

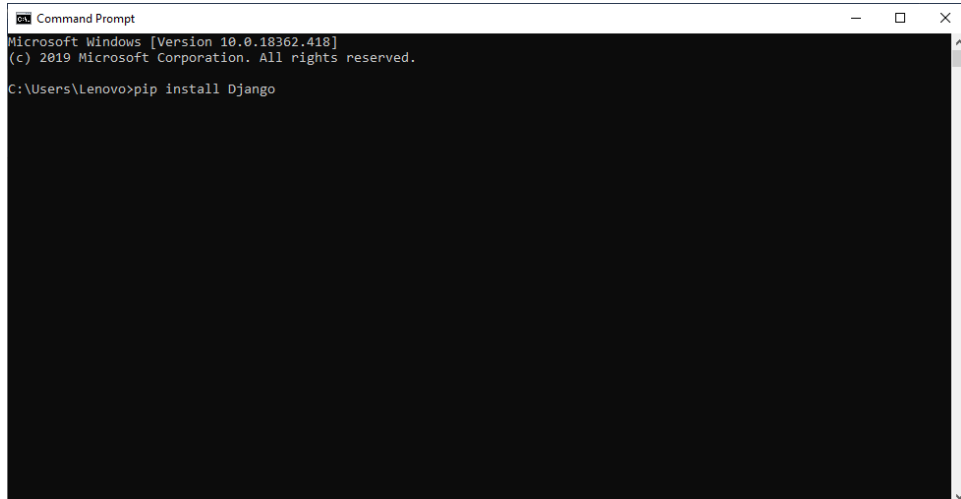
TUTORIAL DJANGO

Instalando Django

*Para la instalación de Django es necesario que tenga instalado **Python** y **pip**.

En la consola de comando (CMD) escribimos el comando:

pip install Django



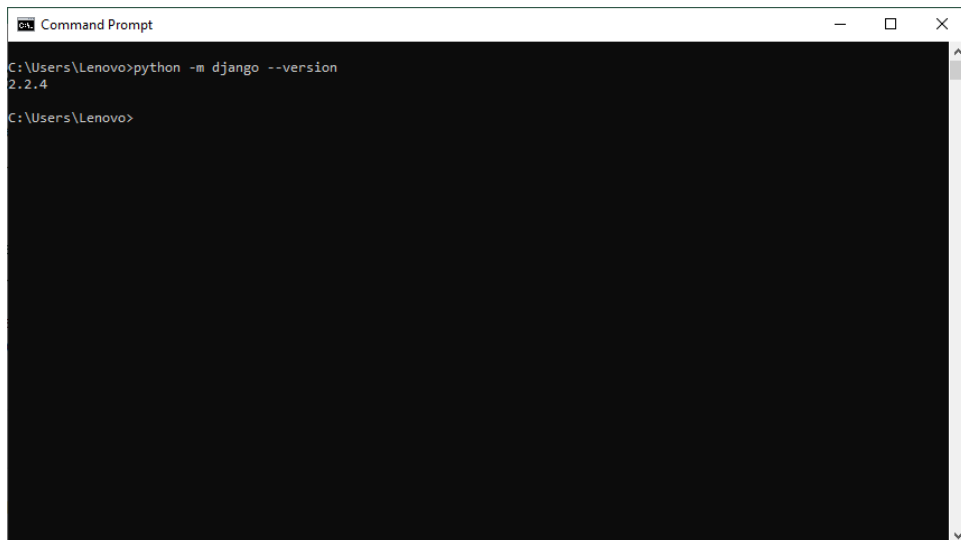
```
Command Prompt
Microsoft Windows [Version 10.0.18362.418]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Lenovo>pip install Django
```

Para verificar la instalación correcta de Django, vemos su versión con el comando:

python -m django --version

Si está correctamente instalada, se muestra la versión, de caso contrario, saldrá un error de que no existe el módulo Django.



```
Command Prompt

C:\Users\Lenovo>python -m django --version
2.2.4

C:\Users\Lenovo>
```

Creando un proyecto

Primero nos dirigimos hacia el directorio donde queramos poner nuestro proyecto, una vez ya ubicados, para la creación de un proyecto con Django, en el CMD se escribe el comando:

django-admin startproject sitioweb

*No poner nombres al proyecto de componentes o módulos reservados por Python o Django, por ejemplo: **django**, **python**, **test**, etc., ya que se puede generar conflictos.

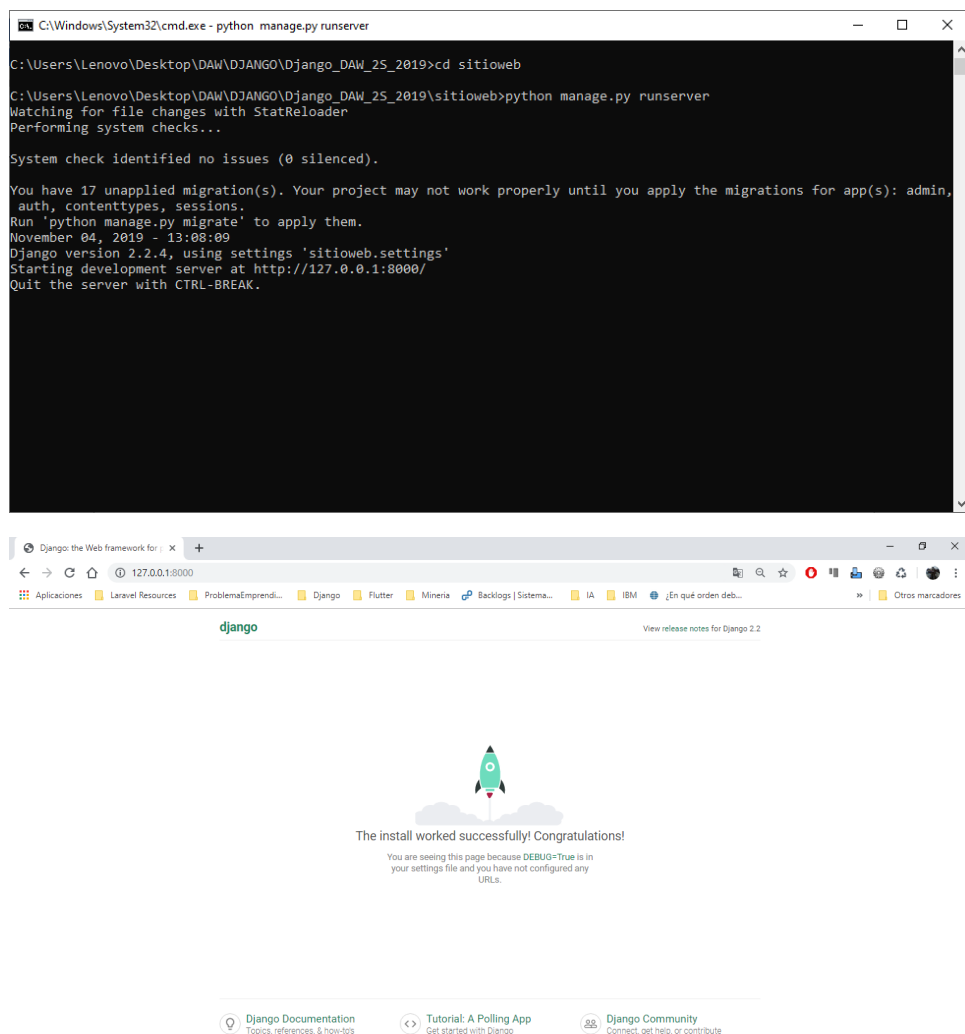
Una vez ejecutado el comando, se nos crea un directorio raíz llamado **sitioweb**, el cual posee un archivo **manage.py** y un subdirectorio llamado **sitioweb**.



Ahora probaremos a iniciar el servidor de Django de manera local, ubicarse en el directorio raíz **sitioweb**, y escribimos el siguiente comando en el CMD:

python manage.py runserver

Nos saldrá una salida como la siguiente, en la que nos indica que la aplicación se encuentra iniciada en **127.0.0.1** con un puerto por defecto, el **8000**. Posteriormente, debemos copiar eso en la barra de búsqueda del navegador y esperemos a ver lo que carga.



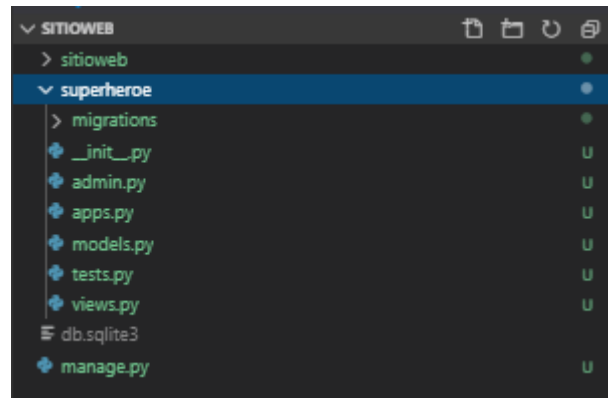
Creando la aplicación de Superheroes

Una vez que ya tenemos nuestro proyecto, vamos a crear nuestra primera aplicación de **superheroe**, que consumirá un **servicio Rest**. Para crear la aplicación en el CMD escribimos el comando siguiente:

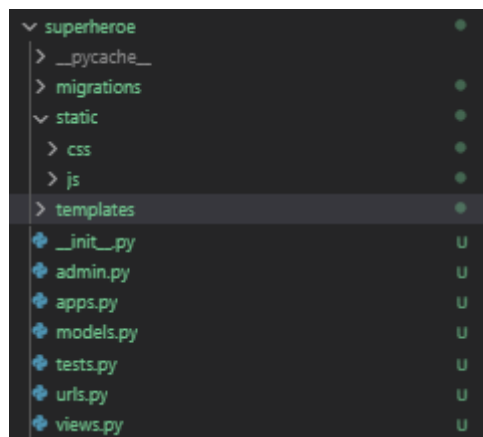
python manage.py startapp superheroe

* ¿Cuál es la diferencia entre un proyecto y una aplicación? Una app es una aplicación web que hace algo, por ejemplo, un sistema de blog, una base de datos de registros públicos o una aplicación de encuesta simple. Un proyecto es un conjunto de configuraciones y aplicaciones para un sitio web determinado. Un proyecto puede contener aplicaciones múltiples. Una aplicación puede estar en varios proyectos.

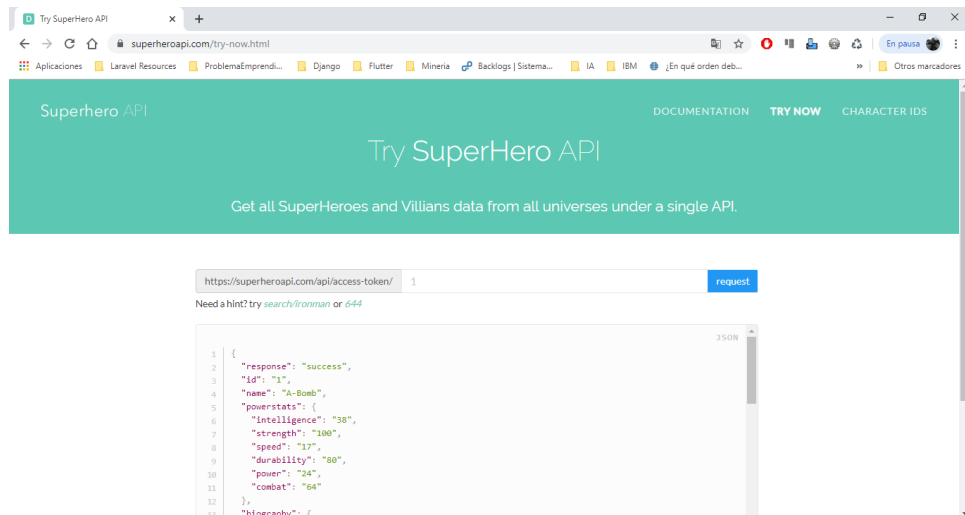
Veremos que se nos crea otro directorio llamado **superheroe** en la raíz **sitioweb**.



Vamos a crear dos carpetas nuevas dentro de **superheroe**, una llamada **templates**, en la cual estarán los archivos **html** para las vistas; y otra llamada **static**, en la que se ubicaran los archivos estáticos en dos subcarpetas adicionales, una llamada **css**, donde estaran las hojas de estilos **css** y otra llamada **js** donde se encontraran archivos scripts **js**; quedando como la imagen siguiente.



En nuestra aplicación consumiremos un servicio rest que contiene información de varios superheroes, en donde su **endpoint** nos devuelve una respuesta en formato **JSON** con información que cargaremos en nuestra página web.



Primero debemos crear nuestra ruta para acceder a la página web, para aquello, en nuestra aplicación **superhero** creamos un archivo llamado **urls.py**, en el cual tendrá lo siguiente.

```
superhero > urls.py > ...
1  from django.urls import path
2
3  from . import views
4
5  urlpatterns = [
6      path('', views.index, name='index'),
7      path('details/<int:id>', views.details, name='details'),
8  ]
```

Aquí registramos las rutas, la primera **url** será nuestra url raíz, por lo que se pone solo comillas vacías, mientras que la otra url será **details/<int:id>**, luego del slash (/) recibe un parámetro, en este caso un **id** de un superhéroe para obtener su información con el servicio rest. **path** está recibiendo 3 parámetros, el primero es la url, el segundo es la función a la cual la url irá para hacer la lógica y el tercer parámetro es un nombre que se le asigna a la ruta para identificar la función de la vista en general.

Como estamos trabajando en la aplicación **superhero** y nuestro proyecto **sitioweb** puede tener varias aplicaciones, debemos registrar la ruta para cada aplicación creada en el archivo **urls.py** del **sitioweb**.

```
sitioweb > urls.py > ...
1  """sitioweb URL Configuration
2
3  The `urlpatterns` list routes URLs to views. For more information please see:
4      https://docs.djangoproject.com/en/2.2/topics/http/urls/
5  Examples:
6  Function views
7      1. Add an import: from my_app import views
8      2. Add a URL to urlpatterns: path('', views.home, name='home')
9  Class-based views
10     1. Add an import: from other_app.views import Home
11     2. Add a URL to urlpatterns: path('', Home.as_view(), name='home')
12  Including another URLconf
13     1. Import the include() function: from django.urls import include, path
14     2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))
15  """
16  from django.contrib import admin
17  from django.urls import include, path
18
19  urlpatterns = [
20      path('admin/', admin.site.urls),
21      path('superhero/', include('superhero.urls')),
22  ]
23
```

Aquí el **include()** incluye a la url **superhero/** todas las urls que se ingresaron para esa aplicación, por lo tanto para nuestras urls al final en el navegador serán estas:

localhost:8000/superhero/

localhost:8000/superhero/details/10

Una vez creado nuestras rutas, debemos ahora crear nuestras funciones en **views** para manejar la lógica y retornar una vista. Primero nos ubicamos en el archivo **views.py** de nuestra aplicación **superhero** y haremos lo siguiente.

```
superhero > views.py > ...
1  from django.shortcuts import render
2  import requests
3
4  # Create your views here.
5  def index(request):
6
7      list_id = ['620', '644', '70', '346', '149', '659', '332', '720', '157']
8      superheroes = []
9
10     for id in list_id:
11         response = requests.get('https://superheroapi.com/api/2542779899290396/%s/image' % id)
12         if response.status_code == 200:
13             superhero = response.json()
14             superheroes.append(superhero)
15
16     #print(superheroes)
17     return render(request, 'index.html', {
18         'superheroes': superheroes,
19     })
20
21  def details(request, id):
22      response = requests.get('https://superheroapi.com/api/2542779899290396/%s' % id)
23      if response.status_code == 200:
24          superhero = response.json()
25          return render(request, 'details.html', {
26              'superhero': superhero,
27          })
28      return None
```

Hay que tener en cuenta que debemos tener instalado el módulo **requests** de Python, en caso de que no se lo tenga, se instala con el siguiente comando:

pip install requests

Cada función que creamos en **views.py** tiene como primer parámetro un objeto **HttpRequest**, que normalmente de lo denomina **request**. En la función **index** primero tenemos una lista de ids (**list_id**), luego iteramos esa lista, y en cada iteración hacemos un requerimiento al servicio usando **requests.get()** al **endpoint** <https://superheroapi.com/api/2542779899290396/1/image>, y si obtenemos una respuesta válida (**status 200**), la agregamos a la lista **superheroes**. Cada respuesta válida se ve de la siguiente manera en formato **JSON**.

```
1 {
2   "response": "success",
3   "id": "70",
4   "name": "Batman",
5   "url": "https://www.superherodb.com/pictures2/portraits/10/100/639.jpg"
6 }
```

Luego en el **return** hacemos un **render**, el cual nos renderiza nuestra vista **index.html** en el navegador. **render** recibe en este caso, **request** (objeto **HttpRequest**), el nombre de nuestro archivo **html**, y como tercer parámetro un **diccionario**, con clave-valor, donde la clave es **superheroes** y como valor la lista denominada **superheroes**.

En la función **details**, a parte del **request**, recibimos un parámetro adicional **id**, porque cuando definimos nuestra **url**, esta recibía aquel **id** como parámetro, hacemos nuestro requerimiento al **endpoint** <https://superheroapi.com/api/2542779899290396/1> y luego dependiendo de que, si obtenemos una respuesta válida, renderizamos ahora nuestro archivo **details.html**, con un **diccionario** que contiene nuestra respuesta **JSON** del servicio.

```
1 {
2   "response": "success",
3   "id": "70",
4   "name": "Batman",
5   "powerstats": {
6     "intelligence": "100",
7     "strength": "26",
8     "speed": "27",
9     "durability": "50",
10    "power": "47",
11    "combat": "100"
12  },
13   "biography": {
14     "full-name": "Bruce Wayne",
15     "alter-egos": "No alter egos found.",
16     "species": "Human"
17   }
18 }
```

Lo último que nos toca crear es nuestros archivos **html** que son utilizados en el **render** por las funciones en **views**. Para este caso se usaron plantillas de **bootstrap** ya hechas. Por lo tanto, debemos colocar los **.css** y **.js** en nuestra carpeta **static**, y nuestros **html** en nuestra carpeta **templates**. A continuación, se muestra la parte más importante en los **html** que utilizan el **motor de plantillas** propio de **Django**.

```

33 <div class="album py-5 bg-light">
34 <div class="container">
35 <div class="row">
36   {% for superhero in superheroes %}
37   <div class="col-md-4">
38     <div class="card mb-4 shadow-sm">
39       
40       <div class="card-body">
41         <h4 class="card-text">{{ superhero.name }}</h4>
42         <div class="d-flex justify-content-between align-items-center">
43           <a href="{% url 'details' superhero.id %}" class="btn btn-sm btn-outline-secondary">Ver más</a>
44         </div>
45       </div>
46     </div>
47   </div>
48   {% endfor %}
49 </div>
50 </div>
51 </div>
52

```

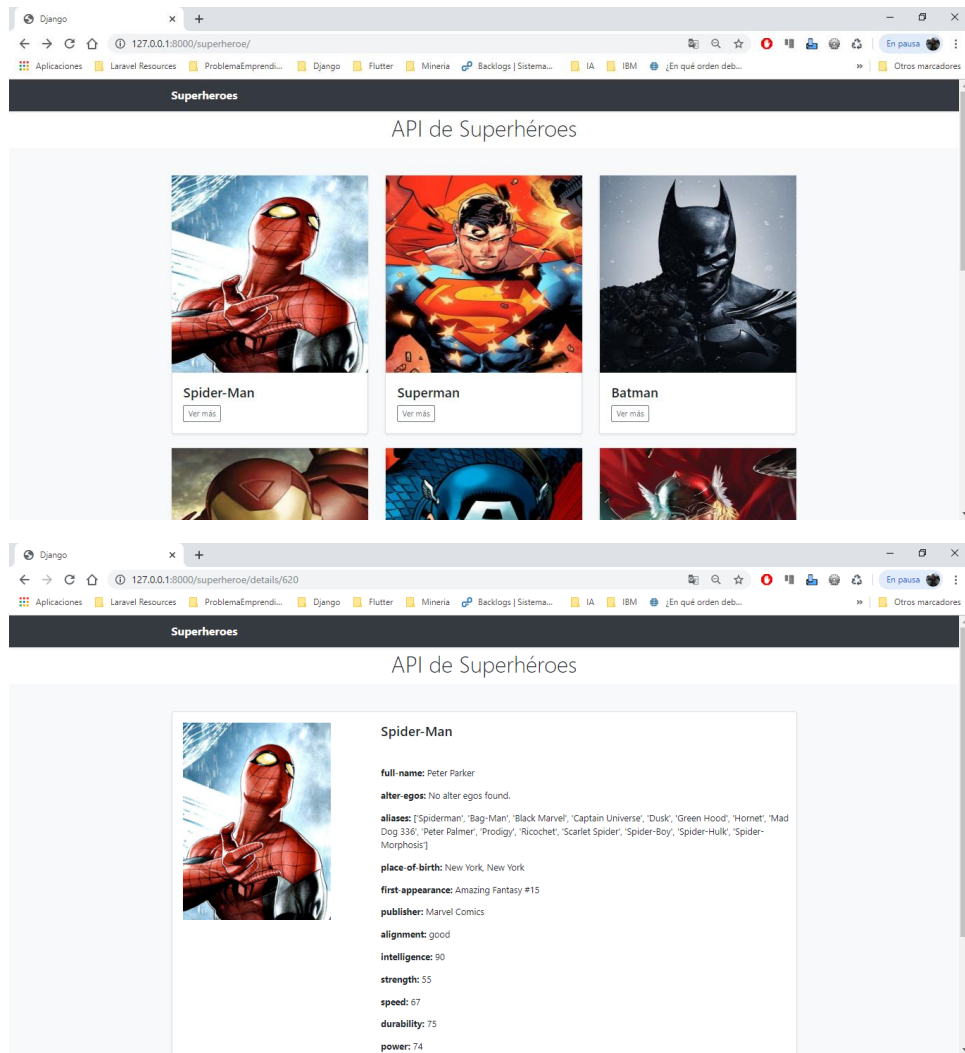
Esta sección es de nuestro archivo **index.html**, como se ve, se está haciendo una iteración **for** de nuestra lista **superheroes** que enviamos en el **render**, y por cada **superhero**, se van creando el contenedor **div** con una imagen (**img**) y un enlace (**a**), el cual, en su **href**, se pone el nombre del **url details** que creamos, junto con **superhero.id** que contiene el valor del id del superhéroe para el **servicio rest**. Por lo tanto, al dar clic sobre ese enlace, nos llevaría a la url **/details/id** y nos redireccionaría a esa ruta con otra vista.

```

<div class="card-body">
  <div class="row">
    <div class="col-md-4">
      
    </div>
    <div class="col-md-8">
      <div class="row">
        <h4 class="card-text">{{ superhero.name }}</h4>
      </div>
      <br><br>
      <div class="row">
        <col class="col-md-12">
          <div>
            {% for key, value in superhero.biography.items %}
            <p><strong>{{ key }}: </strong><span>{{ value }}</span></p>
            {% endfor %}
          </div>
          <div>
            {% for key, value in superhero.powerstats.items %}
            <p><strong>{{ key }}: </strong><span>{{ value }}</span></p>
            {% endfor %}
          </div>
        </col>
      </div>
    </div>
  </div>
</div>

```

En esta vista, ya solo tenemos un superhéroe, el cual viene a hacer un **JSON** y los manejamos como **diccionario**. En **img** colocamos la imagen que nos da el servicio y abajo, recorremos los diccionarios que contienen más información detalla del superhéroe. Al final, en el navegador tendremos las siguientes páginas.



Repositorio en Git

https://github.com/emiliomorán/Django_DAW_2S_2019