Django y Dashboards

November 23, 2020

1 Introducción a Django

En primer lugar vamos a ver cómo funciona Django. Para ello he creado un pequeño proyecto llamado "tango with django project". En primer lugar, veamos los requisitos que necesitamos para poder realizar un proyecto en Django. Los podemos encontrar en el fichero "requirements.txt" pero básicamente son estos:

- coverage
- dj-static
- django
- gunicorn
- mccabe
- pillow
- psycopg2
- pycodestyle
- pyflakes
- pytz

Aparecen más en el fichero pero son para detalles más avanzados, como bases de datos y registros.
Para comenzar un proyecto en Django basta con ir a la carpeta donde queramos guardar todo y ejecutar el siguiente comando:
django-admin.py startproject ¡nombre del proyecto¿

donde ¡nombre del proyecto¿ en este caso sería tango with django proyect. A continuación, tendremos que crear una app, para que el servidor pueda llenarse de templates y bases de datos. En este caso la app se llama "rango" y se crea con el siguiente comando:

python manage.py startapp rango
De este modo, tendremos los siguientes contenidos en la carpeta:

- Carpeta rango: contiene lo relacionado con la app rango que veremos más adelante.
- Carpeta static: contiene los ficheros relacionados con las imágenes, el código javascript y el CSS que queramos introducir en el proyecto.
- Carpeta templates: contiene los html del servidor.
- Carpeta tango with django proyect: contiene los ajustes del servidor.
- Fichero manage.py: sirve para ejecutar el servidor con el comando "python manage.py runserver"
- Otros ficheros que son parte de Django y no deberíamos tocar.

1.1 Carpeta tango with django project

Dentro de este directorio podemos encontrar cuatro ficheros pero sólo hablaremos de dos pues los demás son creados por Django y no podemos modificarlos. En primer lugar hablaremos de settings.py. Este fichero se encarga de registrar y decirle a Django dónde está situado cada fichero, cómo queremos las bases de datos... Pero lo único que hay que saber es que hay que añadir la app creada al array llamado INSTALLED APPS. En este caso añadiremos la app "rango":

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rango',
]
```

Por otro lado, tenemos el fichero "urls.py". Éste contiene las urls que ponemos en el navegador al buscar un servidor. Como tenemos la app creada, sólo necesitamos decirle dónde se almacena nuestra app, es decir, en la carpeta "rango". Aquí podemos observarlo:

```
urlpatterns = [
    path('', views.index, name='index'),
    path('rango/', include('rango.urls')),
    path('admin/', admin.site.urls),
]

urlpatterns += static(settings.STATIC_URL,
                      document_root=settings.STATIC_ROOT)
```

También podemos observar que se añade el path a la carpeta "static", para poder utilizarla desde los html de la app rango.

1.2 Carpeta rango

En esta carpeta podemos encontrar 7 ficheros. En primer lugar, init.py es creado por Django luego no podemos modificarlo. Por otro lugar no explicaremos forms.py pues tiene que ver con bases de datos. Luego nos quedan los siguientes ficheros:

- Models.py: contiene los "modelos" que queremos introducir en la app, es decir, las clases que tendrá. Para este ejemplo sólo he utilizado una clase, la del usuario que se conecta. Esta clase dispone de nombre de usuario, contraseña y foto de perfil tal y como podemos observar en la foto:

```python
class UserProfile(models.Model):
    # This line is required. Links UserProfile to a User model instance.
    user = models.OneToOneField(User, on_delete=models.CASCADE)

    # The additional attributes we wish to include.
    website = models.URLField(blank=True)
    picture = models.ImageField(upload_to='profile_images', blank=True)

    def __str__(self):
        return self.user.username
```

- Apps.py: esta carpeta solamente registra el nombre de la app:

```python
class RangoConfig(AppConfig):
    name = 'rango'
```

- Urls.py: igual que en la sección anterior, este fichero registra la url a cada pestaña de nuestra app, es decir, a cada html que hayamos creado y que esté contenido en la carpeta "templates". En este proyecto he creado varias pestañas: una para que el usuario se registre, otra para hacer login, una por cada gráfico descrito... Además, es muy importante siempre tener un path al index, pues es el directorio raíz, el primero que podemos observar aquí:

```python
app_name = 'rango'

urlpatterns = [
    path('', views.index, name='index'),
    path('areachart/', views.areachart, name='areachart'),
    path('donutchart/', views.donutchart, name='donutchart'),
    path('barchart/', views.barchart, name='barchart'),
    path('mixedchart/', views.mixedchart, name='mixedchart'),
    path('table/', views.table, name='table'),
    path('register/', views.register, name='register'),
    path('login/', views.user_login, name='login'),
    path('logout/', views.user_logout, name='logout'),
]
```

- Views.py: este fichero se encarga de enviarle a Django los templates, es decir, utiliza urls.py para comunicarse con Django y mandarle con la función request el html. Con otras palabras, es el puente entre los templates html

y urls.py.
Es muy importante que tengamos una función por cada html. Es decir, si tenemos dos paths en urls.py necesitamos al menos dos funciones con el mismo nombre en views.py. Por ejemplo, en urls.py tenemos un path a "areachart" del mismo modo que en views.py tenemos una función llamada "areachart" que devuelve el fichero "areachart.html" que se encuentra en la carpeta "templates":

```
def areachart(request):
    return render(request, 'rango/areachart.html')
```

Podemos realizar varias opciones desde estas funciones, como modificar datos a través de los html. Pero esto ya es más avanzado.

1.3 Carpeta Templates

La carpeta Templates contiene a su vez una carpeta llamada "Rango", esto se debe a que son todos los templates contenidos en nuestra app, los cuales hemos registrado en el fichero "urls.py" y los cuales son devueltos por el fichero "views.py".
IMPORTANTE: hay que tener una url y una view por cada html que tengamos.

1.4 Carpeta Static

En este directorio encontraremos el archivo CSS para darle estilo al HTML, la carpeta imagenes, para añadir fotos al HTML, y la carpeta js, donde encontraremos los ficheros javascript para realizar los dashboards.

2 Dashboards

En esta sección vamos a ver cómo crear un Dashboard con tablas de datos y gráficos. En primer lugar necesitamos los HTML para darle un lugar a las gráficas en la página web y en segundo lugar necesitamos los ficheros javascript para representar estos datos.
Comenzaremos con un poco de HTML en Django. Si abrimos la carpeta Templates/Rango, observaremos que hay varios ficheros:

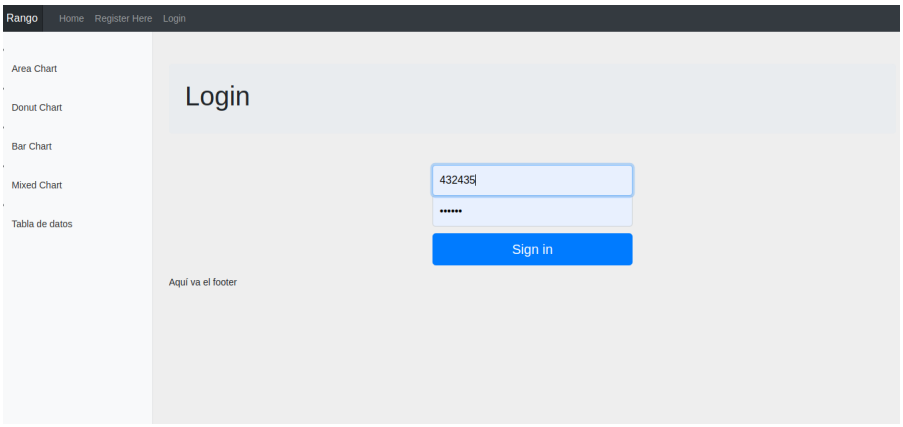
- Base.html: este es el fichero base, en el definimos los links, los scripts... y le damos un cuerpo a nuestra aplicación:

Esta fotografía corresponde al index.html. Podemos observar que encontramos una barra de navegación arriba que cuenta con Home, Login y Register y una barra de navegación a la izquierda que cuenta con los links a las páginas de cada chart. Entonces base.html se encarga de hacer de base añadiendo todas las opciones descritas mientras que los demás ficheros html solamente se encargan de darle forma a la parte de la derecha, dejando siempre estáticas las barras de navegación. Esto lo conseguimos gracias a Django.
Para ello solo tenemos que añadir en base.html lo siguiente:

```
<!-- AQUI VA EL CONTENIDO DE CADA PAGINA -->
{% block body_block %}

{% endblock %}
```

Y al resto de los ficheros html solo habrá que añadirles el html que queramos introducir justo en esa parte del programa, por ejemplo, en login.html vemos que solo hemos añadido el html correspondiente a ello, mientras que si clickamos en login aparecerán también las barras de navegación:

```
{% block body_block %}
<link href="https://getbootstrap.com/docs/4.0/examples/signin/signin.css" rel="stylesheet">
<div class="jumbotron p-4">
  <h1 class="jumbotron-heading">Login</h1>
</div>
<div class="container">
  <form class="form-signin" role="form" method="post" action=".">
    {% csrf_token %}
    <label for="inputUsername" class="sr-only">Username</label>
    <input type="text" name="username" id="id_username" class="form-control" placeholder="Username" required autofocus>
    <label for="inputPassword" class="sr-only">Password</label>
    <input type="password" name="password" id="id_password" class="form-control" placeholder="Password" required>
    <button class="btn btn-lg btn-primary btn-block" type="submit" value="Submit">Sign in</button>
  </form>
</div>
{% endblock %}
```



Por último, base.html va a ser donde importemos los ficheros de estilo CSS y los ficheros Javascript que queramos usar en nuestro proyecto. Para ello podemos observar que dentro de la etiqueta ¡head¿ tenemos los estilos importados con la etiqueta ¡link¿:

```
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <meta name="description" content="">
  <!-- Icono de la pagina -->
  <link rel="icon" href="{% static 'images/favicon.ico' %}">
  <title>
    Dashboards - {% block title_block %} {% endblock %}
  </title>
  <link href="../../static/css/estilo.css" rel="stylesheet">
  <!-- Añadimos Bootstrap core CSS -->
  <link href="https://getbootstrap.com/docs/4.2/dist/css/bootstrap.min.css" rel="stylesheet">
  <!-- Añadimos el dashboard -->
  <link href="https://getbootstrap.com/docs/4.2/examples/dashboard/dashboard.css" rel="stylesheet">
  <link rel="stylesheet" type="text/css" href="https://cdn.datatables.net/1.10.22/css/jquery.dataTables.css">
</head>
```

mientras que al final del todo, antes de cerrar la etiqueta ¡body¿ tenemos los scripts de javascript importados con la etiqueta ¡script¿:

```
  <!-- Bootstrap core JavaScript -->
  <!-- Placed at the end of the document so the pages load faster -->
  <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"></script>
  <script>
    window.jQuery || document.write('<script src="https://getbootstrap.com/docs/4.2/assets/js/vendor/jquery-slim.min.js"><\/script>')
  </script>
  <script src="https://getbootstrap.com/docs/4.2/dist/js/bootstrap.bundle.min.js">
  </script>


  <script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.5.0/Chart.min.js"></script>
  <script src="../../static/js/char-area.js"></script>
  <script src="../../static/js/char-bar.js"></script>
  <script src="../../static/js/char-pie.js"></script>
  <script src="../../static/js/char-mixed.js"></script>

  <script type="text/javascript" charset="utf8" src="https://cdn.datatables.net/1.10.22/js/jquery.dataTables.js"></script>
  <script src="../../static/js/database.js"></script>
</body>

</html>
```

Es importante poner en el atributo href dónde se encuentra cada fichero,

algunos son enlaces a internet y otros enlaces al proyecto en local. Pongamos un ejemplo dentro de local: base.html se encuentra en la carpeta rango y los estilos dentro de la carpeta css. El href completo será: ../../static/css/estilo.css Esto significa que con los dos primeros .. salimos de la carpeta rango a la carpeta templates, con los dos segundos .. salimos de la carpeta templates a la carpeta del proyecto, luego accedemos a la carpeta static, luego a css y finalmente al fichero estilo.css que queremos importar. E igual con los href de los ficheros javascript. De este modo se importan así:

```
<link href="../../static/css/estilo.css" rel="stylesheet">
<script src="../../static/js/database.js"></script>
```

- Login.html y Register.html son dos ficheros que contienen ejemplos del uso de HTML, de sus formularios y de como poner inputs y botones.
- El resto de ficheros los vamos a ver ahora pues corresponden al Dasboard

2.1 Charts

En esta sección vamos a ver los distintos tipos de charts que podemos crear. En general, para poder crear un chart necesitamos seguir una serie de pasos:
1. Importar Chart.js a nuestro proyecto, es decir, incluir la línea

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.5.0/Chart.min.js"></script>
```

al final de base.html, pues es el link donde se encuentran las funciones para poder realizar los charts.
2. Crear un fichero javascript para definir la función que queremos.
3. Añadir `<canvas id="myBarChart"></canvas>` a la zona del documento html que queramos.

Veámoslo mejor con un ejemplo: supongamos que queremos construir un gráfico de barras y ya tenemos importada la línea del punto número 1, los siguientes pasos serían:

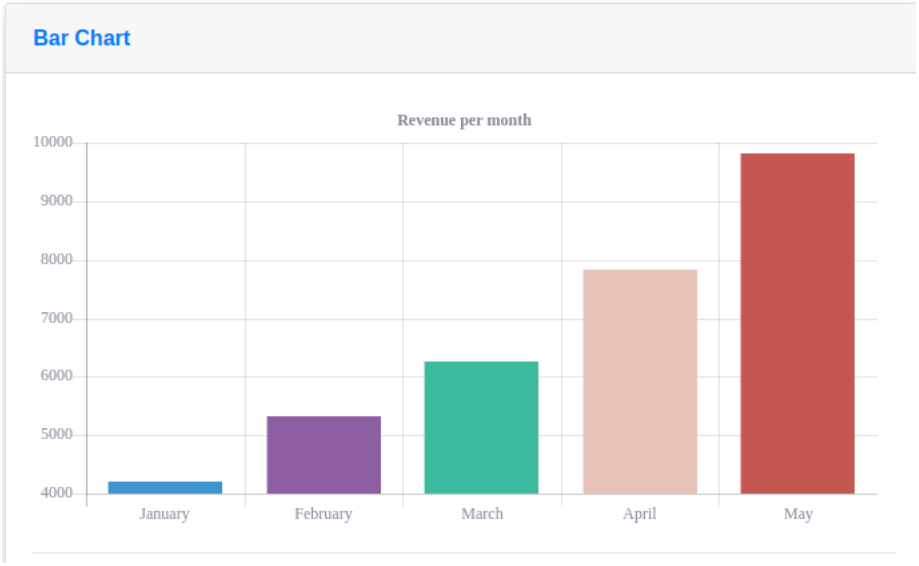
- Añadir el canvas como el anterior en el paso 3. Observemos que su id es "myBarChart" pues será importante para el siguiente paso. En este proyecto he añadido esta línea al fichero barchart.html.
- Crear un fichero nuevo en javascript que se llame "char-bar.js" e importarlo en base.html: `<script src="../../static/js/char-bar.js"></script>`
- Dentro de este nuevo fichero crearemos la función para el chart. Primero invocaremos en la variable llamada "chart html" el canvas con el mismo id que vimos en el primer paso, es decir, con el id del html "myBarChart"

```
var chart_html = document.getElementById("myBarChart");
```

- Por último, crearemos la función igual que la siguiente:

```
new Chart(chart_html, {
    type: 'bar', //tipo de gráfico que queremos
    data: {
      labels: ["January", "February", "March", "April", "May"],
      datasets: [
        {
          label: "Revenue", // un color para cada barra
          backgroundColor: ["#3e95cd", "#8e5ea2","#3cba9f","#e8c3b9","#c45850"],
          borderColor: "#4e73df",
          data: [4215, 5312, 6251, 7841, 9821],
        }
      ]
    },
    options: {
      legend: { display: false },
      title: {
        display: true,
        text: 'Revenue per month'
      }
    }
});
```

La variable type contiene el tipo de gráfico, como este gráfico es de barras entonces type es "bar".
Los labels de los meses corresponden a los datos que queremos representar en el eje de las Xs.
El backgroundColor corresponde al color de fondo de cada barra respectivamente y en orden.
Los datos de "data" corresponden al valor de cada barra, de modo que el primer dato corresponde a January, el segundo a February y así sucesivamente.
Y el texto corresponde al título del gráfico. A continuación podemos ver el resultado:

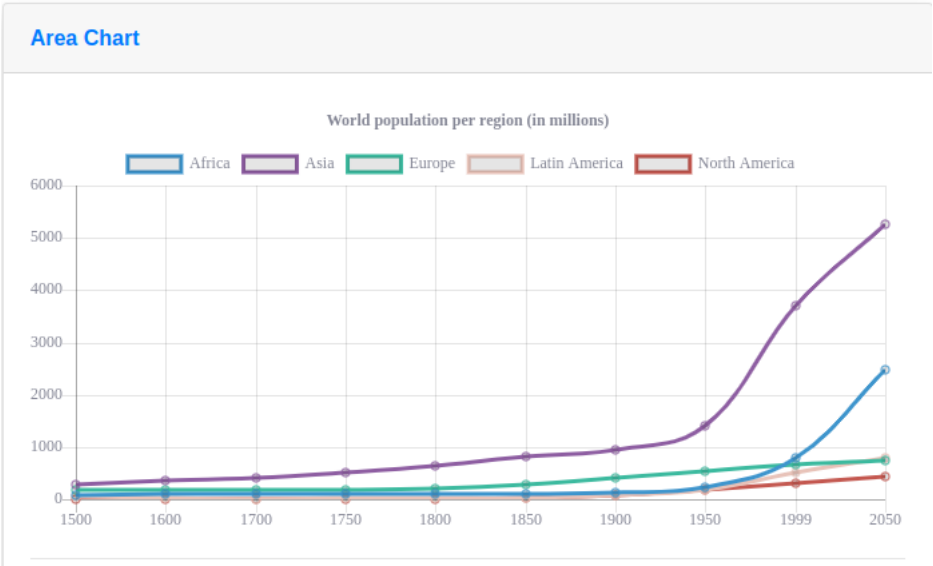


Una vez sabemos cómo hacer un tipo de chart, el resto de charts son iguales respecto de los datos, los sabels, el título, el color... Veamos ahora cada tipo por separado con su código.

Pero primero hagamos un import de todos ellos en el base.html:

```
<script src="../../static/js/char-area.js"></script>
<script src="../../static/js/char-bar.js"></script>
<script src="../../static/js/char-pie.js"></script>
<script src="../../static/js/char-mixed.js"></script>
```

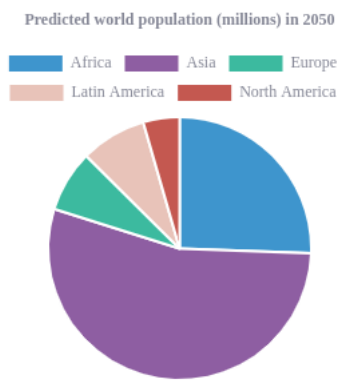
1. Area Chart: se encuentra en area-chart.js. Este utiliza el tipo "line" pues queremos una gráfica normal. Una vez más observamos que los labels se corresponden a los datos del eje X. Podemos observar que hay varias funciones, cada una de ellas se corresponden a cada color, de modo que si solo queremos una bastaría con eliminar las demás. Cada una de ellas se compone de "data", los datos de la función respecto de X, de "label", la etiqueta que queremos para esa función, y de "borderColor", el color de la función.



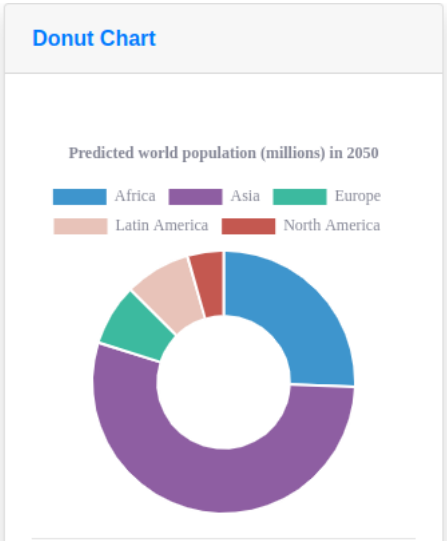
```
var chart_html = document.getElementById("myAreaChart");

new Chart(chart_html, {
  type: 'line',
  data: {
    labels: [1500,1600,1700,1750,1800,1850,1900,1950,1999,2050],
    datasets: [{
        data: [86,114,106,106,107,111,133,221,783,2478],
        label: "Africa",
        borderColor: "#3e95cd",
        fill: false
      }, {
        data: [282,350,411,502,635,809,947,1402,3700,5267],
        label: "Asia",
        borderColor: "#8e5ea2",
        fill: false
      }, {
        data: [168,170,178,190,203,276,408,547,675,734],
        label: "Europe",
        borderColor: "#3cba9f",
        fill: false
      }, {
        data: [40,20,10,16,24,38,74,167,508,784],
        label: "Latin America",
        borderColor: "#e8c3b9",
        fill: false
      }, {
        data: [6,3,2,2,7,26,82,172,312,433],
        label: "North America",
        borderColor: "#c45850",
        fill: false
      }
    ]
  },
  options: {
    title: {
      display: true,
      text: 'World population per region (in millions)'
    }
  }
});
```

2. Pie y Donut Chart: se encuentra en pie-chart.js. Ambos gráficos son iguales, lo único en lo que difieren es en el tipo: "pie" si lo queremos completo y "doughnut" si lo queremos con un agujero en el centro. Una vez más, estos gráficos contienen en la función la variable "data" para los datos de la función, la variable "label" para saber a que corresponde cada dato, y un color por cada dato.

Pie Chart

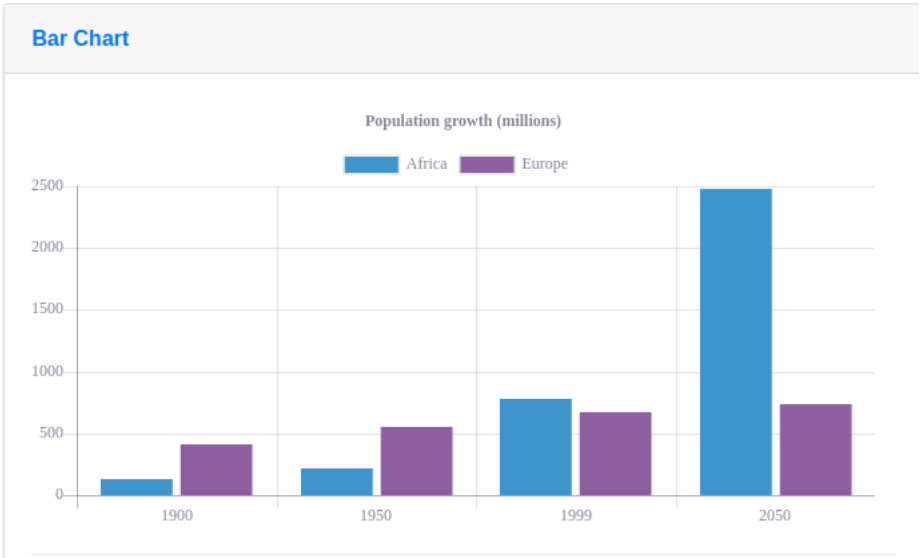



```
// Pie Chart Example
var chart_html = document.getElementById("myPieChart");
new Chart(chart_html, {
    type: 'pie',
    data: {
      labels: ["Africa", "Asia", "Europe", "Latin America", "North America"],
      datasets: [{
        label: "Population (millions)",
        backgroundColor: ["#3e95cd", "#8e5ea2","#3cba9f","#e8c3b9","#c45850"],
        data: [2478,5267,734,784,433]
      }]
    },
    options: {
      title: {
        display: true,
        text: 'Predicted world population (millions) in 2050'
      }
    }
});
```



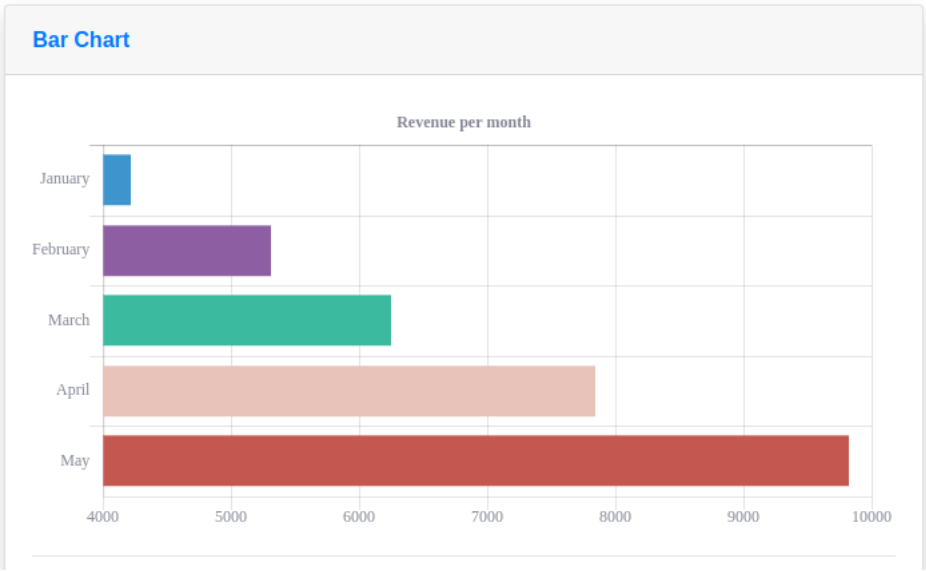

```
var chart_html = document.getElementById("myDonutChart");
new Chart(chart_html, {
    type: 'doughnut',
    data: {
      labels: ["Africa", "Asia", "Europe", "Latin America", "North America"],
      datasets: [{
        label: "Population (millions)",
        backgroundColor: ["#3e95cd", "#8e5ea2","#3cba9f","#e8c3b9","#c45850"],
        data: [2478,5267,734,784,433]
      }]
    },
    options: {
      title: {
        display: true,
        text: 'Predicted world population (millions) in 2050'
      }
    }
});
```

3. Otros Bar Chart: se encuentra en bar-chart.js. En primer lugar podemos observar un gráfico de barras de tipo "bar", sin embargo esta vez es doble. Podemos observar que esta vez tenemos dos arrays distintos que contienen el data, el color de fondo y el label de cada barra. De este modo cada uno de ellos corresponde a uno de los valores distintos, pero ambos en el mismo grupo.



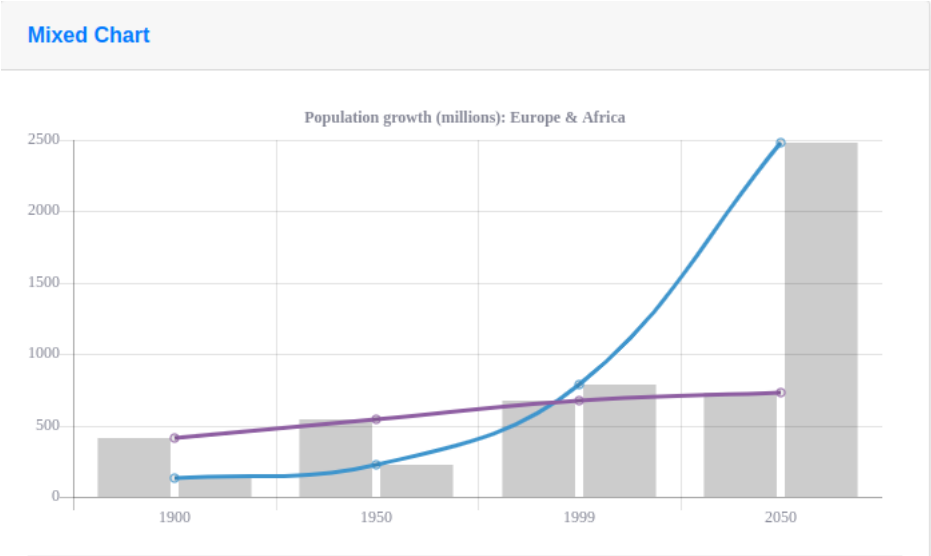

```
var chart_html = document.getElementById("myBarGroupChart");
new Chart(chart_html, {
    type: 'bar',
    data: {
      labels: ["1900", "1950", "1999", "2050"],
      datasets: [
        {
          label: "Africa",
          backgroundColor: "#3e95cd",
          data: [133,221,783,2478]
        }, {
          label: "Europe",
          backgroundColor: "#8e5ea2",
          data: [408,547,675,734]
        }
      ]
    },
    options: {
      title: {
        display: true,
        text: 'Population growth (millions)'
      }
    }
});
```

Por otro lado, tenemos exactamente el mismo gráfico que el de barras normal pero en horizontal. Esto se debe a que el tipo es "horizontalBar" en vez de solo "bar". Los datos son los mismos que en el primer ejemplo de "bar".




```
var chart_html = document.getElementById("myHorizontalChart");
new Chart(chart_html, {
    type: 'horizontalBar',
    data: {
      labels: ["January", "February", "March", "April", "May"],
      datasets: [
        {
          label: "Revenue", // un color para cada barra
          backgroundColor: ["#3e95cd", "#8e5ea2","#3cba9f","#e8c3b9","#c45850"],
          borderColor: "#4e73df",
          data: [4215, 5312, 6251, 7841, 9821],
        }
      ]
    },
    options: {
      legend: { display: false },
      title: {
        display: true,
        text: 'Revenue per month'
      }
    }
});
```

4. Mixed Chart: se encuentra en mixed-chart.js. Este es el último tipo de chart que veremos, es uno mixto que mezcla el area chart y el bar chart. Podemos observar que es de tipo "bar", sin embargo, a esta gráfica le hemos añadido el tipo "line" dentro de dos bases de datos y el tipo "bar" dentro de otras dos.




```
var chart_html = document.getElementById("myMixedChart");
new Chart(chart_html, {
    type: 'bar',
    data: {
      labels: ["1900", "1950", "1999", "2050"],
      datasets: [{
          label: "Europe",
          type: "line",
          borderColor: "#8e5ea2",
          data: [408,547,675,734],
          fill: false
        }, {
          label: "Africa",
          type: "line",
          borderColor: "#3e95cd",
          data: [133,221,783,2478],
          fill: false
        }, {
          label: "Europe",
          type: "bar",
          backgroundColor: "rgba(0,0,0,0.2)",
          data: [408,547,675,734],
        }, {
          label: "Africa",
          type: "bar",
          backgroundColor: "rgba(0,0,0,0.2)",
          backgroundColorHover: "#3e95cd",
          data: [133,221,783,2478]
        }
      ]
    },
    options: {
      title: {
        display: true,
        text: 'Population growth (millions): Europe & Africa'
      },
      legend: { display: false }
    }
});
```

En conclusión, hemos podido observar que todos los charts son iguales, solo difieren en el tipo de chart que queramos pues data, labels y backgroundColor siempre aparecen igual. Luego para realizar distintos charts solo tenemos que

modificar esos atributos.

2.2 Tablas de Datos

Los pasos para realizar una tabla de datos serían los mismos que los de los charts:
1. Importar en base.html la página web de donde sacamos el código javascript necesario para realizar la base de datos:

```
<script type="text/javascript" charset="utf8" src="https://cdn.datatables.net/1.10.22/js/jquery.dataTables.js"></script>
```

2. Importar en base.html el fichero donde escribiremos el script necesario para crear la base de datos. En este proyecto se llama "database.js" y se encuentra en la carpeta static/js:

```
<script src="../../static/js/database.js"></script>
```

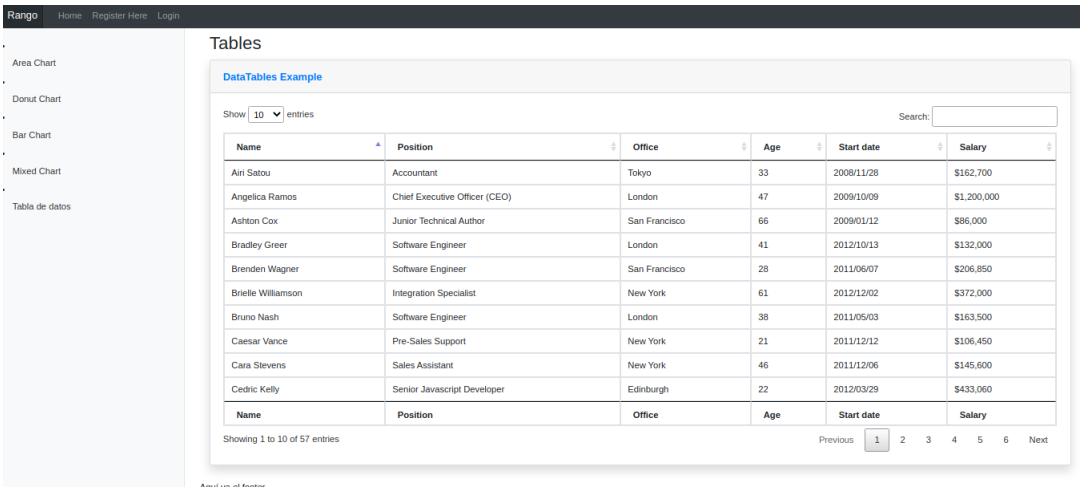
3. Creamos el html para alojar los datos de la tabla, en este proyecto se alojan en el fichero "table.html" en la carpeta "templates/rango".
Primero veremos cómo funcionan las tablas en HTML. Para crear una tabla en primer lugar necesitamos la etiqueta "table". La etiqueta "thead" significa que son los títulos de las columnas, la etiqueta "tr" significa que estamos en una fila, la etiqueta "th" significa que estamos en la casilla de esa fila. De este modo si hay dos etiquetas "th" seguidas las dos corresponden a la misma fila pero a distintas columnas, en orden. Dentro de las etiquetas "tr" crearemos todas sus filas (mirar ejemplo abajo). La etiqueta "tfoot" es exactamente igual que "thead" pero aparece al final de la tabla. Por último, para crear el cuerpo de la tabla necesitaremos la etiqueta "tbody". No nos podemos olvidar de cerrar todas las etiquetas que abrimos. Veamos ahora un ejemplo de tabla en HTML:

```
<table class="table table-bordered" id="dataTable" width="100%" cellspacing="0">
  <thead>
    <tr>
      <th>Name</th>
      <th>Position</th>
      <th>Office</th>
      <th>Age</th>
      <th>Start date</th>
      <th>Salary</th>
    </tr>
  </thead>
  <tfoot>
    <tr>
      <th>Name</th>
      <th>Position</th>
      <th>Office</th>
      <th>Age</th>
      <th>Start date</th>
      <th>Salary</th>
    </tr>
  </tfoot>
  <tbody>
    <tr>
      <td>Tiger Nixon</td>
      <td>System Architect</td>
      <td>Edinburgh</td>
      <td>61</td>
      <td>2011/04/25</td>
      <td>$320,800</td>
    </tr>
    <tr>
      <td>Garrett Winters</td>
      <td>Accountant</td>
      <td>Tokyo</td>
      <td>63</td>
      <td>2011/07/25</td>
      <td>$170,750</td>
    </tr>
```

4. Por último, necesitamos meter contenido en el fichero "database.js". Para ello utilizaremos JQuery. Fijémonos primero en la imagen de la tabla en HTML, la etiqueta "table" tiene como id "dataTable". Este id se lo tenemos que pasar a la función JQuery para que pueda reconocer la tabla. De este modo el fichero "database.js" quedará:

```
$(document).ready( function () {
    $('#dataTable').DataTable();
} );
```

Una vez tenemos todo ya nos aparecerá la tabla de datos, y quedará como la siguiente:


Rango
Home
Register Here
Login
Area Chart
Donut Chart
Bar Chart
Mixed Chart
Tabla de datos
Tables
DataTables Example
Show 10 entries
Search:
Name
Position
Office
Age
Start date
Salary
Airi Satou
Accountant
Tokyo
33
2008/11/28
$162,700
Angelica Ramos
Chief Executive Officer (CEO)
London
47
2009/10/09
$1,200,000
Ashton Cox
Junior Technical Author
San Francisco
66
2009/01/12
$86,000
Bradley Greer
Software Engineer
London
41
2012/10/13
$132,000
Brenden Wagner
Software Engineer
San Francisco
28
2011/06/07
$206,850
Brielle Williamson
Integration Specialist
New York
61
2012/12/02
$372,000
Bruno Nash
Software Engineer
London
38
2011/05/03
$163,500
Caesar Vance
Pre-Sales Support
New York
21
2011/12/12
$106,450
Cara Stevens
Sales Assistant
New York
46
2011/12/06
$145,600
Cedric Kelly
Senior Javascript Developer
Edinburgh
22
2012/03/29
$433,060
Name
Position
Office
Age
Start date
Salary
Showing 1 to 10 of 57 entries
Previous
1
2
3
4
5
6
Next