

EXERCISES QUE TRABAJAREMOS EN EL CUE

- EXERCISE 1: AGREGANDO CAMPOS DESDE LA SHELL DE DJANGO
- EXERCISE 2: CREANDO UNA VISTA EN EL NAVEGADOR WEB
- EXERCISE 3: ELIMINAR REGISTRO DEL POST
- EXERCISE 4: ACTUALIZACIÓN DE REGISTROS

EXERCISE 1: AGREGANDO CAMPOS DESDE LA SHELL DE DJANGO

La tabla de Post se encuentra vacía, debemos agregarle algunos datos.

En este ejercicio se aprenderá 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 agreguemos algunos post a la base de datos.

Para abrir un shell de Python en Django, se escribe este comando:

```
1 $ python manage.py shell
2 Python 3.8.3 (default, Jul 2 2020, 16:21:59)
3 [GCC 7.3.0] on linux
4 Type "help", "copyright", "credits" or "license" for more information.
5 (InteractiveConsole)
6 >>>
```

Importamos la librería del modelo:

```
1 >>> from blogsite.models import Post
```

Consultamos los objetos que están en el modelo:

```
1 >>> Post.objects.all()
2 <QuerySet []>
```

Creamos un primer objeto Post:

```
1 >>> Post.objects.all()
2 <QuerySet []>
```

Lo guardamos:

```
1 >>> post.save()
```

Consultamos nuevamente los objetos que están en el modelo:

```
1 >>> Post.objects.all()
2 <QuerySet [<Post: Post object (1)>]>
```

Observamos que ya se ha creado un primer objeto. Consultamos los valores creados:

```
1 >>> Post.objects.all().values()
2 <QuerySet [{ 'id': 1, 'title': 'Titulo del Post', 'slug': 'titulo-del-
3 post', 'content': 'conte
```

Procedemos a crear varios objetos post:

```
1 >>> post1 = Post(title='Titulo del Post1', content='contenido del
2 post1', author='Juan Perez1')
3 >>> post2 = Post(title='Titulo del Post2', content='contenido del
4 post2', author='Juan Perez2')
5 >>> post3 = Post(title='Titulo del Post3', content='contenido del
6 post3', author='Juan Perez3')
7 >>> post4 = Post(title='Titulo del Post4', content='contenido del
8 post4', author='Juan Perez4')
9 post_list=[post1, post2, post3, post4]
10 >>> for post in post_list:
11 ...     post.save()
12 ...
13 >>>
```

Consultamos los valores de cada objeto:

```
1 >>> Post.objects.all().values()
2 <QuerySet [{ 'id': 1, 'title': 'Titulo del Post', 'slug': 'titulo-del-
3 post', 'content': 'contenido del post', 'created_on':
4 datetime.datetime(2022, 9, 14, 0, 19, 7, 703978,
5 tzinfo=datetime.timezone.utc), 'author': 'Juan Perez'}, { 'id': 2,
6 'title': 'Titulo del Post1', 'slug': 'titulo-del-post1', 'content':
7 'contenido del post1', 'created_on': datetime.datetime(2022, 9, 14, 0,
8 22, 24, 186423, tzinfo=datetime.timezone.utc), 'author': 'Juan
9 Perez1'}, { 'id': 3, 'title': 'Titulo del Post2', 'slug': 'titulo-del-
10 post2', 'content': 'contenido del post2', 'created_on':
11 datetime.datetime(2022, 9, 14, 0, 22, 24, 196979,
12 tzinfo=datetime.timezone.utc), 'author': 'Juan Perez2'}, { 'id': 4,
13 'title': 'Titulo del Post3', 'slug': 'titulo-del-post3', 'content':
14 'contenido del post3', 'created_on': datetime.datetime(2022, 9, 14, 0,
15 22, 24, 199202, tzinfo=datetime.timezone.utc), 'author': 'Juan
16 Perez3'}, { 'id': 5, 'title': 'Titulo del Post4', 'slug': 'titulo-del-
17 post4', 'content': 'contenido del post4', 'created_on':
18 datetime.datetime(2022, 9, 14, 0, 22, 24, 201273,
19 tzinfo=datetime.timezone.utc), 'author': 'Juan Perez4'}]>
20 >>>
```

Consultando en la bases de datos:

```

1 root=# \c project_orm_django
2
3 project_orm_django=# \d blogsite_post
4
5 project_orm_django=# SELECT * FROM blogsite_post;
6
7 id | title | slug | content | created_on | author
8 -----
9 1 | Titulo del Post | titulo-del-post | contenido del post | 2022-09-14 00:19:07.703978+00 | Juan Perez
10 2 | Titulo del Post1 | titulo-del-post1 | contenido del post1 | 2022-09-14 00:22:24.186423+00 | Juan Perez1
11 3 | Titulo del Post2 | titulo-del-post2 | contenido del post2 | 2022-09-14 00:22:24.196979+00 | Juan Perez2
12 4 | Titulo del Post3 | titulo-del-post3 | contenido del post3 | 2022-09-14 00:22:24.199202+00 | Juan Perez3
13 5 | Titulo del Post4 | titulo-del-post4 | contenido del post4 | 2022-09-14 00:22:24.201273+00 | Juan Perez4
14 (5 rows)
  
```

EXERCISE 2: CREANDO UNA VISTA EN EL NAVEGADOR WEB

Deseamos observar el resultado en una página web, no en un entorno de shell de Python. Para ello debemos crear una vista. En la aplicación blogsite, abra el archivo `views.py`, y agregue el siguiente contenido:

BLOGSITE/VIEWS.PY

```

1 from django.shortcuts import render
2 from django.http import HttpResponse
3 from django.template import loader
4 from .models import Post
5
6 def index(request):
7     posts = Post.objects.all().values()
8     template = loader.get_template('index.html')
9     context = {
10         'posts': posts,
11     }
12     return HttpResponse(template.render(context, request))
  
```

Creemos la plantilla en `blogsite/templates/index.html`:

```

1 <h1>Posts</h1>
2
3 <table border="1">
4 <tr>
5 <td>id</td>
6 <td>title</td>
7 <td>slug</td>
8 <td>content</td>
9 <td>created_on</td>
10 <td>author</td>
11 </tr>
  
```

```
12 {% for post in posts %}
13 <tr>
14 <td>{{ post.id }}</td>
15 <td>{{ post.title }}</td>
16 <td>{{ post.slug }}</td>
17 <td>{{ post.content }}</td>
18 <td>{{ post.created_on }}</td>
19 <td>{{ post.author }}</td>
20 </tr>
21 {% endfor %}
22 </table>
```

Creamos el archivo `urls.py` en la aplicación blogsite:

BLOGSITE/URLS.PY

```
1 from django.urls import path
2 from . import views
3
4 urlpatterns = [
5     path('', views.index, name='index'),
6 ]
```

Agregamos el `path` en la URLs del proyecto:

CONFIG/URLS.PY

```
1 from django.contrib import admin
2 from django.urls import path, include
3
4 urlpatterns = [
5     path('admin/', admin.site.urls),
6     path('post/', include('blogsite.urls'))
7 ]
```

Adecuamos el archivo `setting.py` del proyecto, la aplicación y ajustador en el directorio de los templates:

CONFIG/SETTINGS.PY

```
1 # Application definition
2
3 INSTALLED_APPS = [
4     'django.contrib.admin',
5     'django.contrib.auth',
6     'django.contrib.contenttypes',
7     'django.contrib.sessions',
8     'django.contrib.messages',
9     'django.contrib.staticfiles',
10    'blogsite.apps.BlogsiteConfig',
11 ]
12
```

```

13
14
15 TEMPLATES = [
16     {
17         'BACKEND': 'django.template.backends.django.DjangoTemplates',
18         'DIRS': [BASE_DIR / 'templates'],
19         'APP_DIRS': True,
20         'OPTIONS': {
21             'context_processors': [
22                 'django.template.context_processors.debug',
23                 'django.template.context_processors.request',
24                 'django.contrib.auth.context_processors.auth',
25                 'django.contrib.messages.context_processors.messages',
26             ],
27         },
28     },
29 ]
30
31 # Configuramos el archivo setting.py del proyecto, adecuando la aplicación y
32 # ajustador el directorio de los templates:
  
```

Finalmente, ingresamos a <http://localhost:8000/post/>:

Posts

id	title	slug	content	created_on	author
1	Titulo del Post	titulo-del-post	contenido del post	Sept. 14, 2022, 12:19 a.m.	Juan Perez
2	Titulo del Post1	titulo-del-post1	contenido del post1	Sept. 14, 2022, 12:22 a.m.	Juan Perez1
3	Titulo del Post2	titulo-del-post2	contenido del post2	Sept. 14, 2022, 12:22 a.m.	Juan Perez2
4	Titulo del Post3	titulo-del-post3	contenido del post3	Sept. 14, 2022, 12:22 a.m.	Juan Perez3
5	Titulo del Post4	titulo-del-post4	contenido del post4	Sept. 14, 2022, 12:22 a.m.	Juan Perez4

AGREGANDO REGISTROS AL POST

Hasta ahora, se ha creado una tabla de Post en la 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.

Queremos poder crear nuevos posts desde una página web. Adecuamos el template:

blogsite/template/index.html

```
1 <h1>Posts</h1>
```

```
2
3 <table border="1">
4 <tr>
5     <td>id</td>
6     <td>title</td>
7     <td>slug</td>
8     <td>content</td>
9     <td>created_on</td>
10    <td>author</td>
11 </tr>
12 {% for post in posts %}
13 <tr>
14     <td>{{ post.id }}</td>
15     <td>{{ post.title }}</td>
16     <td>{{ post.slug }}</td>
17     <td>{{ post.content }}</td>
18     <td>{{ post.created_on }}</td>
19     <td>{{ post.author }}</td>
20 </tr>
21 {% endfor %}
22 </table>
23 <p>
24     <a href="agregar/">Agregar Post</a>
25 </p>
```

Creemos una nueva plantilla para agregar Post, crearemos el template `agregar.html`:

`blogsite/template/agregar.html`

```
1 <h1>Agregar Post</h1>
2
3 <form action="agregarregistro/" method="post">
4     {% csrf_token %}
5     Titulo:<br>
6     <input name="title">
7     <br><br>
8     Contenido:<br>
9     <input name="content">
10    <br><br>
11    Autor:<br>
12    <input name="author">
13    <br><br>
14    <input type="submit" value="Submit">
15 </form>
```

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

Django requiere esta línea en el formulario:

```
1 {% csrf_token %}
```

Para manejar falsificaciones de solicitudes entre sitios en formularios donde el método es POST, procedemos a adecuar la Vista, agregando el método:

BLOGSITE/VIEWS.PY

```
1 def agregar(request):
2     template = loader.get_template('agregar.html')
3     return HttpResponse(template.render({}, request))
```

Agregamos la ruta en la URLs:

BLOGSITE/URLS.PY

```
1 from django.urls import path
2 from . import views
3
4 urlpatterns = [
5     path('', views.index, name='index'),
6     path('agregar/', views.agregar, name='agregar'),
7 ]
```

El atributo **action** especifica dónde enviar los datos del formulario; en este caso, se enviarán a **agregarregistro/**, por lo que debemos agregar una función **path()** en el archivo **blogsite/urls.py**.

BLOGSITE/URLS.PY

```
1 from django.urls import path
2 from . import views
3
4 urlpatterns = [
5     path('', views.index, name='index'),
6     path('agregar/agregarregistro/', views.agregarregistro,
7 name='agregar'),
8 ]
```

Adecuamos la vista para agregar el método **agregarregistro** a la misma:

BLOGSITE/VIEWS.PY

```
1 def agregarregistro(request):
2     titulo = request.POST['title']
3     contenido = request.POST['content']
4     autor = request.POST['author']
5     post = Post(title=titulo, content=contenido, author=autor)
6     post.save()
7     return HttpResponseRedirect(reverse('index'))
```

Cambios que se realizan en el archivo `views.py`:

- Importar `HttpResponseRedirect`.
- Importar `reverse`.

El método `agregarregistro`:

1. Obtiene el título, contenido, y autor del post con una instrucción `request.POST`.
2. Agrega un nuevo registro en la tabla de Post.
3. Redirige al usuario a la vista del indexa.

Seguidamente, procedemos a agregar un nuevo Post en <http://localhost:8000/post/agregar/>:

Agregar Post

Título:

Contenido:

Autor:

Posts

id	title	slug	content	created_on	author
1	Titulo del Post	titulo-del-post	contenido del post	Sept. 14, 2022, 12:19 a.m.	Juan Perez
2	Titulo del Post1	titulo-del-post1	contenido del post1	Sept. 14, 2022, 12:22 a.m.	Juan Perez1
3	Titulo del Post2	titulo-del-post2	contenido del post2	Sept. 14, 2022, 12:22 a.m.	Juan Perez2
4	Titulo del Post3	titulo-del-post3	contenido del post3	Sept. 14, 2022, 12:22 a.m.	Juan Perez3
5	Titulo del Post4	titulo-del-post4	contenido del post4	Sept. 14, 2022, 12:22 a.m.	Juan Perez4
6	Titulo 2	titulo-2	Contenido 2	Sept. 14, 2022, 2:28 a.m.	Juan Cuerdo
8	Titulo 3	titulo-3	Contenido 3	Sept. 14, 2022, 2:30 a.m.	Juan Cuerdo
9	Titulo del Post 5	titulo-del-post-5	Contenido del Post 6	Sept. 14, 2022, 2:35 a.m.	Juan Carlos Cuerdo

[Agregar Post](#)

EXERCISE 3: ELIMINAR REGISTRO DEL POST

Para eliminar un registro no necesitamos una nueva plantilla, pero si hacer algunos cambios en la plantilla de post, `index.html`.

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.

Modificamos la plantilla `index`:

BLOGSITE/TEMPLATE/INDEX.HTML

```

1 <h1>Posts</h1>
2
3 <table border="1">
4 <tr>
5   <td>id</td>
6   <td>title</td>
7   <td>slug</td>
8   <td>content</td>
9   <td>created_on</td>
10  <td>author</td>
11 </tr>
12 {% for post in posts %}
13 <tr>
14   <td>{{ post.id }}</td>
15   <td>{{ post.title }}</td>
16   <td>{{ post.slug }}</td>
17   <td>{{ post.content }}</td>
18   <td>{{ post.created_on }}</td>
19   <td>{{ post.author }}</td>

```

```
20 <td><a href="eliminar/{{ post.id }}">eliminar</a></td>
21 </tr>
22 {% endfor %}
23 </table>
24 <p>
25 <a href="agregar/">Agregar Post</a>
26 </p>
```

Se agrega la URL:

BLOGSITE/URLS.PY

```
1 from django.urls import path
2 from . import views
3
4 urlpatterns = [
5     path('', views.index, name='index'),
6     path('agregar/', views.agregar, name='agregar'),
7     path('agregar/agregarregistro/', views.agregarregistro,
8     name='agregarregistro'),
9     path('eliminar/<int:id>', views.eliminar, name='eliminar'),
10 ]
```

Agregamos el método eliminar en la vista:

BLOGSITE/VIEWS.PY

```
1 def eliminar(request, id):
2     post = Post.objects.get(id=id)
3     post.delete()
4     return HttpResponseRedirect(reverse('index'))
5 ]
```

La vista de eliminación hace lo siguiente:

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

Al realizar una eliminación nos dirigimos a <http://localhost:8000/post/>:

Y observamos:

Posts

id	title	slug	content	created_on	author	
1	Titulo del Post	titulo-del-post	contenido del post	Sept. 14, 2022, 12:19 a.m.	Juan Perez	eliminar
2	Titulo del Post1	titulo-del-post1	contenido del post1	Sept. 14, 2022, 12:22 a.m.	Juan Perez1	eliminar
3	Titulo del Post2	titulo-del-post2	contenido del post2	Sept. 14, 2022, 12:22 a.m.	Juan Perez2	eliminar
4	Titulo del Post3	titulo-del-post3	contenido del post3	Sept. 14, 2022, 12:22 a.m.	Juan Perez3	eliminar
5	Titulo del Post4	titulo-del-post4	contenido del post4	Sept. 14, 2022, 12:22 a.m.	Juan Perez4	eliminar
9	Titulo del Post 5	titulo-del-post-5	Contenido del Post 6	Sept. 14, 2022, 2:35 a.m.	Juan Carlos Cuerdo	eliminar

[Agregar Post](#)

EXERCISE 4: ACTUALIZACIÓN DE REGISTROS

Para actualizar un registro, necesitamos su **ID** y una plantilla con interfaz que nos permita cambiar los valores.

Primero, debemos hacer algunos cambios en la plantilla index.html:

BLOGSITE/TEMPLATE/INDEX.HTML

```

1 <h1>Posts</h1>
2
3 <table border="1">
4 <tr>
5   <td>id</td>
6   <td>title</td>
7   <td>slug</td>
8   <td>content</td>
9   <td>created_on</td>
10  <td>author</td>
11 </tr>
12 {% for post in posts %}
13 <tr>
14   <td><a href="actualizar/{{ post.id }}">{{ post.id }}</td>
15   <td>{{ post.title }}</td>
16   <td>{{ post.slug }}</td>
17   <td>{{ post.content }}</td>
18   <td>{{ post.created_on }}</td>
19   <td>{{ post.author }}</td>
20   <td><a href="eliminar/{{ post.id }}">eliminar</a></td>
21 </tr>
22 {% endfor %}
23 </table>
24 <p>
25   <a href="agregar/">Agregar Post</a>
  
```

```
26 </p>
```

Adecuamos la vista, agregando el método de actualizar:

BLOGSITE/VIEWS.PY

```
1 def actualizar(request, id):
2     post = Post.objects.get(id=id)
3     template = loader.get_template('actualizar.html')
4     context = {
5         'post': post,
6     }
7     return HttpResponse(template.render(context, request))
```

El método actualizar:

- Obtiene el **id** como un argumento.
- Utiliza **id** para ubicar el registro correcto en la tabla Post.
- Carga una plantilla llamada **actualizar.html**.
- Crea un objeto que contiene el post.
- Envía el objeto a la plantilla.
- Muestra el HTML que representa la plantilla.

Creemos la plantilla para actualizar:

BLOGSITE/TEMPLATES/ACTUALIZAR.HTML

```
1 <h1>Actualizar Post</h1>
2
3 <form action="actualizarregistro/{{ post.id }}" method="post">
4     {% csrf_token %}
5     Titulo:<br>
6     <input name="title" value="{{ post.title }}">
7     <br><br>
8     Contenido:<br>
9     <input name="content" value="{{ post.content }}">
10    <br><br>
11    Autor:<br>
12    <input name="author" value="{{ post.author }}">
13    <br><br>
14    <input type="submit" value="Submit">
15 </form>
```

Agregamos la URLs:

BLOGSITE/URLS.PY

```
1 from django.urls import path
2 from . import views
3
4 urlpatterns = [
5     path('', views.index, name='index'),
6     path('agregar/', views.agregar, name='agregar'),
7     path('agregar/agregarregistro/', views.agregarregistro,
8     name='agregarregistro'),
9     path('eliminar/<int:id>', views.eliminar, name='eliminar'),
10    path('actualizar/<int:id>', views.actualizar, name='actualizar'),
11    path('actualizarregistro/<int:id>',
12         views.actualizarregistro,
13         name='actualizarregistro'),
14 ]
```

Adecuamos el nuevo método de **actualizarregistro** en la vista:

BLOGSITE/VIEWS.PY

```
1 def actualizarregistro(request, id):
2     titulo = request.POST['title']
3     contenido = request.POST['content']
4     autor = request.POST['author']
5     post = Post.objects.get(id=id)
6     post.title = titulo
7     post.content = contenido
8     post.author = autor
9     post.save()
10    return HttpResponseRedirect(reverse('index'))
```

Finalmente, procedemos a actualizar algún registro <http://localhost:8000/post/>:

Posts

id	title	slug	content	created_on	author	
1	Titulo del Post 1	titulo-del-post	contenido del post1	Sept. 14, 2022, 12:19 a.m.	Juan Perez	eliminar
2	Titulo del Post	titulo-del-post1	contenido del post	Sept. 14, 2022, 12:22 a.m.	Juan Perez	eliminar
3	Titulo del Post2	titulo-del-post2	contenido del post2	Sept. 14, 2022, 12:22 a.m.	Juan Perez2	eliminar
4	Titulo del Post3	titulo-del-post3	contenido del post3	Sept. 14, 2022, 12:22 a.m.	Juan Perez3	eliminar
5	Titulo del Post4	titulo-del-post4	contenido del post4	Sept. 14, 2022, 12:22 a.m.	Juan Perez4	eliminar
2	Titulo del Post 5	titulo-del-post-5	Contenido del Post 6	Sept. 14, 2022, 2:35 a.m.	Juan Carlos Cuerdo	eliminar

[Agregar Post](#)

Al seleccionar algún **id**, podemos actualizar el registro:

Actualizar Post

Título:

Contenido:

Autor: