

# Django Introducción

## ¿Qué es Django?

- Django es un framework de Python que facilita la creación de sitios web usando Python.
- Django se ocupa de las cosas difíciles para que puedas concentrarte en crear tus aplicaciones web.
- Django enfatiza la reutilización de los componentes, también denominado DRY (Don't Repeat Yourself), y viene con funciones listas para usar como sistema de inicio de sesión, conexión a la base de datos y operaciones CRUD
- Django es especialmente útil para sitios web basados en bases de datos.

## ¿Cómo funciona Django?

Django sigue el patrón de diseño MVT (Model View Template).

- **Modelo:** los datos que desea presentar, generalmente datos de una base de datos.
- **Vista:** un controlador de solicitudes que devuelve la plantilla y el contenido relevantes, en función de la solicitud del usuario.
- **Template:** un archivo de texto (como un archivo HTML) que contiene el diseño de la página web, con lógica sobre cómo mostrar los datos.

## Modelo

El modelo proporciona datos de la base de datos.

En Django, los datos se entregan como Mapeo relacional de objetos (ORM), que es una técnica diseñada para facilitar el trabajo con bases de datos.

La forma más común de extraer datos de una base de datos es SQL. Un problema con SQL es que debe tener una comprensión bastante buena de la estructura de la base de datos para poder trabajar con ella.

Django, con ORM, facilita la comunicación con la base de datos, sin tener que escribir sentencias SQL complejas.

Los modelos suelen estar ubicados en un archivo llamado **models.py**.

## View

Una vista es una función o método que toma solicitudes http como argumentos, importa los modelos relevantes y descubre qué datos enviar a la plantilla y devuelve el resultado final.

Las vistas suelen estar ubicadas en un archivo llamado **views.py**.

## Template

Una plantilla es un archivo donde se describe cómo se debe representar el resultado.

Las plantillas suelen ser archivos .html, con código HTML que describe el diseño de una página web, pero también pueden estar en otros formatos de archivo para presentar otros resultados, pero nos concentraremos en los archivos .html.

Django usa HTML estándar para describir el diseño, pero usa etiquetas de Django para agregar lógica:

```
<h1>My Homepage</h1>

<p>My name is {{ firstname }}.</p>
```

Las plantillas de una aplicación se encuentran en una carpeta llamada **templates**.

## URLs

Django también proporciona una forma de navegar por las diferentes páginas de un sitio web.

Cuando un usuario solicita una URL, Django decide a qué vista la enviará.

Esto se hace en un archivo llamado **urls.py**.

## ¿Entonces qué pasa?

Cuando ha instalado Django y ha creado su primera aplicación web Django, y el navegador solicita la URL, esto es básicamente lo que sucede:

1. Django recibe la URL, verifica el archivo **urls.py** y llama a la vista que coincide con la URL.
2. La vista, ubicada en **views.py**, busca modelos relevantes.
3. Los modelos se importan del archivo **models.py**.
4. Luego, la vista envía los datos a una plantilla específica en la carpeta **template**.
5. La plantilla contiene etiquetas HTML y Django, y con los datos devuelve el contenido HTML terminado al navegador.

## Historia de Django

Django fue inventado por Lawrence Journal-World en 2003, para cumplir con los breves plazos del periódico y al mismo tiempo satisfacer las demandas de los desarrolladores web experimentados.

El lanzamiento inicial al público fue en julio de 2005.

La última versión de Django es 4.0.3 (marzo de 2022).

## Iniciemos un proyecto

Para instalar Django, debe tener Python instalado y un administrador de paquetes como PIP . PIP está incluido en Python desde la versión 3.4.

### Django requiere Python

Para verificar si su sistema tiene Python instalado, ejecute este comando en el símbolo del sistema:

```
python --version
```

Si Python está instalado, obtendrá un resultado con el número de versión, como este Python 4.0.5

Si descubre que no tiene Python instalado en su computadora, puede descargarlo de forma gratuita desde el siguiente sitio web: <https://www.python.org/>

### PIP

Para instalar Django, debe usar un administrador de paquetes como PIP, que se incluye en Python desde la versión 3.4.

Para verificar si su sistema tiene PIP instalado, ejecute este comando en el símbolo del sistema:

```
pip --version
```

Si PIP está instalado, obtendrá un resultado con el número de versión.

Para mí, en una máquina con Windows, el resultado se ve así:

```
pip 20.2.3 from c:\python39\lib\site-packages\pip (python 3.9)
```

Si no tiene PIP instalado, puede descargarlo e instalarlo desde esta página:

<https://pypi.org/project/pip/>

## Entorno virtual

Se sugiere tener un entorno virtual dedicado para cada proyecto de Django, y una forma de administrar un entorno virtual es venv , que se incluye en Python.

Con venv, puede crear un entorno virtual escribiendo esto en el símbolo del sistema, recuerde navegar hasta donde desea crear su proyecto:

Windows:

```
py -m venv myproject
```

Unix/Mac OS:

```
python -m venv myproject
```

Esto configurará un entorno virtual y creará una carpeta llamada "miproyecto" con subcarpetas y archivos, como este:

```
myproject
├── Include
├── Lib
├── Scripts
└── pyvenv.cfg
```

Luego tienes que activar el entorno, escribiendo este comando:

Windows:

```
myproject\Scripts\activate.bat
```

Unix/Mac OS:

```
source myproject/bin/activate
```

Una vez que el entorno esté activado, verá este resultado en el símbolo del sistema:

Windows:

```
(myproject) C:\Users\Your Name>
```

Unix/Mac OS:

```
(myproject) ... $
```

**Nota:** Debe activar el entorno virtual cada vez que abra el símbolo del sistema para trabajar en su proyecto.



## Instalar Django

Django se instala usando pip, con este comando:

Windows:

```
(myproject) C:\Users\Your Name>py -m pip install Django
```

Unix/Mac OS:

```
(myproject) ... $ python -m pip install Django
```

Lo que dará un resultado similar a este

```
Collecting Django
  Downloading Django-4.0.3-py3-none-any.whl (8.0 MB)
    | 8.0 MB 2.2 MB/s
Collecting sqlparse>=0.2.2
  Using cached sqlparse-0.4.2-py3-none-any.whl (42 kB)
Collecting asgiref<4,>=3.4.1
  Downloading asgiref-3.5.0-py3-none-any.whl (22 kB)
Collecting tzdata; sys_platform == "win32"
  Downloading tzdata-2021.5-py2.py3-none-any.whl (339 kB)
    | 339 kB 6.4 MB/s
Installing collected packages: sqlparse, asgiref, tzdata, Django
Successfully installed Django-4.0.3 asgiref-3.5.0 sqlparse-0.4.2 tzdata-2021.5
WARNING: You are using pip version 20.2.3; however, version 22.0.4 is available.
You should consider upgrading via the 'C:\Users\Your Name\myproject\Scripts\python.exe -m pip install --upgrade pip' command.
```

## ¿Windows, Mac o Unix?

Puede ejecutar este proyecto en cualquiera de los dos. Hay algunas pequeñas diferencias, como cuando se escriben comandos en el símbolo del sistema, Windows usa py como primera palabra en la línea de comandos, mientras que Unix y MacOS usan python:

Windows:

```
py --version
```

Unix/Mac OS:

```
python --version
```

## Comprobar la versión de Django

Puede verificar si Django está instalado solicitando su número de versión de esta manera:

```
(myproject) C:\Users\Your Name>django-admin --version
```

Si Django está instalado, obtendrá un resultado con el número de versión:

4.0.5

## Creación proyecto de Django

### mi primer proyecto

Una vez que haya encontrado un nombre adecuado para su proyecto Django, como el mío: **myworld**, navegue hasta el lugar del sistema de archivos en el que desea almacenar el código (en el entorno virtual) y ejecute este comando en el símbolo del sistema:

```
django-admin startproject myworld
```

Django crea una carpeta **myworld** en la computadora, con este contenido:

```
myworld
manage.py
myworld/
__init__.py
asgi.py
settings.py
urls.py
wsgi.py
```

Todos estos son archivos y carpetas con un significado específico, conocerá algunos de ellos más adelante , pero por ahora, es más importante saber que esta es la ubicación de su proyecto y que puede comenzar a crear aplicaciones en eso.

### Ejecutar el Proyecto Django

Ahora que tiene un proyecto Django, puede ejecutarlo y ver cómo se ve en un navegador.

Navegue a la carpeta /myworld y ejecute este comando en el símbolo del sistema:

```
py manage.py runserver
```

Lo que producirá este resultado:

```
Watching for file changes with StatReloader
Performing system checks...
```

```
System check identified no issues (0 silenced).
```

```
You have 18 unapplied migration(s). Your project may not work properly until you apply the
migrations for app(s): admin, auth, contenttypes, sessions.
```

```
Run 'python manage.py migrate' to apply them.
```

```
December 02, 2021 - 13:14:51
```

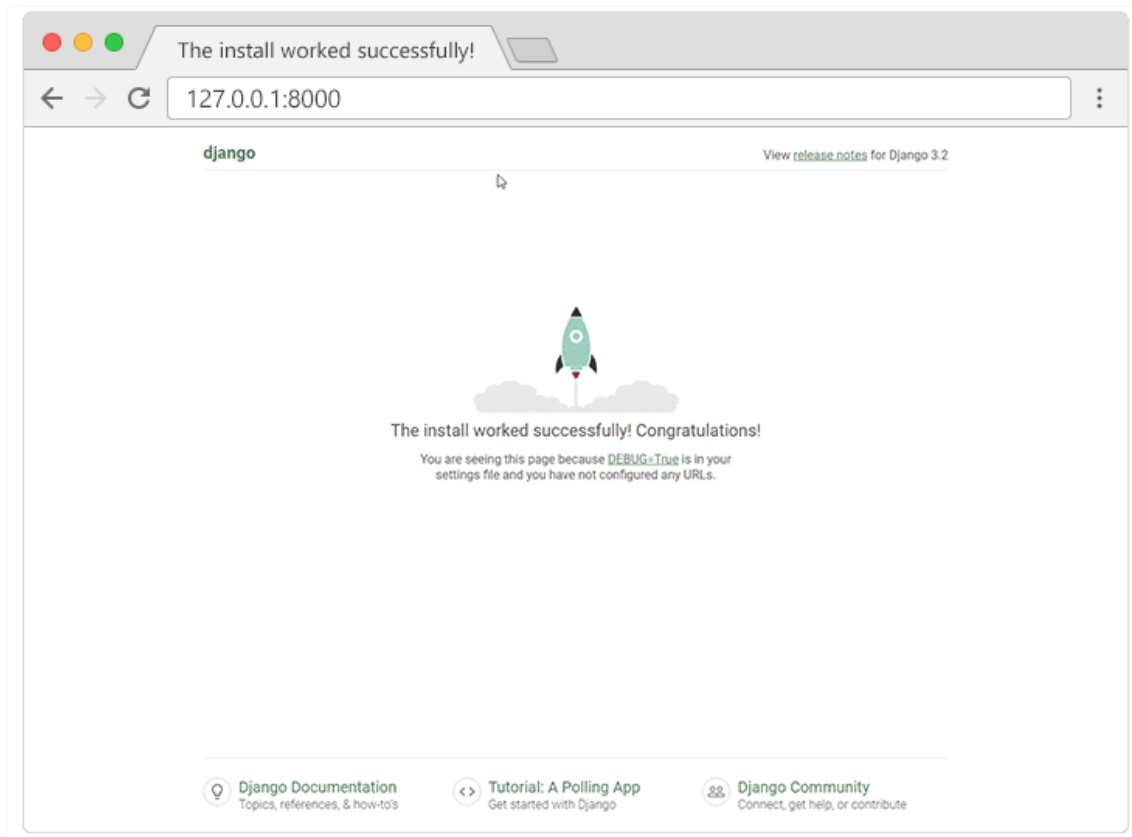
```
Django version 3.2.9, using settings 'myworld.settings'
```

```
Starting development server at http://127.0.0.1:8000/
```

```
Quit the server with CTRL-BREAK.
```

Abra una nueva ventana del navegador y escriba 127.0.0.1:8000 en la barra de direcciones.

El resultado:



## Creación de Aplicación de Django

### ¿Qué es una aplicación?

Una aplicación es una aplicación web que tiene un significado específico en su proyecto, como una página de inicio, un formulario de contacto o una base de datos de miembros.

En este tutorial crearemos una aplicación que nos permita enumerar y registrar miembros en una base de datos.

Pero primero, creemos una aplicación simple de Django que muestre "¡Hola mundo!".

### Crear aplicación

Voy a nombrar mi aplicación **members**.

Comience navegando a la ubicación seleccionada donde desea almacenar la aplicación y ejecute el comando a continuación.

Si el servidor aún se está ejecutando y no puede escribir comandos, presione [CTRL] [BREAK] para detener el servidor y debería volver al entorno virtual.

```
py manage.py startapp members
```

Django crea una carpeta nombrada **members** en mi proyecto, con este contenido:

```
myworld
manage.py
myworld/
members/
  migrations/
  __init__.py
__init__.py
admin.py
apps.py
models.py
tests.py
views.py
```

Estos son todos los archivos y carpetas con un significado específico.

Primero, eche un vistazo al archivo llamado **views.py**.

Aquí es donde recopilamos la información que necesitamos para enviar una respuesta adecuada.



## Vistas de Django

### Views

Las vistas de Django son funciones de Python que toman solicitudes http y devuelven respuestas http, como documentos HTML.

Una página web que usa Django está llena de vistas con diferentes tareas y misiones.

Las vistas generalmente se colocan en un archivo llamado **views.py** ubicado en la carpeta de su aplicación.

Hay una **views.py** en su carpeta **members** que se ve así:

**members/views.py:**

```
from django.shortcuts import render
```

```
# Create your views here.
```

Encuéntrelo y ábralo, y reemplace el contenido con esto:

**members/views.py:**

```
from django.shortcuts import render
from django.http import HttpResponse
```

```
def index(request):
    return HttpResponse("Hello world!")
```

Este es un ejemplo simple de cómo enviar una respuesta al navegador.

Pero, ¿cómo podemos ejecutar la vista? Bueno, debemos llamar a la vista a través de una URL.

### URLs

Cree un archivo con el nombre **urls.py** en la misma carpeta que el archivo **views.py** y escriba este código en él:

**members/urls.py:**

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

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

El archivo **urls.py** que acaba de crear es específico para la aplicación **members**. También tenemos que hacer algunas rutas en el directorio raíz **myworld**. Esto puede parecer complicado, pero por ahora, solo siga las instrucciones a continuación.

Hay un archivo llamado **urls.py** en la carpeta **myworld**, abra ese archivo y agregue el módulo **include** en la declaración **import**, y también agregue una función **path()** en la lista **urlpatterns[]**, con argumentos que enrutarán a los usuarios que ingresan a través de **127.0.0.1:8000/members/**.

Entonces su archivo se verá así:

**myworld/urls.py:**

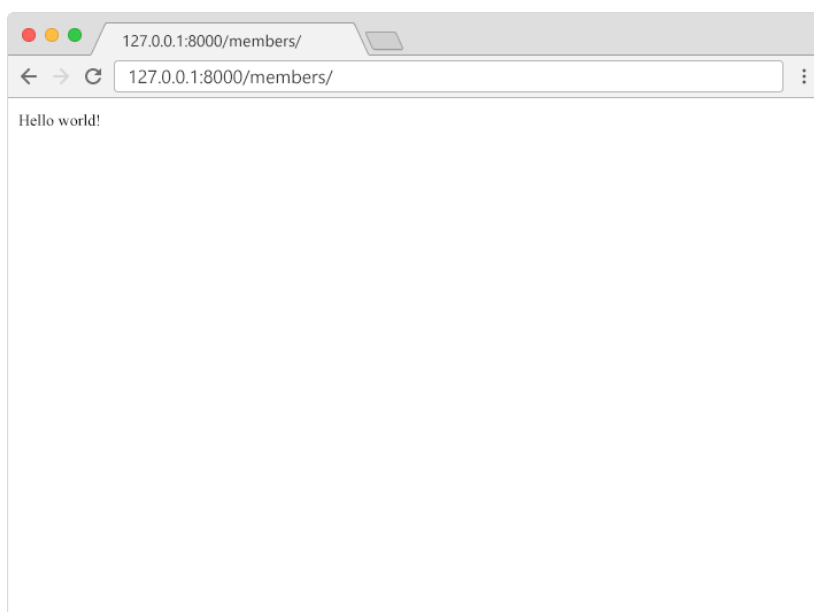
```
from django.contrib import admin
from django.urls import include, path

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

Si el servidor no se está ejecutando, navegue hasta la carpeta **/myworld** y ejecute este comando en el símbolo del sistema:

```
py manage.py runserver
```

En la ventana del navegador, escriba **127.0.0.1:8000/members/** en la barra de direcciones.



## Plantillas Django

### Templates

En la página de introducción de Django , aprendimos que el resultado debe estar en HTML y debe crearse en una plantilla.

Cree una carpeta **templates** dentro de la carpeta **members** y cree un archivo HTML llamado **myfirst.html**.

La estructura del archivo debería ser algo como esto:

```
myworld
manage.py
myworld/
members/
  templates/
    myfirst.html
```

Abra el archivo HTML e inserte lo siguiente:

**members/templates/myfirst.html:**

```
<!DOCTYPE html>
<html>
<body>

<h1>Hello World!</h1>
<p>Welcome to my first Django project!</p>

</body>
</html>
```

### Modificar la vista

Abra el archivo views.py y reemplace la vista de índice con esto:

**members/views.py:**

```
from django.http import HttpResponse
from django.template import loader

def index(request):
    template = loader.get_template('myfirst.html')
    return HttpResponse(template.render())
```

## Cambiar Settings

Para poder trabajar con cosas más complicadas que "Hello World!", tenemos que decirle a Django que se crea una nueva aplicación.

Esto se hace en el archivo **settings.py** en la carpeta **myworld**.

Busque la lista **INSTALLED\_APPS[]** y agregue la aplicación members de esta manera:

myworld/settings.py:

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

Luego ejecuta este comando:

```
py manage.py migrate
```

Lo que producirá esta salida:

```
Operations to perform:  
Apply all migrations: admin, auth, contenttypes, sessions  
Running migrations:  
Applying contenttypes.0001_initial... OK  
Applying auth.0001_initial... OK  
Applying admin.0001_initial... OK  
Applying admin.0002_logentry_remove_auto_add... OK  
Applying admin.0003_logentry_add_action_flag_choices... OK  
Applying contenttypes.0002_remove_content_type_name... OK  
Applying auth.0002_alter_permission_name_max_length... OK  
Applying auth.0003_alter_user_email_max_length... OK  
Applying auth.0004_alter_user_username_opts... OK  
Applying auth.0005_alter_user_last_login_null... OK  
Applying auth.0006_require_contenttypes_0002... OK  
Applying auth.0007_alter_validators_add_error_messages... OK  
Applying auth.0008_alter_user_username_max_length... OK  
Applying auth.0009_alter_user_last_name_max_length... OK  
Applying auth.0010_alter_group_name_max_length... OK  
Applying auth.0011_update_proxy_permissions... OK  
Applying auth.0012_alter_user_first_name_max_length... OK  
Applying sessions.0001_initial... OK
```

```
(myproject)C:\Users\Your Name\myproject\myworld>
```

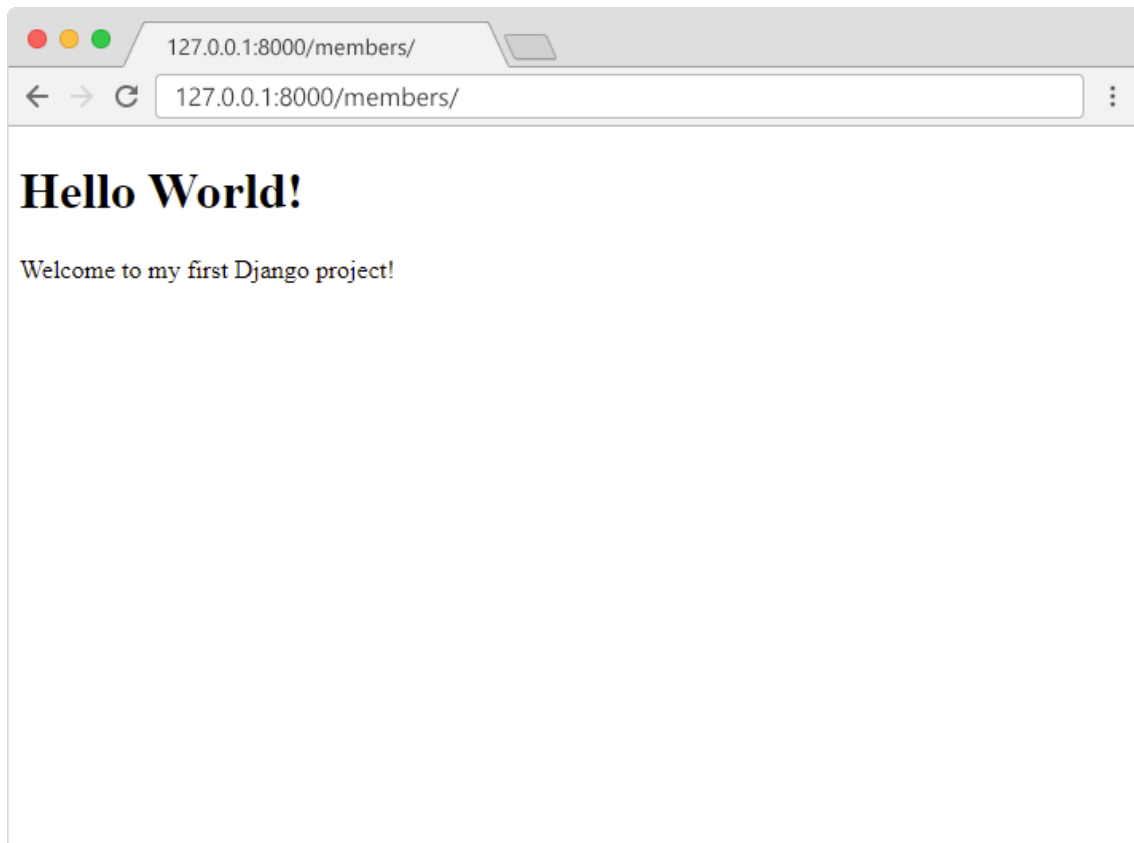


Inicie el servidor navegando a la carpeta **/myworld** y ejecute este comando:

```
py manage.py runserver
```

En la ventana del navegador, escriba **127.0.0.1:8000/members/** en la barra de direcciones.

El resultado debería verse así:



## Modelos Django

Un modelo de Django es una tabla en su base de datos.

Cuando creamos el proyecto Django, obtuvimos una base de datos SQLite vacía. Fue creado en la carpeta myworld raíz.

Usaremos esta base de datos.

### Crear tabla (modelo)

Para crear una nueva tabla, debemos crear un nuevo modelo.

En la carpeta **/members/**, abra el archivo **models.py**. Está casi vacío por defecto, con solo una declaración de importación y un comentario:

**members/models.py:**

```
from django.db import models
```

```
# Create your models here.
```

Para agregar una tabla de miembros en nuestra base de datos, comience creando una clase de miembros y describa los campos de la tabla en ella:

**members/models.py:**

```
from django.db import models
```

```
class Members(models.Model):
```

```
    firstname = models.CharField(max_length=255)
```

```
    lastname = models.CharField(max_length=255)
```

El primer campo, "firstname" es un campo de texto y contendrá el primer nombre de los miembros.

El segundo campo, "apellido" también es un campo de Texto, con el apellido de los miembros.

Tanto el "nombre" como el "apellido" están configurados para tener un máximo de 255 caracteres.

Luego navegue a la carpeta **/myworld/** y ejecute este comando:

```
py manage.py makemigrations members
```

Lo que resultará en esta salida:

```
Migrations for 'members':
members\migrations\0001_initial.py
- Create model Members

(myproject) C:\Users\Your Name\myproject\myworld>
```

Django crea un archivo con cualquier cambio nuevo y almacena el archivo en la carpeta **/migrations/**.

La próxima vez que ejecute **py manage.py migrate** Django, creará y ejecutará una instrucción SQL, basada en el contenido del nuevo archivo en la carpeta de migraciones.

Ejecute el comando de migración:

```
py manage.py migrate
```

Lo que resultará en esta salida:

```
Operations to perform:
Apply all migrations: admin, auth, contenttypes, members, sessions
Running migrations:
Applying members.0001_initial... OK

(myproject) C:\Users\Your Name\myproject\myworld>
```

La sentencia SQL creada a partir del modelo es:

```
CREATE TABLE "members_members" (
  "id" INT NOT NULL PRIMARY KEY AUTOINCREMENT,
  "firstname" varchar(255) NOT NULL,
  "lastname" varchar(255) NOT NULL);
```

¡Ahora tiene una tabla de Miembros en su base de datos!

## Django Agregar miembros

### Agregar registros

La tabla Miembros está vacía, debemos agregarle algunos miembros.

Aprenda a crear una interfaz de usuario que se encargará de las operaciones CRUD (Crear, Leer, Actualizar, Eliminar), pero por ahora, escribamos el código de Python directamente en el intérprete de Python (shell de Python) y añadamos algunos miembros en nuestra base de datos, sin la interfaz de usuario.

Para abrir un shell de Python, escriba este comando:

```
py manage.py shell
```

Ahora que estamos en el shell, el resultado debería ser algo como esto:

```
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>>
```

En la parte inferior, después de los tres `>>>` escribe lo siguiente:

```
>>> from members.models import Members
```

Presiona [enter] y escribe esto para ver la tabla de miembros vacía:

```
>>> Members.objects.all()
```

Esto debería darte un objeto QuerySet vacío, como este:

```
<QuerySet []>
```

Un QuerySet es una colección de datos de una base de datos.

Agregue un registro a la tabla, ejecutando estas dos líneas:

```
>>> member = Members(firstname='Emil', lastname='Refsnes')
>>> member.save()
```

Ejecute este comando para ver si la tabla **Members** tiene un miembro:

```
>>> Members.objects.all().values()
```



El resultado se verá así:

```
<QuerySet [{ 'id': 1, 'firstname': 'Emil', 'lastname': 'Refsnes'}]>
```

## Agregar varios registros

Puede agregar varios registros haciendo una lista de objetos **Members** y ejecutar **.save()** en cada entrada:

```
>>> member1 = Members(firstname='Tobias', lastname='Refsnes')
>>> member2 = Members(firstname='Linus', lastname='Refsnes')
>>> member3 = Members(firstname='Lene', lastname='Refsnes')
>>> member4 = Members(firstname='Stale', lastname='Refsnes')
>>> members_list = [member1, member2, member3, member4]
>>> for x in members_list:
>>>     x.save()
```

Ahora hay 5 miembros en la tabla Miembros:

```
>>> Members.objects.all().values()
<QuerySet [{ 'id': 1, 'firstname': 'Emil', 'lastname': 'Refsnes'},
{'id': 2, 'firstname': 'Tobias', 'lastname': 'Refsnes'}, {'id': 3, 'firstname': 'Linus', 'lastname':
'Refsnes'}, {'id': 4, 'firstname': 'Lene', 'lastname': 'Refsnes'}, {'id': 5, 'firstname': 'Stale',
'lastname': 'Refsnes'}]>
```

## Ver en el navegador

Queremos ver el resultado en una página web, no en un entorno de shell de Python.

Para ver el resultado en una página web, podemos crear una vista para esta tarea en particular.

En la aplicación **members**, abra el archivo **views.py** :

**members/views.py:**

```
from django.http import HttpResponse
from django.template import loader

def index(request):
    template = loader.get_template('myfirst.html')
    HttpResponse(template.render())
```

Cambie el contenido del **views.py** archivo para que se vea así:

**members/views.py:**

```
from django.http import HttpResponse
from django.template import loader
from .models import Members

def index(request):
    mymembers = Members.objects.all().values()
    output = ""
    for x in mymembers:
        output += x["firstname"]
    return HttpResponse(output)
```

Como puede ver en la línea 3, el modelo **Members** se importa y la vista **index** hace lo siguiente:

- crea un objeto **mymembers** con todos los valores del modelo **Members**.
- Recorre todos los elementos del **mymembers** objeto para crear una cadena con todos los valores de nombre.
- Devuelve la cadena como salida al navegador.
- 

Vea el resultado en su navegador. Si todavía está en el shell de Python, escriba este comando para salir del shell:

```
>>> quit()
```

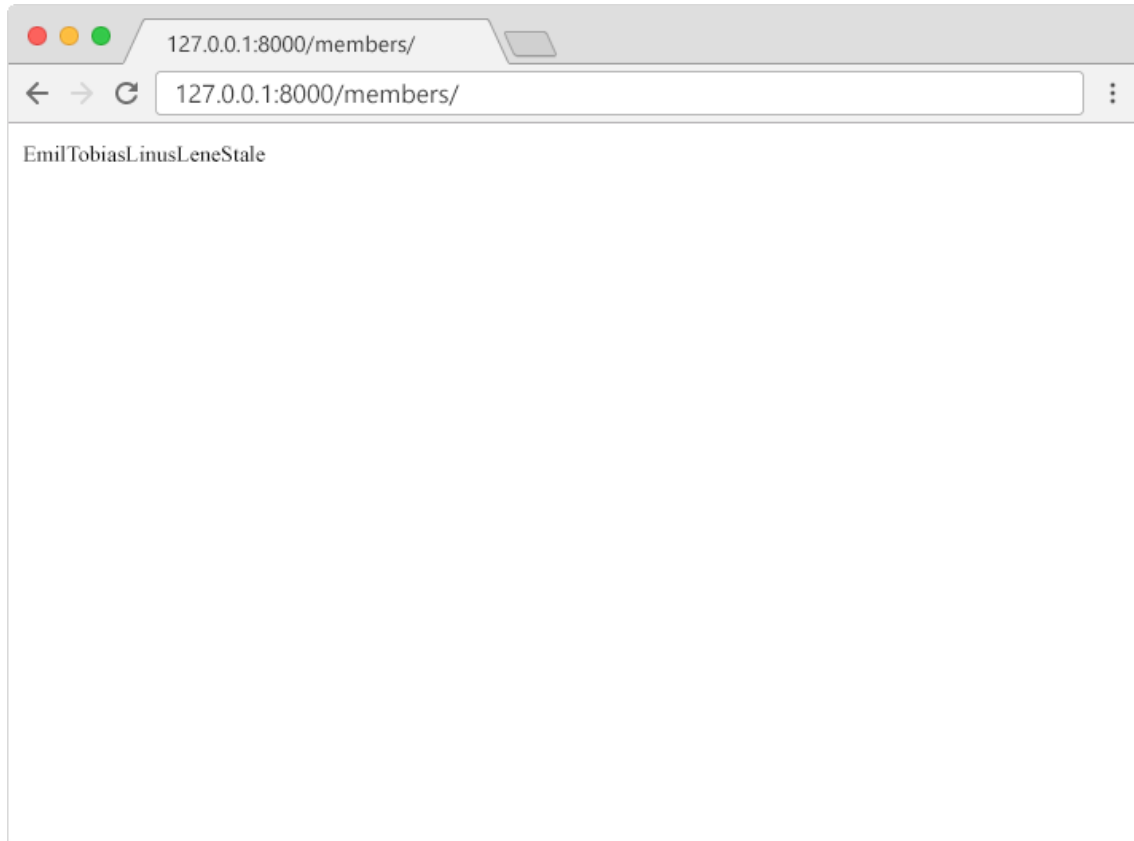


Navegue a la carpeta **/myworld/** y escriba esto para iniciar el servidor:

```
py manage.py runserver
```

En la ventana del navegador, escriba **127.0.0.1:8000/members/** en la barra de direcciones.

El resultado:



## Agregar Template

### Agregar de una plantilla a la aplicación

Para agregar algo de HTML alrededor de los valores, crearemos una plantilla para la aplicación.

Todas las plantillas deben estar ubicadas en la carpeta **templates** de su aplicación

En la carpeta **templates**, cree un archivo llamado **index.html**, con el siguiente contenido:

**members/templates/index.html:**

```
<h1>Members</h1>

<table border="1">
{% for x in mymembers %}
<tr>
<td>{{ x.id }}</td>
<td>{{ x.firstname }}</td>
<td>{{ x.lastname }}</td>
</tr>
{% endfor %}
</table>
```

Las partes `{% %}` y `{{ }}` se llaman etiquetas de plantilla.

Las etiquetas de plantilla le permiten realizar lógica y representar variables en sus plantillas; aprenderá más sobre las etiquetas de plantilla más adelante.

## Modificar la vista

Cambie la view `index` para incluir la plantilla:

`members/views.py`:

```
from django.http import HttpResponse
from django.template import loader
from .models import Members

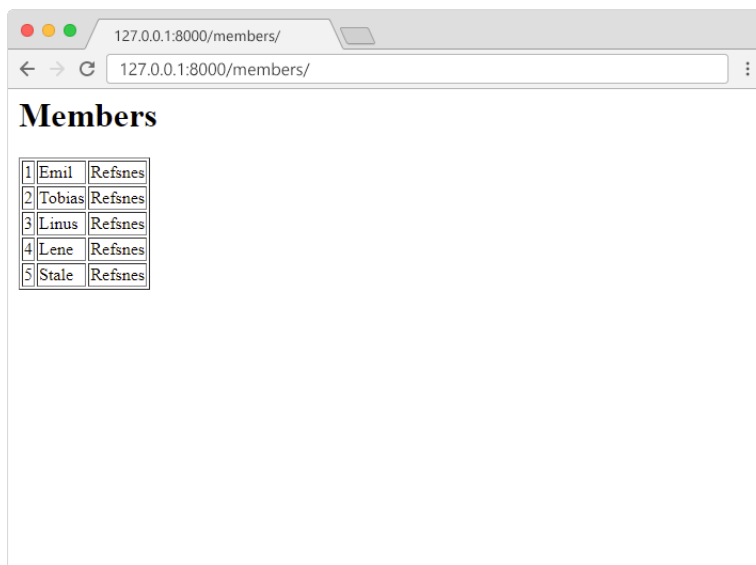
def index(request):
    mymembers = Members.objects.all().values()
    template = loader.get_template('index.html')
    context = {
        'mymembers': mymembers,
    }
    return HttpResponse(template.render(context, request))
```

La vista `index` hace lo siguiente:

- Crea un objeto **mymembers** con todos los valores del modelo `Members`.
- Carga una plantilla `index.html`.
- Crea un objeto que contiene el objeto `mymember`.
- Envía el objeto a la plantilla.
- Muestra el HTML que representa la plantilla.

En la ventana del navegador, escriba **127.0.0.1:8000/members/** en la barra de direcciones.

El resultado:



## Django Añadir registro

### Añadir registros

Hasta ahora, hemos creado una tabla de miembros en nuestra base de datos y hemos insertado cinco registros escribiendo código en el shell de Python.

También hemos realizado una plantilla que nos permite mostrar el contenido de la tabla en una página web.

Ahora queremos poder crear nuevos miembros desde una página web.

### Template

Comience agregando un enlace en la plantilla de miembros:

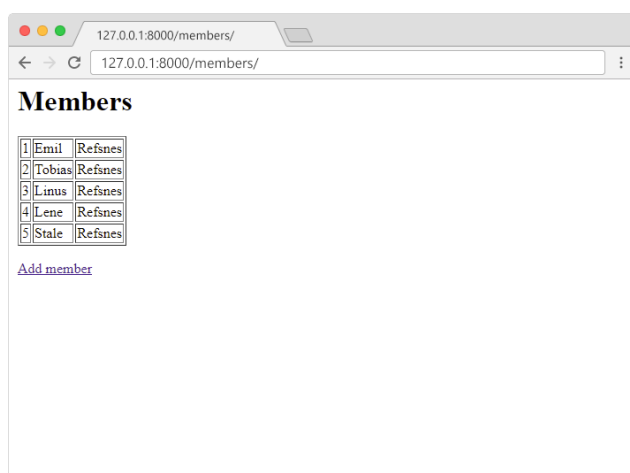
**members/templates/index.html:**

```
<h1>Members</h1>

<table border="1">
{% for x in mymembers %}
<tr>
<td>{{ x.id }}</td>
<td>{{ x.firstname }}</td>
<td>{{ x.lastname }}</td>
</tr>
{% endfor %}
</table>

<p>
<a href="add/">Add member</a>
</p>
```

El resultado se verá así:



## Nuevo Template

Agregue una nueva plantilla en la carpeta **templates**, llamada **add.html**:

**members/templates/add.html:**

```
<h1>Add member</h1>

<form action="/addrecord/" method="post">
{% csrf_token %}
First Name:<br>
<input name="first">
<br><br>
Last Name:<br>
<input name="last">
<br><br>
<input type="submit" value="Submit">
</form>
```

La plantilla contiene un formulario HTML vacío con dos campos de entrada y un botón de envío.

Nota: Django requiere esta línea en el formulario:

```
{% csrf_token %}
```

para manejar falsificaciones de solicitudes entre sitios en formularios donde el método es POST.

## View

A continuación, agregue una vista en el archivo **members/views.py**, nombre la nueva vista **add**:

**members/views.py:**

```
from django.http import HttpResponse
from django.template import loader
from .models import Members

def index(request):
    mymembers = Members.objects.all().values()
    template = loader.get_template('index.html')
    context = {
        'mymembers': mymembers,
    }
    return HttpResponse(template.render(context, request))

def add(request):
    template = loader.get_template('add.html')
    return HttpResponse(template.render({}, request))
```



## URLs

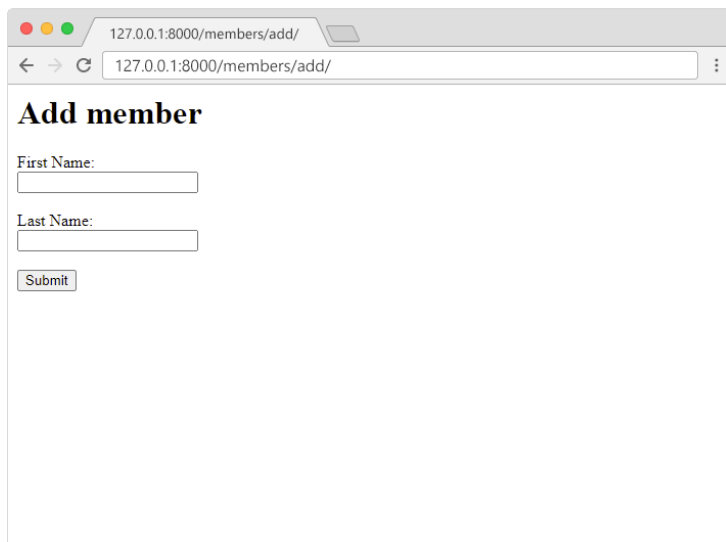
Agregue una función de ruta ( ) en el archivo **members/urls.py**, que apunte la URL **127.0.0.1:8000/members/add/** a la ubicación correcta:

**members/urls.py:**

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

urlpatterns = [
    path("", views.index, name='index'),
    path('add/', views.add, name='add'),
]
```

En el navegador, haga clic en el enlace "Agregar miembro" y el resultado debería verse así:



The screenshot shows a web browser window with the address bar displaying '127.0.0.1:8000/members/add/'. The page content includes the heading 'Add member', followed by two text input fields labeled 'First Name:' and 'Last Name:', and a 'Submit' button at the bottom.

## Más URLs

¿Notó el atributo **action** en el formulario HTML? El atributo **action** especifica a dónde enviar los datos del formulario, en este caso los datos del formulario se enviarán a **addrecord/**, por lo que debemos agregar una función **path()** en el archivo **members/urls.py** que apunte a la vista correcta:

**members/urls.py:**

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

urlpatterns = [
    path("", views.index, name='index'),
    path('add/', views.add, name='add'),
    path('add/addrecord/', views.addrecord, name='addrecord'),
]
```

## Código para agregar registros

Hasta ahora hemos creado la interfaz de usuario y apuntamos la URL a la vista llamada **addrecord**, pero aún no hemos creado la vista.

Asegúrese de agregar la vista **addrecord** en el archivo **members/views.py**:

**members/views.py**:

```
from django.http import HttpResponseRedirect
from django.template import loader
from django.urls import reverse
from .models import Members

def index(request):
    mymembers = Members.objects.all().values()
    template = loader.get_template('index.html')
    context = {
        'mymembers': mymembers,
    }
    return HttpResponseRedirect(template.render(context, request))

def add(request):
    template = loader.get_template('add.html')
    return HttpResponseRedirect(template.render({}, request))

def addrecord(request):
    x = request.POST['first']
    y = request.POST['last']
    member = Members(firstname=x, lastname=y)
    member.save()
    return HttpResponseRedirect(reverse('index'))
```

Cambios que se realizan en el archivo views.py:

**Línea 1:** importar HttpResponseRedirect

**Línea 3:** importar reverse

La vista **addrecord** hace lo siguiente:

- Obtiene el nombre y el apellido con la declaración **request.POST**.
- Agrega un nuevo registro en la tabla de miembros.
- Redirige al usuario a la vista **index**.



Intente agregar un nuevo registro y vea cómo funciona:

**Add member**

First Name:

Last Name:

Si presiona el botón Enviar, la tabla de miembros debería haberse actualizado:

**Members**

1	Emil	Refsnes
2	Tobias	Refsnes
3	Linus	Refsnes
4	Lene	Refsnes
5	Stale	Refsnes
17	Jane	Doe

[Add member](#)

## Eliminación de registros

Para eliminar un registro no necesitamos una nueva plantilla, pero necesitamos hacer algunos cambios en la plantilla de miembros.

Por supuesto, puede elegir cómo desea agregar un botón de eliminación, pero en este ejemplo, agregaremos un enlace "eliminar" para cada registro en una nueva columna de la tabla.

El enlace "eliminar" también contendrá la ID de cada registro.

## Modificar Template

Agregue una columna "eliminar" en la plantilla de miembros:

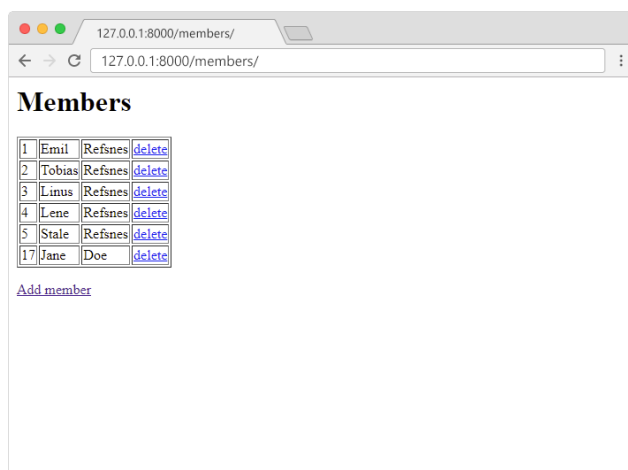
**members/templates/index.html:**

```
<h1>Members</h1>

<table border="1">
{% for x in mymembers %}
  <tr>
    <td>{{ x.id }}</td>
    <td>{{ x.firstname }}</td>
    <td>{{ x.lastname }}</td>
    <td><a href="delete/{{ x.id }}">delete</a></td>
  </tr>
{% endfor %}
</table>

<p>
<a href="add/">Add member</a>
</p>
```

El resultado se verá así:



## URLs

El enlace "eliminar" en la tabla HTML apunta a **127.0.0.1:8000/members/delete/** , por lo que agregaremos una función **path()** en el archivo **members/urls.py**, que apunta la URL a la ubicación correcta, con la ID como parámetro:

**members/urls.py:**

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

urlpatterns = [
    path('', views.index, name='index'),
    path('add/', views.add, name='add'),
    path('add/addrecord/', views.addrecord, name='addrecord'),
    path('delete/<int:id>', views.delete, name='delete'),
]
```

## Código para borrar registros

Ahora necesitamos agregar una nueva vista llamada **delete** en el archivo **members/views.py**:

**members/views.py:**

```
from django.http import HttpResponseRedirect
from django.template import loader
from django.urls import reverse

from .models import Members

def index(request):
    mymembers = Members.objects.all().values()
    template = loader.get_template('index.html')
    context = {
        'mymembers': mymembers,
    }
    return HttpResponseRedirect(template.render(context, request))

def add(request):
    template = loader.get_template('add.html')
    return HttpResponseRedirect(template.render({}, request))

def addrecord(request):
    x = request.POST['first']
    y = request.POST['last']
    member = Members(firstname=x, lastname=y)
```

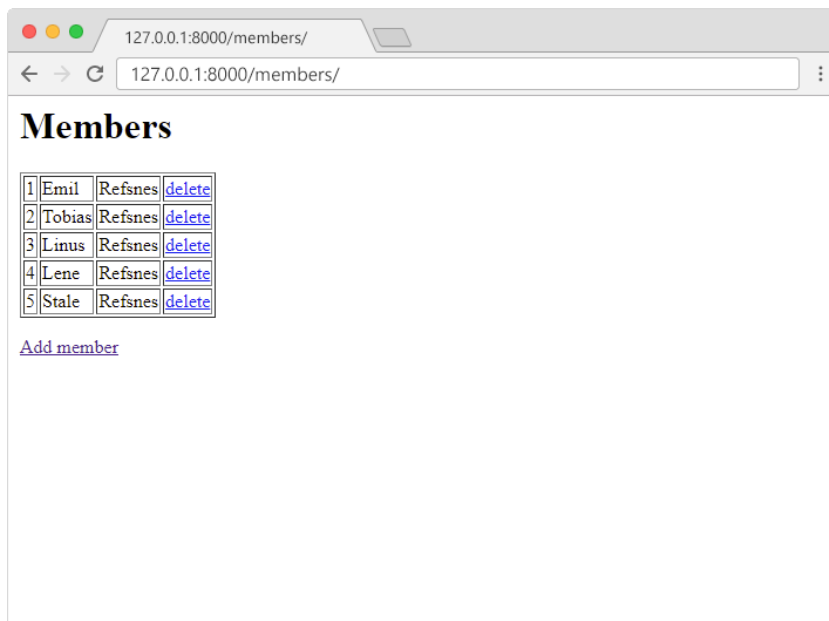
```
member.save()
return HttpResponseRedirect(reverse('index'))

def delete(request, id):
    member = Members.objects.get(id=id)
    member.delete()
    return HttpResponseRedirect(reverse('index'))
```

La vista de eliminación hace lo siguiente:

- Obtiene el **id** como un argumento.
- Utiliza **id** para ubicar el registro correcto en la tabla Members.
- Elimina ese registro.
- Redirige al usuario a la vista **index**.

Haga clic en el enlace "eliminar" de Jane Doe y vea el resultado:



## Actualización de registros

Para actualizar un registro, necesitamos el ID del registro y necesitamos un template con una interfaz que nos permita cambiar los valores.

Primero necesitamos hacer algunos cambios en la plantilla **index.html**.

### Modificar plantilla

Comience agregando un enlace para cada miembro de la tabla:

**members/templates/index.html:**

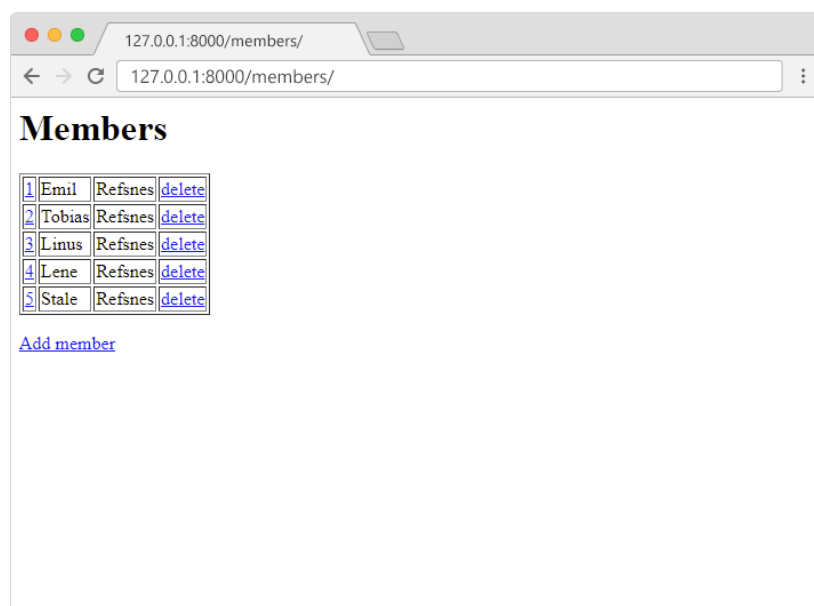
```
<h1>Members</h1>

<table border="1">
{% for x in mymembers %}
<tr>
<td><a href="update/{{ x.id }}">{{ x.id }}</a></td>
<td>{{ x.firstname }}</td>
<td>{{ x.lastname }}</td>
<td><a href="delete/{{ x.id }}">delete</a>
</tr>
{% endfor %}
</table>

<p>
<a href="add/">Add member</a>
</p>
```

El enlace va a una vista llamada **update** con el ID del miembro actual.

El resultado se verá así:





## View

A continuación, agregue la vista **update** en el archivo **members/views.py**:

**members/views.py**:

```
from django.http import HttpResponseRedirect
from django.template import loader
from django.urls import reverse
from .models import Members

def index(request):
    mymembers = Members.objects.all().values()
    template = loader.get_template('index.html')
    context = {
        'mymembers': mymembers
    }
    return HttpResponseRedirect(template.render(context, request))

def add(request):
    template = loader.get_template('add.html')
    return HttpResponseRedirect(template.render({}, request))

def addrecord(request):
    first = request.POST['first']
    last = request.POST['last']
    member = Members(firstname=first, lastname=last)
    member.save()

    return HttpResponseRedirect(reverse('index'))

def delete(request, id):
    member = Members.objects.get(id=id)
    member.delete()
    return HttpResponseRedirect(reverse('index'))

def update(request, id):
    mymember = Members.objects.get(id=id)
    template = loader.get_template('update.html')
    context = {
        'mymember': mymember,
    }
    return HttpResponseRedirect(template.render(context, request))
```

La vista `update` hace lo siguiente:

- Obtiene el **id** como un argumento.
- Utiliza **id** para ubicar el registro correcto en la tabla `Members`.
- carga una plantilla llamada **`update.html`**.
- Crea un objeto que contiene el miembro.
- Envía el objeto a la plantilla.
- Muestra el HTML que representa la plantilla.

## Nueva template

Agregue una nueva plantilla en la carpeta **`templates`**, llamada **`update.html`**:

**`members/templates/update.html`:**

```
<h1>Update member</h1>

<form action="updaterecord/{{ mymember.id }}" method="post">
{% csrf_token %}
First Name:<br>
<input name="first" value="{{ mymember.firstname }}">
<br><br>
Last Name:<br>
<input name="last" value="{{ mymember.lastname }}">
<br><br>
<input type="submit" value="Submit">
</form>
```

La plantilla contiene un formulario HTML con los valores del miembro seleccionado.

Nota: Django requiere esta línea en el formulario:

```
{% csrf_token %}
```

para manejar falsificaciones de solicitudes entre sitios en formularios donde el método es POST.

## URLs

Agregue una función **path()** en el archivo **members/urls.py**, que apunte la URL **127.0.0.1:8000/members/update/** a la ubicación correcta, con la ID como parámetro:

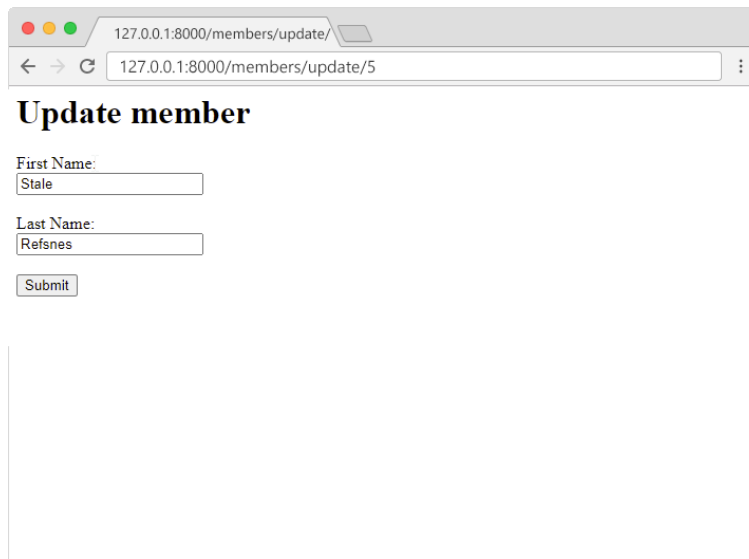
**members/urls.py:**

```
from django.urls import path

from . import views

urlpatterns = [
    path('', views.index, name='index'),
    path('add/', views.add, name='add'),
    path('add/addrecord/', views.addrecord, name='addrecord'),
    path('delete/<int:id>', views.delete, name='delete'),
    path('update/<int:id>', views.update, name='update'),
]
```

En el navegador, haga clic en la ID del miembro que desea cambiar y el resultado debería verse así:



The screenshot shows a web browser window with the address bar displaying '127.0.0.1:8000/members/update/5'. The page title is 'Update member'. Below the title, there are two text input fields: 'First Name:' with the value 'Stale' and 'Last Name:' with the value 'Refsnes'. Below these fields is a 'Submit' button.

## ¿Qué sucede al enviar?

¿Notó el atributo **action** en el formulario HTML? El atributo **action** especifica a dónde enviar los datos del formulario, en este caso los datos del formulario se enviarán a:

**updaterecord/{{ mymember.id }}**, por lo que debemos agregar una función **path()** en el archivo **members/urls.py** que apunte a la vista correcta:

**members/urls.py:**

```
from django.urls import path

from . import views

urlpatterns = [
    path('', views.index, name='index'),
    path('add/', views.add, name='add'),
    path('add/addrecord/', views.addrecord, name='addrecord'),
    path('delete/<int:id>', views.delete, name='delete'),
    path('update/<int:id>', views.update, name='update'),
    path('update/updaterecord/<int:id>', views.updaterecord, name='updaterecord'),
]
```

## Código de Actualización de Registros

Hasta ahora hemos creado la interfaz de usuario y apuntamos la URL a la vista llamada **updaterecord**, pero aún no hemos creado la vista.

Asegúrese de agregar la vista **updaterecord** en el archivo **members/views.py**:

**members/views.py:**

```
from django.http import HttpResponseRedirect
from django.template import loader
from django.urls import reverse
from .models import Members

def index(request):
    mymembers = Members.objects.all().values()
    template = loader.get_template('index.html')
    context = {
        'mymembers': mymembers,
    }
    return HttpResponseRedirect(template.render(context, request))
```

```
def add(request):
    template = loader.get_template('add.html')
    return HttpResponse(template.render({}, request))

def addrecord(request):
    x = request.POST['first']
    y = request.POST['last']
    member = Members(firstname=x, lastname=y)
    member.save()
    return HttpResponseRedirect(reverse('index'))

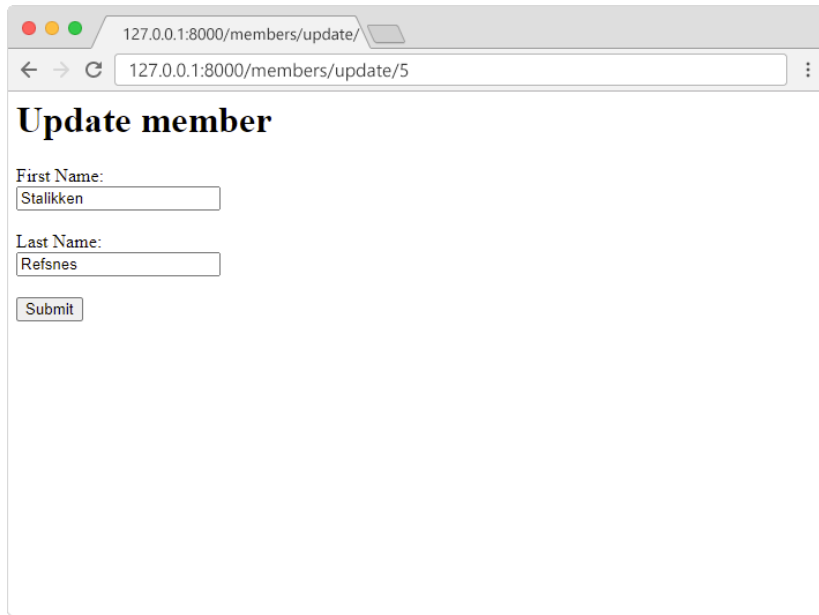
def delete(request, id):
    member = Members.objects.get(id=id)
    member.delete()
    return HttpResponseRedirect(reverse('index'))

def update(request, id):
    mymember = Members.objects.get(id=id)
    template = loader.get_template('update.html')
    context = {
        'mymember': mymember,
    }
    return HttpResponse(template.render(context, request))

def updaterecord(request, id):
    first = request.POST['first']
    last = request.POST['last']
    member = Members.objects.get(id=id)
    member.firstname = first
    member.lastname = last
    member.save()
    return HttpResponseRedirect(reverse('index'))
```

La función **updaterecord** actualizará el registro en la tabla de miembros con el ID seleccionado.

Intente actualizar un registro y vea cómo funciona:



127.0.0.1:8000/members/update/

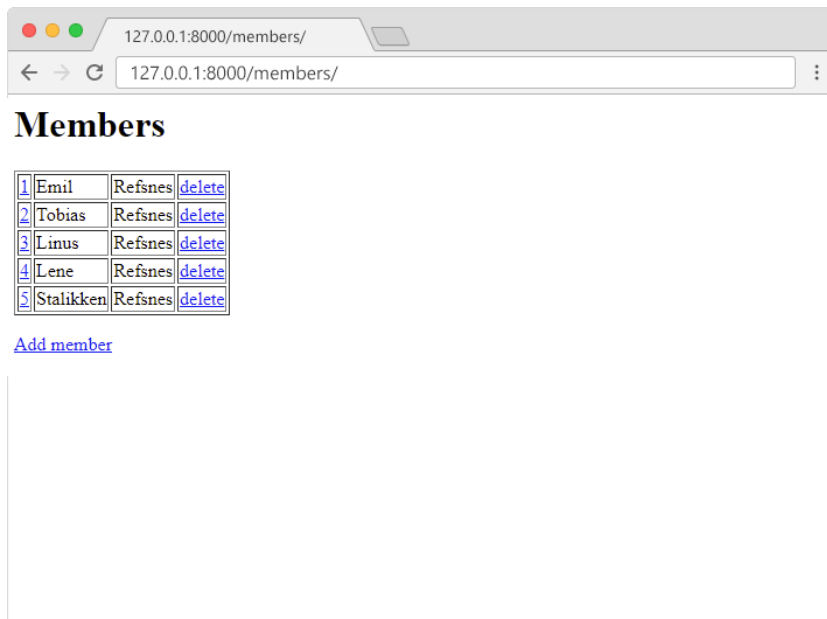
127.0.0.1:8000/members/update/5

## Update member

First Name:

Last Name:

Si presiona el botón Enviar, la tabla de miembros debería haberse actualizado:



127.0.0.1:8000/members/

127.0.0.1:8000/members/

## Members

1	Emil	Refsnes	<a href="#">delete</a>
2	Tobias	Refsnes	<a href="#">delete</a>
3	Linus	Refsnes	<a href="#">delete</a>
4	Lene	Refsnes	<a href="#">delete</a>
5	Stalikken	Refsnes	<a href="#">delete</a>

[Add member](#)

