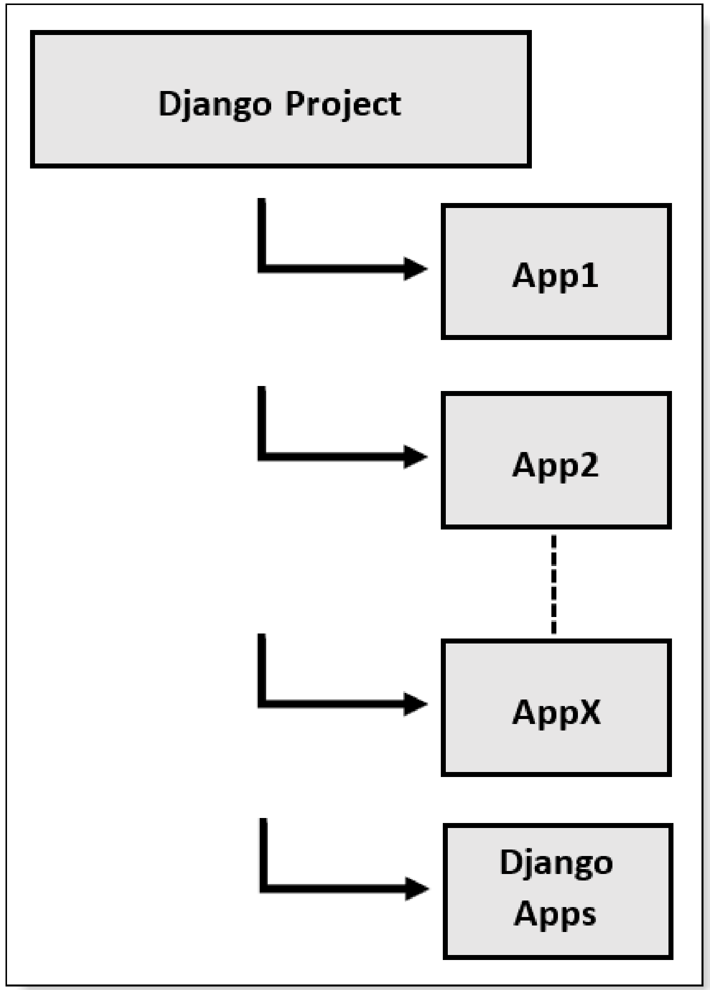
2. **URL configuration**, contained in **urls.py**, selects a View to handle the request

3. The **View**, contained in **views.py** gets the request and
   a) Talks to a database via the **Models** (**models.py**)
   b) Renders an HTML **Template**

4. The View returns an **HttpResponse** which gets sent to the client to be rendered as a web page in the browser

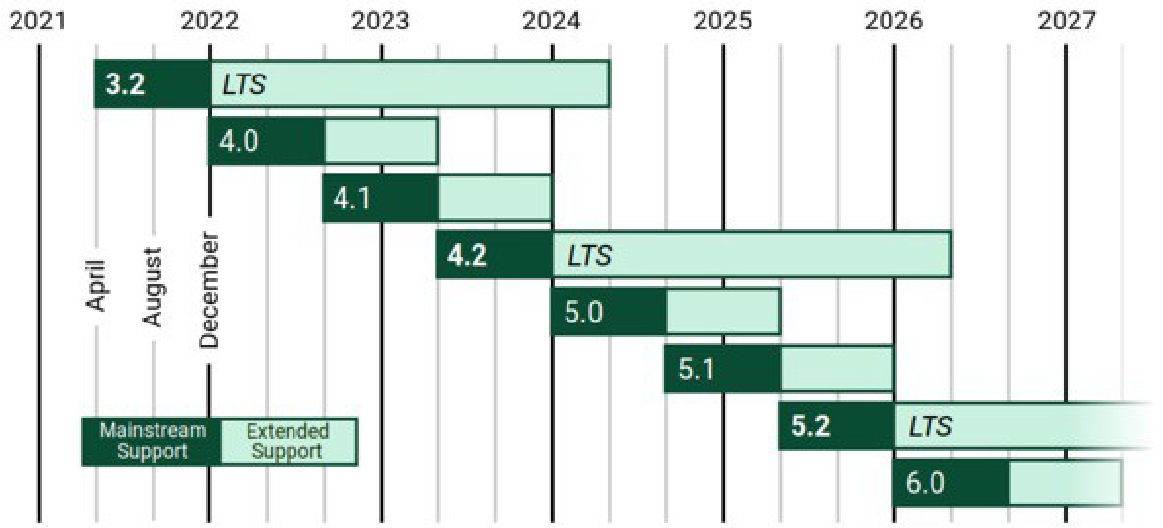


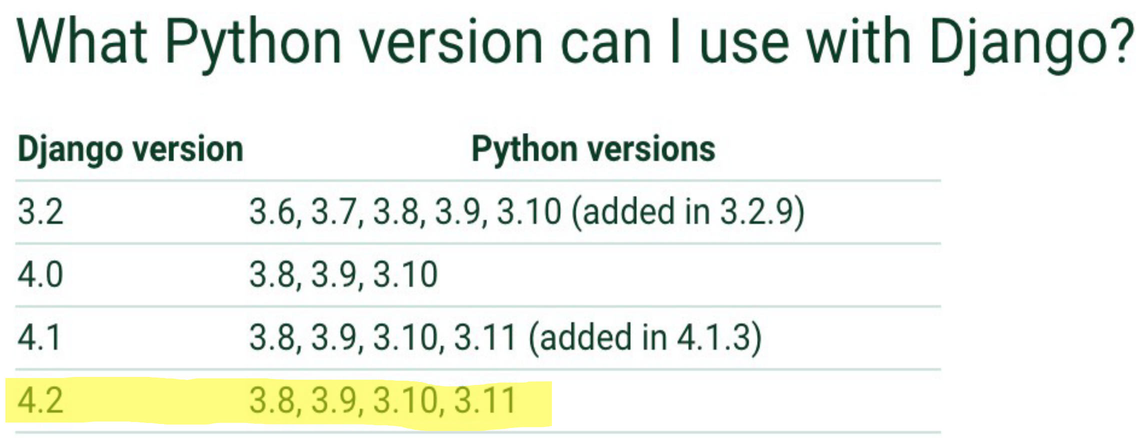
Django project & apps



Django
Preparing our development environment

Django and Python version





What Python version can I use with Django?

Django version	Python versions
3.2	3.6, 3.7, 3.8, 3.9, 3.10 (added in 3.2.9)
4.0	3.8, 3.9, 3.10
4.1	3.8, 3.9, 3.10, 3.11 (added in 4.1.3)
4.2	3.8, 3.9, 3.10, 3.11

Create our first Django project

1. Install Django using pip
2. Create a new project using django-admin startproject
3. Run the development server
4. Set-up VSCode for automatically running the server

Create our first Django project

1. **Install Django using pip**
2. Create a new project using django-admin startproject
3. Run the development server
4. Set-up VSCode for automatically running the server

Install Django using pip

In your activated virtual environment, use one of the following:

```
$(webprog) pip install django
```

Latest version of Django (any version)

```
$(webprog) pip install "django>=4.2,<5"
```

Latest version of Django 4.2

```
$(webprog) pip install "django==4.2.11"
```

Specific version of Django

Create our first Django project

1. Install Django using pip
2. **Create a new project using `django-admin startproject`**
3. Run the development server
4. Set-up VSCode for automatically running the server

Create a new Django project

```
$ django-admin startproject <project_name>
```

This will create a new folder with *project_name* and the following structure (if project name is **pr_league**)

- `pr_league`
    - **`manage.py`** This is the file you'll be running for most tasks
    - `pr_league`
        - `__init__.py`
        - **`settings.py`** Important configurations settings of our application
        - **`urls.py`** Table of contents of our web app (routes)
        - `wsgi.py`
        - `asgi.py`

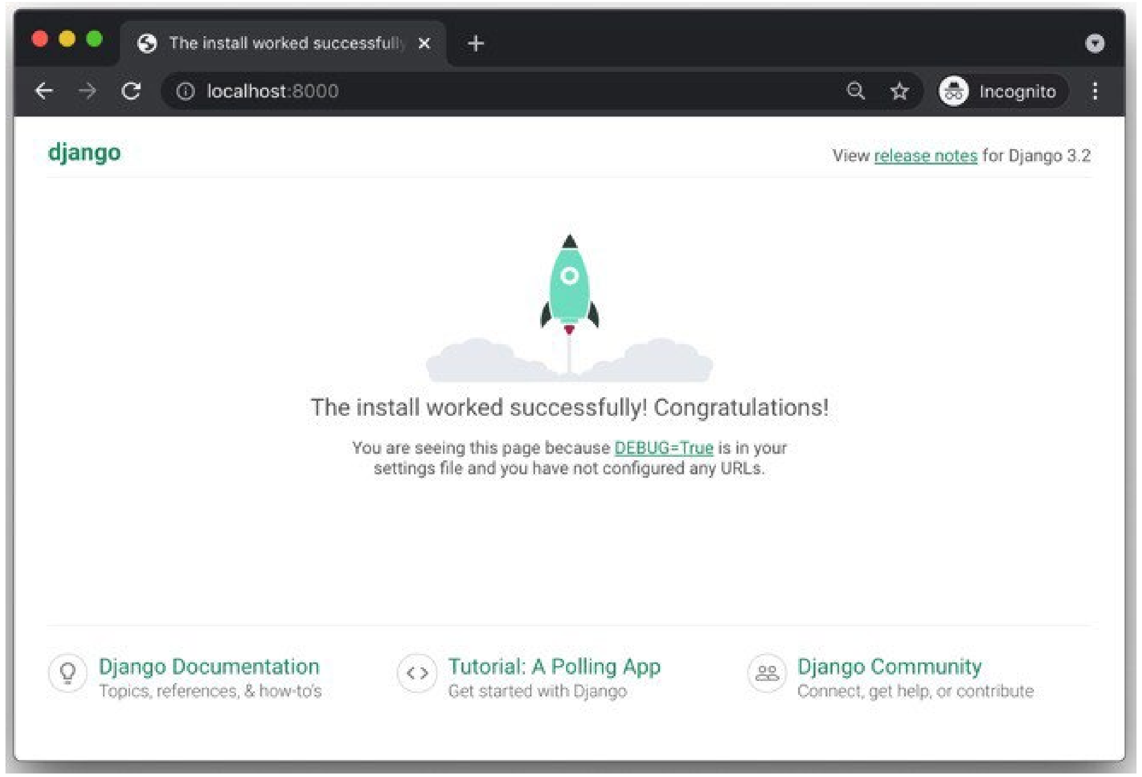
Create our first Django project

1. Install Django using pip
2. Create a new project using `django-admin startproject`
3. **Run the development server**
4. Set-up VSCode for automatically running the server

Run the development server

```
$ python manage.py runserver
```

Open http://127.0.0.1:8000/ or http://localhost:8000/ in your browser



django

View release notes for Django 3.2

The install worked successfully! Congratulations!

You are seeing this page because DEBUG=True is in your settings file and you have not configured any URLs.

Django Documentation
Topics, references, & how-to's

Tutorial: A Polling App
Get started with Django

Django Community
Connect, get help, or contribute

Create our first Django project

1. Install Django using pip
2. Create a new project using `django-admin startproject`
3. Run the development server
4. **Set-up VSCode for automatically running the server**

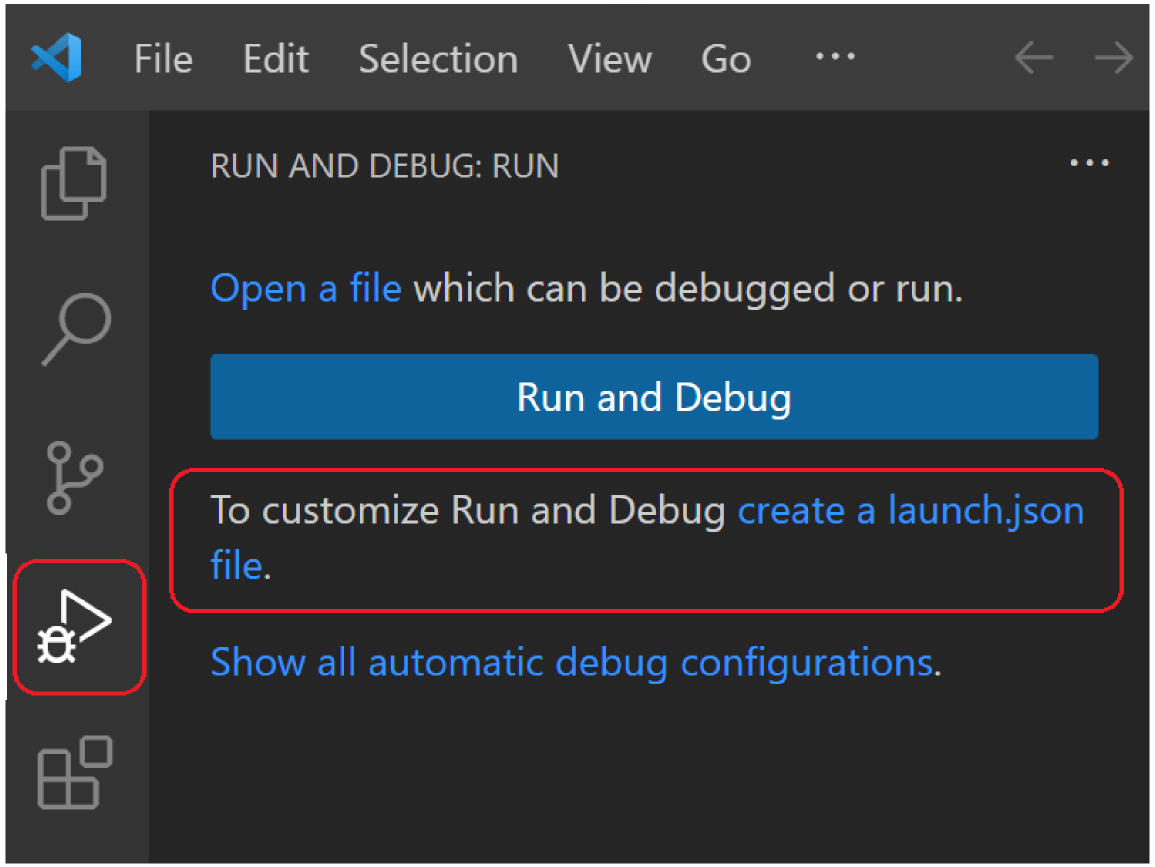
Run the server through VS Code

1. Switch to Run view in VS Code (using the left-side activity bar or F5)

   You may see the message "To customize Run and Debug create a `launch.json` file".

   This means that you don't yet have a `launch.json` file containing debug configurations.

   VS Code can create that for you if you click on the create a `launch.json` file link



Open a file which can be debugged or run.

To customize Run and Debug create a launch.json file.

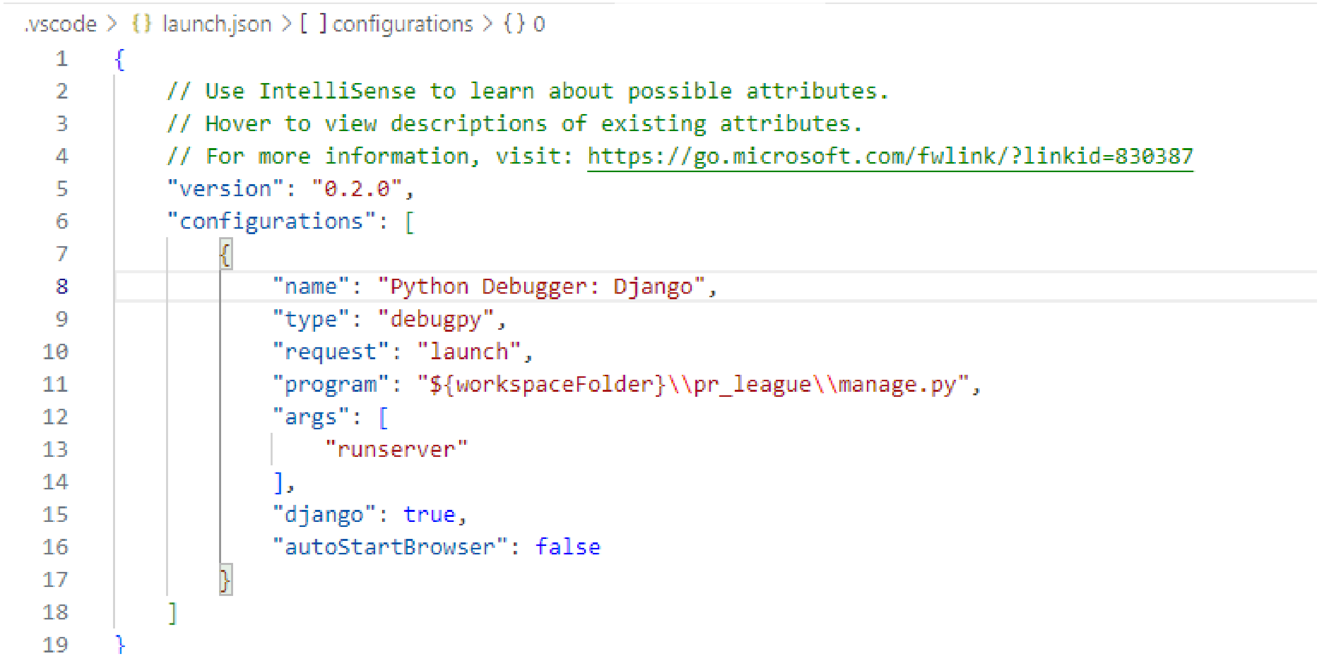
Show all automatic debug configurations.

https://code.visualstudio.com/docs/python/tutorial-django

Run the server through VS Code

2. Select the link and VS Code will prompt for a debug configuration.

   Select Django from the dropdown and VS Code will populate a new `launch.json` file with a Django run configuration.

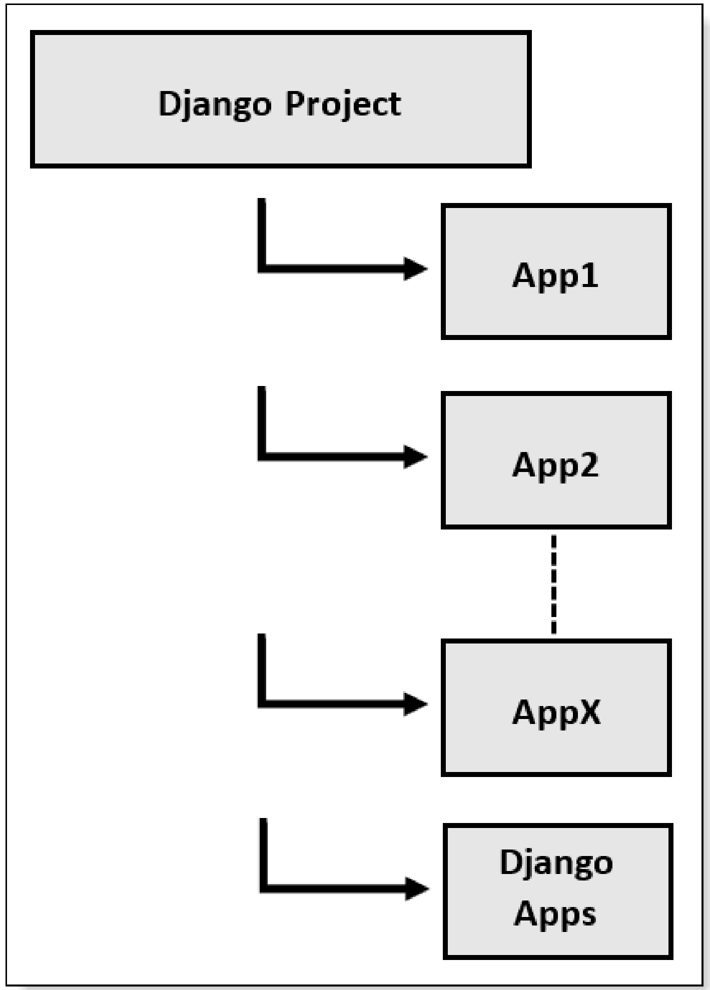


.vscode > {} launch.json > [ ] configurations > {} 0

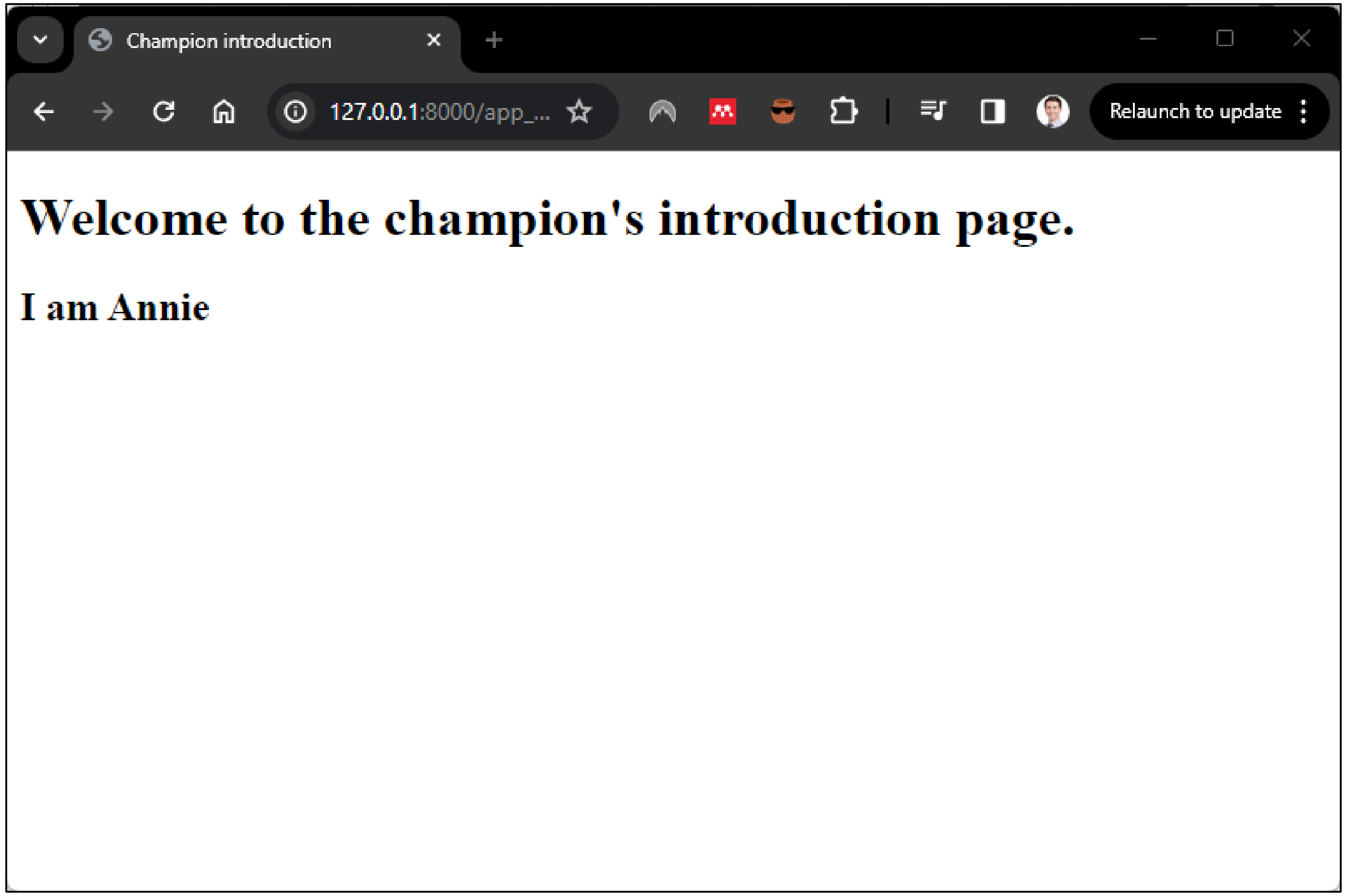
```
{
    // Use IntelliSense to learn about possible attributes.
    // Hover to view descriptions of existing attributes.
    // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
    "version": "0.2.0",
    "configurations": [
        {
            "name": "Python Debugger: Django",
            "type": "debugpy",
            "request": "launch",
            "program": "${workspaceFolder}\\pr_league\\manage.py",
            "args": [
                "runserver"
            ],
            "django": true,
            "autoStartBrowser": false
        }
    ]
}
```

https://code.visualstudio.com/docs/python/tutorial-django

Django project & apps



Our first simple app

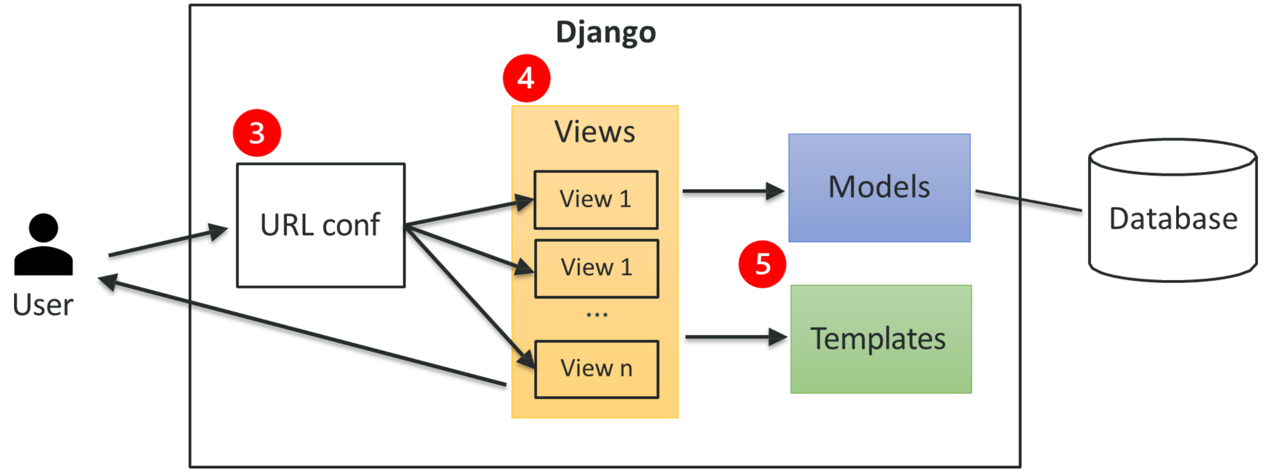


Welcome to the champion's introduction page.

I am Annie

Create a Django app

1. Create a new app using **`startapp`**
2. Add it to the **settings**
3. Create a **URL**
4. Link it to a **view**
5. Return a **template**
6. Make it dynamic with **context**
7. Add some **static** files



Create a Django app

1. **Create a new app using `startapp`**
2. Add it to the **settings**
3. Create a **URL**
4. Link it to a **view**
5. Return a **template**
6. Make it dynamic with **context**
7. Add some **static** files

Create a new app using startapp

```
$ manage.py startapp app_champs
```

or

```
$ django-admin startapp app_champs
```

- app_champs
  - __init__.py
  - admin.py
  - apps.py
  - models.py
  - tests.py
  - views.py
  - migrations
    - __init__.py

Create a Django app

1. Create a new app using **`startapp`**
2. **Add it to the settings**
3. Create a **URL**
4. Link it to a **view**
5. Return a **template**
6. Make it dynamic with **context**
7. Add some **static** files

Add it to the settings

- In **settings.py:**

```
INSTALLED_APPS = [
    'app_champs',
    'django.contrib.admin’,
    ...
]
```

- Django uses **INSTALLED_APPS** as a list of places to look for models, management commands, tests, and other utilities

Create a Django app

1. Create a new app using `startapp`
2. Add it to the settings
3. **Create a URL**
4. Link it to a **view**
5. Return a **template**
6. Make it dynamic with **context**
7. Add some **static** files

Create a URL – pt. I

- Create a new file called **`urls.py`**

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

app_name = 'app_champs'
urlpatterns = [
    path('', views.index, name='index'),
]
```

Create a URL – pt. 1

- Create a new file called **`urls.py`**

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

Relative import

```
app_name = 'app_champs'
urlpatterns = [
    path('', views.index, name='index'),
]
```

Create a URL – pt. 2

- Connect it to **`app_champs.urls.py`**

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

urlpatterns = [
    path('admin/', admin.site.urls),
    path('app_champs', include('app_champs.urls'))
    ]
```

Include also this urls from app_champs

Create a Django app

1. Create a new app using `startapp`
2. Add it to the **settings**
3. Create a **URL**
4. **Link it to a view**
5. Return a **template**
6. Make it dynamic with **context**
7. Add some **static** files

Link it to a view

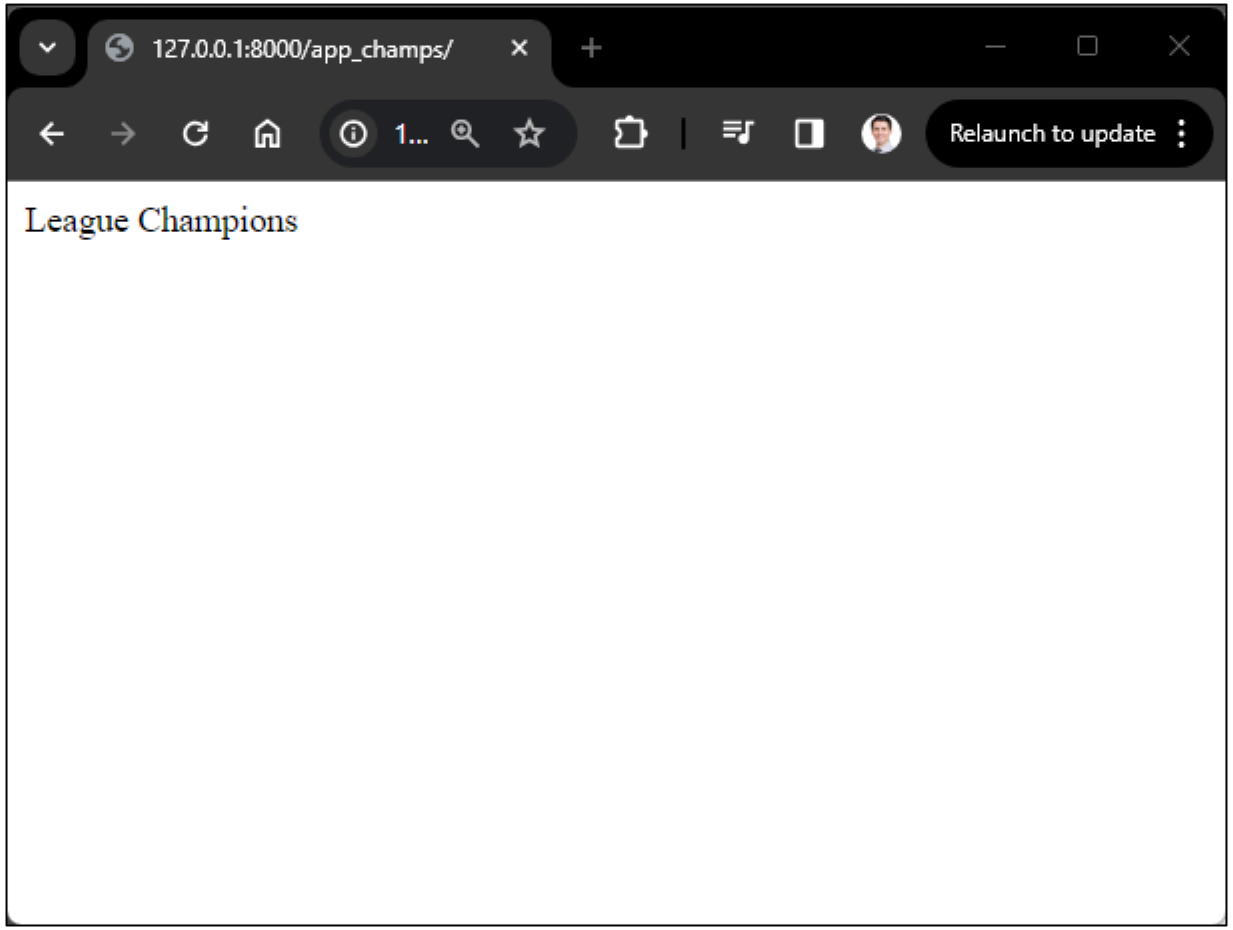
- In **`views.py`**

```
from django.http import HttpResponse


def index(request):
    return HttpResponse("League champions!")
```

Link it to a view

- We now have a simple index page



League Champions

Add extra views

- In **app_champs/views.py**

```
def leona(request):
     return HttpResponse("I am Leona!")

def annie(request):
     return HttpResponse("I am Annie")
```

- In **app_champs/urls.py**

```
urlpatterns = [
     …
     path('leona', views.leona, name="leona"),
     path(‘annie', views.annie, name="annie")
     ]
```

Create a Django app

1. Create a new app using `startapp`
2. Add it to the **settings**
3. Create a **URL**
4. Link it to a **view**
5. **Return a template**
6. Make it dynamic with **context**
7. Add some **static** files

Return a template – pt. I

- Create a new folder called **templates** in your **app_champs** folder
- Create a new folder called **app_champs** in your **templates** folder
- Create a new file in **templates** called **index.html**
- Add some html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Champion Page</title>
</head>
<body>
    <h1>Welcome to the champion's introduction page.</h1>
</body>
</html>
```

Return a template – pt. 2

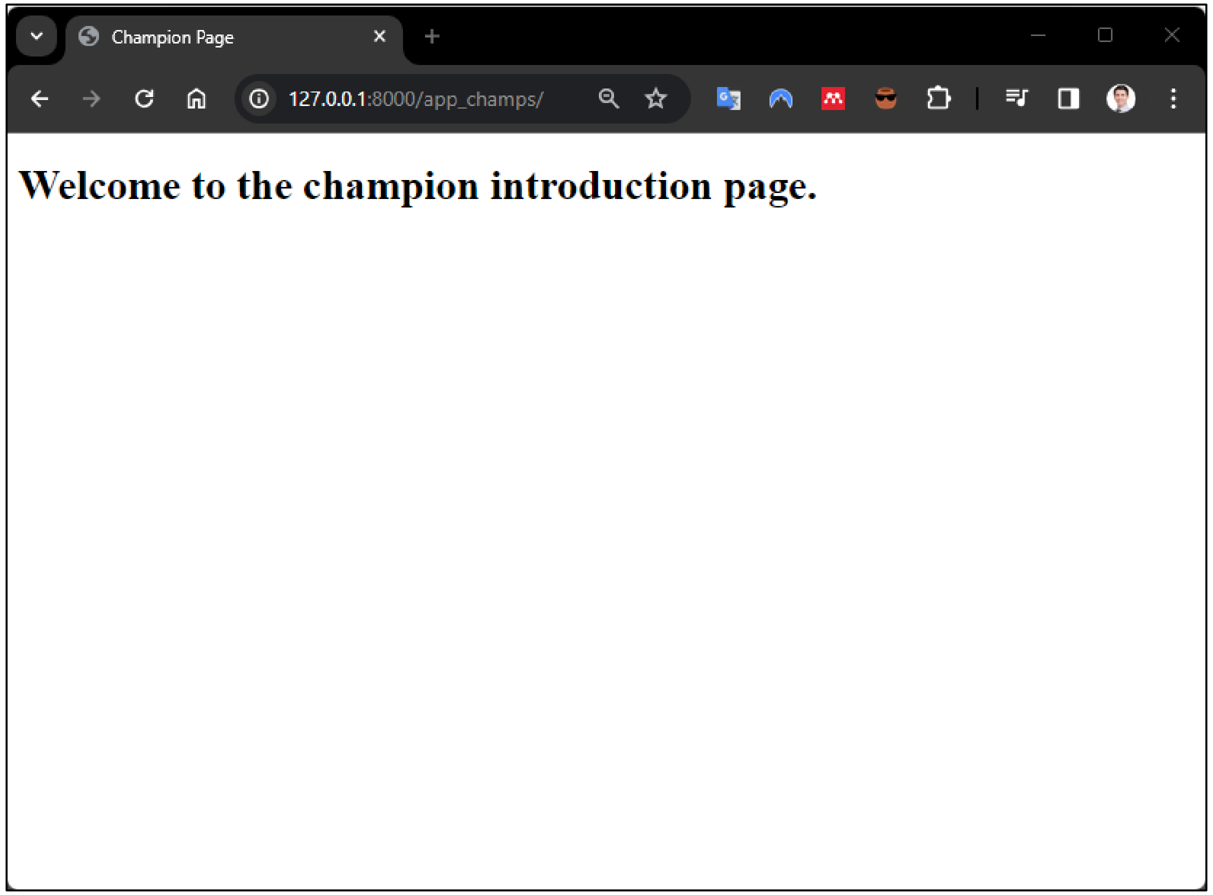
- Render the template from your view in **view.py**

```
from django.shortcuts import render

def introduce champion(request):
    return render(request, 'app_champs/index.html')
```

- Render is a django "shortcut" that:
  1. Retrieves the template
  2. Renders it to an HTML file
  3. Returns it as an HttpResponse

Return a template


Champion Page

127.0.0.1:8000/app_champs/

Welcome to the champion introduction page.

- This is just an HTML page until we start adding some dynamic content

Create a Django app

1. Create a new app using **`startapp`**
2. Add it to the **settings**
3. Create a **URL**
4. Link it to a **view**
5. Return a **template**
6. **Make it dynamic with context**
7. Add some **static** files

Make it dynamic – pt. I

- Send some data to the template from **views.py**

```
from django.shortcuts import render


def introduce_champion(request, name):
    context = {"name":name.capitalize()}
    return render(request, 'app_champs/index.html', context=context)
```

- Context is a dictionary that gets passed to the template
- The template can then use that data when rendering to HTML

Make it dynamic – pt. 2

- Create a **introduce.html** template to use that data

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Champion introduction</title>
</head>
<body>
    <h1>Welcome to the champion's introduction page.</h1>
    <h2>I am {{name}}</h2>
</body>
</html>
```

Make it dynamic – pt. 2

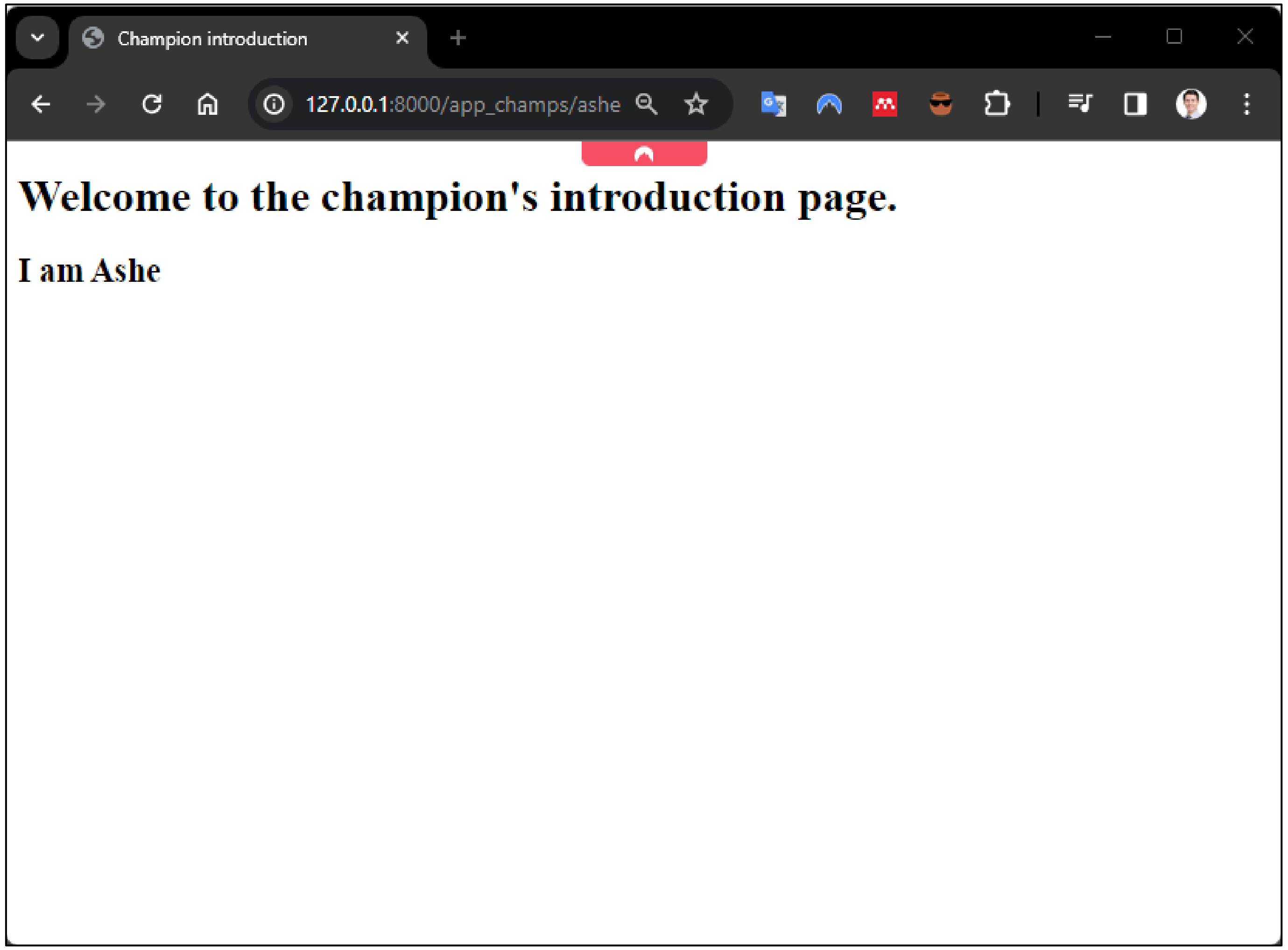
The Django Template Language

- Provides tags which function similarly to some programming constructs – an if tag for boolean tests, a for tag for looping, etc
- A template contains variables, which get replaced with values when the template is evaluated, and tags, which control the logic of the template.
- The **{{ var }}** syntax is a **variable** that gets evaluated (from the context) and is replaced when the template is rendered

https://docs.djangoproject.com/en/4.2/ref/templates/builtins/#ref-templates-builtins-filters

https://docs.djangoproject.com/en/4.2/ref/templates/language/

Make it dynamic



127.0.0.1:8000/app_champs/ashe

Welcome to the champion's introduction page.

I am Ashe

Create a Django app

1. Create a new app using `startapp`
2. Add it to the **settings**
3. Create a **URL**
4. Link it to a **view**
5. Return a **template**
6. Make it dynamic with **context**
7. **Add some static files**

Add some static files – pt. I

- Create a new folder called **static** in your **app_champs** folder
- Create a new folder called **app_champs** in your **static** folder
- Create a file style.css and add some style
- Copy some image to the **static/app_champs** folder

```
h1{
    color: blue;
    font-size: 48px;
    font-family: 'Courier New', Courier, monospace;
}
```

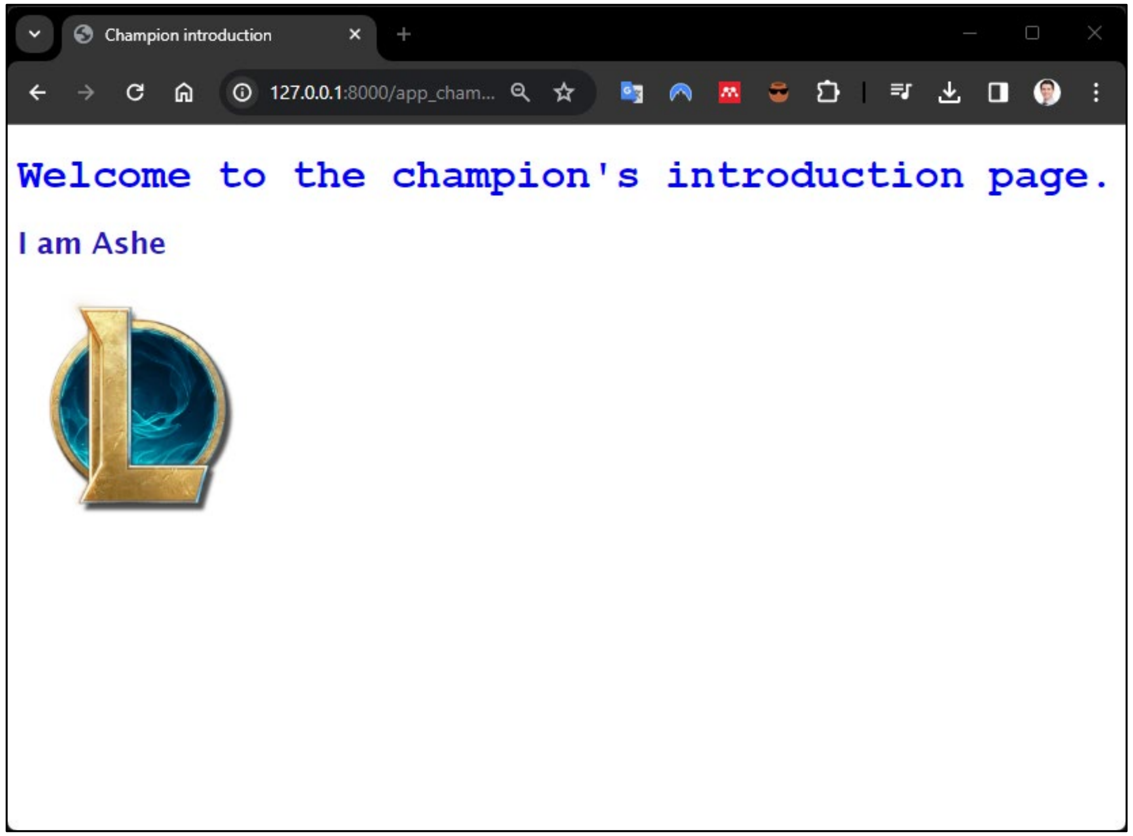
Add some static files – pt. 2

- Update your **introduce.html**

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="{% static 'app_champs/style.css' %}">
    <title>Champion introduction</title>
</head>
<body>
    <h1>Welcome to the champion's introduction page.</h1>
    <h2>I am {{name}}</h2>
    <img src="{% static 'app_champs/lol-logo.png' %}" width="200" alt="logo">
</body>
</html>
```

https://docs.djangoproject.com/en/4.2/ref/templates/builtins/#static

Add some static files



Welcome to the champion's introduction page.

I am Ashe

- **Restart** the development server and **refresh** your browser
- You should now see slightly nicer style and a logo in the body of the page

Web Programming