



## EXERCISES QUE TRABAJAREMOS EN EL CUE

- EXERCISE 1: SISTEMA DE AUTENTICACIÓN DE DJANGO.
- EXERCISE 2: CREACIÓN DE UNA PÁGINA DE REGISTRO DE USUARIOS.
- EXERCISE 3: INICIO DE SESIÓN.
- EXERCISE 4: CIERRE DE SESIÓN DE UN USUARIO.

## EXERCISE 1: SISTEMA DE AUTENTICACIÓN DE DJANGO.

El objetivo del presente ejercicio es plantear una guía paso a paso para el registro, inicio de sesión y validación de usuario en el Framework Django.

Requisitos previos:

- Tener conocimiento de una terminal o línea de comando e instalación de paquetes de software en el sistema operativo, tanto en Windows 10 como en Linux (en este caso Linux Ubuntu).
- En el caso de Windows, se debe tener instalado el entorno virtual explicado en el Exercises del CUE03 y la configuración del proyecto en el CUE04.
- Tener previamente instalado la versión de Python 3 y el entorno virtual con (virtualenvwrapper).
- Hacer uso de la herramienta Visual Studio Code.

## SISTEMA DE AUTENTICACIÓN DE DJANGO

Para esta práctica vamos a considerar el entorno virtual `project_django`, y dentro creamos el proyecto `site_web_django` y una aplicación boards. Para abrir el proyecto nos ubicamos en el directorio `site_web_django`, y lo hacemos con VSC:

```
1 $ cd site_web_django
2 $ code .
```

Activamos el proyecto con `workon`:

```
1 $ workon projects_django
```

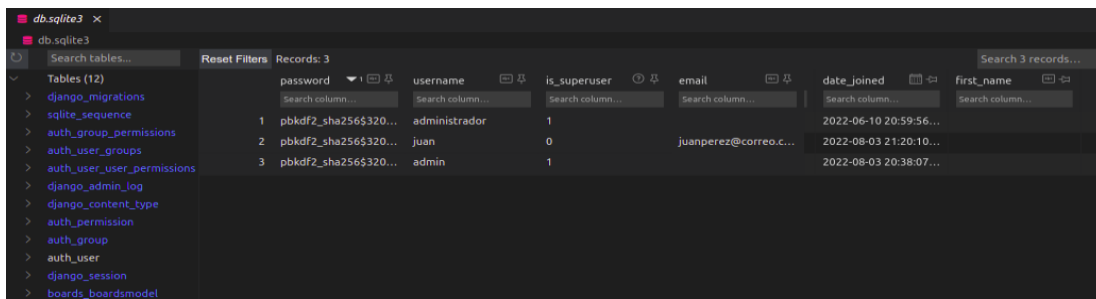
Procedemos a iniciar la terminal en VS Code, y empezamos la shell de Django:

```
1 $ python manage.py shell
2
3 Python 3.8.3 (default, Jul  2 2020, 16:21:59)
4 [GCC 7.3.0] on linux
5 Type "help", "copyright", "credits" or "license" for more information.
6 (InteractiveConsole)
7 >>>
```

La autenticación Django proporciona tanto la autenticación como la autorización juntas, y se suele denominar sistema de autenticación, ya que estas características están algo acopladas. Procedemos a crear un usuario en la terminal para el framework Django, con el nombre de juan, correo [juanperez@correo.com](mailto:juanperez@correo.com) y contraseña juanperez, respectivamente.

```
1 >>> from django.contrib.auth.models import User
2 >>> user = User.objects.create_user('juan', 'juanperez@correo.com',
3 'juanperez')
```

Verificamos si se ha creado el usuario. Por defecto, estamos usando como gestor de base de datos SQLite, con una extensión de VS Code podremos revisar la estructura de la base de datos dicha extensión es SQLite Viewer.



| Tables (12)                  | password | username | is_superuser | email | date_joined | first_name |
|------------------------------|----------|----------|--------------|-------|-------------|------------|
| > django_migrations          |          |          |              |       |             |            |
| > sqlite_sequence            |          |          |              |       |             |            |
| > auth_group_permissions     |          |          |              |       |             |            |
| > auth_user_groups           |          |          |              |       |             |            |
| > auth_user_user_permissions |          |          |              |       |             |            |
| > django_admin_log           |          |          |              |       |             |            |
| > django_content_type        |          |          |              |       |             |            |
| > auth_permission            |          |          |              |       |             |            |
| > auth_group                 |          |          |              |       |             |            |
| > auth_user                  |          |          |              |       |             |            |
| > django_session             |          |          |              |       |             |            |
| > boards_boardsmodel         |          |          |              |       |             |            |

|   | password              | username      | is_superuser | email                 | date_joined            | first_name |
|---|-----------------------|---------------|--------------|-----------------------|------------------------|------------|
| 1 | pbkdf2_sha256\$320... | administrador | 1            |                       | 2022-06-10 20:59:56... |            |
| 2 | pbkdf2_sha256\$320... | juan          | 0            | juanperez@correo.c... | 2022-08-03 21:20:10... |            |
| 3 | pbkdf2_sha256\$320... | admin         | 1            |                       | 2022-08-03 20:38:07... |            |

Procedemos a cambiar la contraseña del usuario juan:

```
1 >>> u = User.objects.get(username='juan')
2 >>> u.set_password('123456')
3 >>> u.save()
```

Podemos manipular el objeto user con sus atributos. Por ejemplo: su correo, password cifrado, entre otros.

```
1 >>>
2 print(u.password)
3 pbkdf2_sha256$320000$K2v7EqrJ59r1GeeSdqVltN$71NV5D1UF6NmDCHGozzXGwGfCixc77B
4 pxYeXxdIIcU=
5 >>> print(u.email)
6 juanperez@correo.com
7 >>> print(u.last_name)
```

## AUTENTICANDO EL USUARIO.

Se utiliza `authenticate()` para verificar un conjunto de credenciales. Toma las credenciales como argumentos de palabras clave, username y password para el caso predeterminado, las compara con cada backend de autenticación y devuelve un objeto `User`.

```
1 >>> from django.contrib.auth import authenticate
2 >>> user = authenticate(username='juan', password='12345')
3 print(user)
4 None
5
6 >>> >>> user = authenticate(username='juan', password='123456')
7 print(user)
8 juan
```

Consultar la documentación de Django para más detalle sobre los atributos y métodos en:

<https://docs.djangoproject.com/en/4.1/ref/contrib/auth/#django.contrib.auth.models.User>

## EXERCISE 2: CREACIÓN DE UNA PÁGINA DE REGISTRO DE USUARIOS.

Procedemos a instalar `django-crispy-forms`, que implementa una clase llamada FormHelper encargada de definir el comportamiento de representación del formulario. Esto permite controlar los atributos del formulario y su diseño, haciéndolo de manera programática usando Python. De esta manera, se escribe la menor cantidad de HTML posible y toda su lógica permanece en los formularios y los archivos de vistas.

```
1 pip install crispy-bootstrap5
```

Agregamos en el `setting.py` la aplicación:

#### SETTINGS.PY

```
1 .....
2 INSTALLED_APPS = [
3     'django.contrib.admin',
4     'django.contrib.auth',
5     'django.contrib.contenttypes',
6     'django.contrib.sessions',
7     'django.contrib.messages',
8     'django.contrib.staticfiles',
9     'boards.apps.BoardsConfig',
10    'bootstrap5',
11    'crispy_forms',
12    "crispy_bootstrap5",
13 ]
14
15 CRISPY_ALLOWED_TEMPLATE_PACKS = "bootstrap5"
16 CRISPY_TEMPLATE_PACK = "bootstrap5"
17 .....
```

Procedemos a crear la una nueva clase de formulario de registro, partiendo del modelo de usuario que contiene Django.

#### BOARDS/FORMS.PY

```
1 from django import forms
2 from .models import BoardsModel
3 # Agregando para el registro de usuarios
4 from django.contrib.auth.forms import UserCreationForm
5 from django.contrib.auth.models import User
6
7 # Agregado para el registro de usuarios
8 class RegistroUsuarioForm(UserCreationForm):
9     email = forms.EmailField(required=True)
10
11     class Meta:
12         model = User
13         fields = ("username", "email", "password1", "password2")
14
15     def save(self, commit=True):
16         user = super(RegistroUsuarioForm, self).save(commit=False)
17         user.email = self.cleaned_data['email']
```

```
18         if commit:
19             user.save()
20         return user
```

Django viene con un formulario de registro preconstruido llamado `UserCreationForm`, que se conecta al usuario del modelo preconstruido. Sin embargo, `UserCreationForm` solo requiere un nombre de usuario y una contraseña (la contraseña 1 es la contraseña inicial, y la contraseña 2 es la confirmación de la contraseña). Para personalizar el formulario preconstruido, primero cree un nuevo archivo llamado `forms.py` en el directorio de la aplicación. Este nuevo archivo se crea en el mismo directorio que `models.py` y `views.py`.

Luego, llame a `UserCreationForm` dentro de una nueva clase llamada `RegistroUsuarioForm`, y agregue otro campo llamado `email`. Guarde el correo electrónico para el usuario. Procedemos a crear la plantilla `html` para el formulario de registro. A continuación, cree una plantilla con el nombre `registro.html`, dentro de `boards/templates/registration/`.

El formulario usa Bootstrap y Django `crispy-forms` para algunos CSS agregados. También extiende el archivo `header.html`, que se encuentra en cómo usar las etiquetas de plantilla de Django extendidas e incluidas.

#### BOARDS/TEMPLATES/REGISTRATION/REGISTRO.HTML

```
1 {% include "menu.html" %} {% block content %} {% load crispy_forms_tags %}
2
3 <!--Registro-->
4 <div class="container py-5">
5     <h1>Registro</h1>
6     <form method="POST">
7         {% csrf_token %} {{ register_form|crispy }}
8         <button class="btn btn-primary" type="submit">Register</button>
9     </form>
10    <p class="text-center">
11        Si tienes actualmente una cuenta, inicie su sesión en
12        <a href="/login">login</a>.
13    </p>
```

```
14 </div>
15
16 {% endblock %}
```

Procedemos a agregar la ruta de registro a las URL de la aplicación para que podamos consultarla en las vistas.

### BOARDS/URLS.PY

```
1 from django.urls import path
2
3 from .views import IndexPageView, obtenerFecha, menuView, mostrar,
4 datosform_wiew, widget_view, boardsform_view, registro_view
5
6 urlpatterns = [
7     path('', IndexPageView.as_view(), name='index'),
8     path('fecha/<name>', obtenerFecha, name='index'),
9     path('menu/', menuView, name='menu'),
10    path('mostrar/', mostrar, name='mostrar'),
11    path('datosform/', datosform_wiew, name='datos_form'),
12    path('widgetform/', widget_view, name='widgetform'),
13    path('boardsform/', boardsform_view, name='boardsform'),
14    path('registro/', registro_view, name="registro"),
15 ]
```

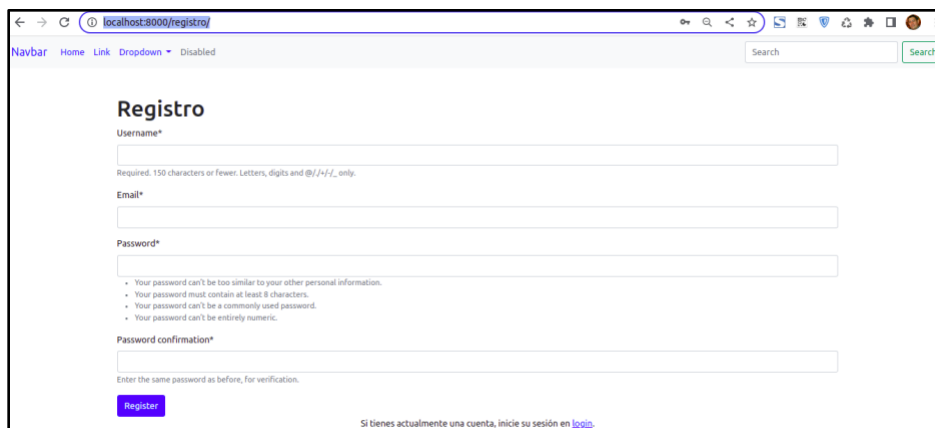
Seguidamente, agregamos las funciones de la vista en `view.py`. Importamos `RegistroUsuarioForm` desde `forms.py`, e iniciamos sesión desde `django.contrib.auth`. Luego escriba una nueva función de vista llamada `registro_view`. Hay dos sentencias `if/else` dentro de la función. El primero verifica si el formulario se está publicando, mientras que el segundo verifica si el formulario es válido. Si ambos son verdaderos, la información del formulario se guarda con un usuario, el usuario inicia sesión y se le redirige a la página de inicio que muestra un mensaje de éxito.

De lo contrario, si el formulario no es válido, se muestra un mensaje de error. Pero si la solicitud no es un POST en primer lugar, lo que significa que la primera declaración `if` devolvió `False`, represente el formulario vacío en la plantilla de registro. Tenga en cuenta que, si desea agregar mensajes a su proyecto Django, debe habilitar Messages Framework e importar mensajes en la parte superior de `views.py`.

## BOARDS/VIEWS.PY

```
1 from django.http import HttpResponseRedirect
2 from tokenize import PseudoExtras
3 from django.views.generic import TemplateView
4 from django.shortcuts import render
5 import datetime
6 from .forms import InputForm, WidgetForm, BoardsForm, RegistroUsuarioForm
7 from django.contrib import messages
8 from django.contrib.auth import login
9
10 def registro_view(request):
11     if request.method == "POST":
12         form = RegistroUsuarioForm(request.POST)
13         if form.is_valid():
14             user = form.save()
15             login(request, user)
16             messages.success(request, "Registrado
17 Satisfactoriamente." )
18
19         return HttpResponseRedirect('/menu')
20         messages.error(request, "Registro invalido. Algunos datos
21 ingresados no son correctos")
22
23         form = RegistroUsuarioForm()
24         return render (request=request,
25 template_name="registration/registro.html",
26 context={"register_form":form})
```

Procedemos a verificar el registro de un usuario en la aplicación, en: <http://localhost:8000/registro/>



The screenshot shows a web browser window with the address bar displaying `localhost:8000/registro/`. The page has a light gray header with a "Navbar" containing "Home", "Link", and "Dropdown" (disabled), and a search bar. The main content area is titled "Registro" and contains a registration form. The form has four input fields: "Username\*", "Email\*", "Password\*", and "Password confirmation\*". Below the "Password\*" field, there are four bullet points providing password requirements: "Your password can't be too similar to your other personal information.", "Your password must contain at least 8 characters.", "Your password can't be a commonly used password.", and "Your password can't be entirely numeric.". Below the "Password confirmation\*" field, there is a note: "Enter the same password as before, for verification.". At the bottom left of the form is a purple "Register" button. At the bottom right, there is a link: "Si tienes actualmente una cuenta, inicie su sesión en [login](#)."

```
1 $ python manage.py shell
2
3 Python 3.8.3 (default, Jul 2 2020, 16:21:59)
4 [GCC 7.3.0] on linux
5 Type "help", "copyright", "credits" or "license" for more information.
6
7 (InteractiveConsole)
8 >>>
9
```

### EXERCISE 3: INICIO DE SESIÓN.

Ya hemos reutilizado partiendo del modelo de usuario la creación de la vista, que registra a un usuario al crear la cuenta. Queremos que el usuario tenga la capacidad de iniciar sesión libremente. Por lo tanto, necesitamos una plantilla de inicio de sesión, una URL y una función de vistas.

La estructura básica de la plantilla HTML es similar a la del registro HTML. Las únicas diferencias son el formulario y el enlace en la parte inferior.

#### BOARDS/TEMPLATES/REGISTRATION/LOGIN.HTML

```
1 {% include "menu.html" %} {% block content %} {% load crispy_forms_tags %}
2 <!--Login-->
3 <div class="container py-5">
4     <h1>Login</h1>
5     <form method="POST">
6         {% csrf_token %}
7         {{ login_form|crispy }}
8         <button class="btn btn-primary" type="submit">Login</button>
9     </form>
10    <p class="text-center">Si nio tienes una cuenta, regístrate en <a
11 href="/registro">, para crear una cuenta</a>.</p>
12 </div>
13 {% endblock %}
```

Agregamos la URL.

#### BOARDS/URL.PY



```
1 from django.urls import path
2 from .views import IndexPageView, obtenerFecha, menuView, mostrar,
3   datosform_wiew, widget_view, boardsform_view, registro_view, login_view
4 urlpatterns = [
5     path('', IndexPageView.as_view(), name='index'),
6     path('fecha/<name>', obtenerFecha, name='index'),
7     path('menu/', menuView, name='menu'),
8     path('mostrar/', mostrar, name='mostrar'),
9     path('datosform/', datosform_wiew, name='datos_form'),
10    path('widgetform/', widget_view, name='widgetform'),
11    path('boardsform/', boardsform_view, name='boardsform'),
12    path('registro/', registro_view, name="registro"),
13    path('login/', login_view, name="login"),
14 ]
```

Agregamos a funciones en la vista, creado la vista **login\_view**. En el archivo views.py se agrega la autenticación a la lista de importaciones desde **django.contrib.auth**, luego importe **AuthenticationForm** desde **django.contrib.auth.forms** en la parte superior del archivo. **AuthenticationForm** es el formulario de Django preconstruido que inicia sesión en un usuario.

Para escribir su función de inicio de sesión, agregue una declaración if/else que use la función **autenticar()** de Django. Esta función se utiliza para verificar las credenciales del usuario (nombre de usuario y contraseña) y devolver el objeto Usuario correcto almacenado en el backend.

Si el backend autenticó las credenciales, la función ejecutará Django **login()** para iniciar sesión en el usuario autenticado. De lo contrario, si el usuario no está autenticado, devuelve un mensaje al usuario que indica que ingresó un nombre de usuario o contraseña no válidos.

La segunda declaración else es si el formulario no es válido, entonces devuelve un mensaje de error similar. La declaración else final es si la solicitud no es un POST, luego devuelva el formulario en blanco en la plantilla HTML de inicio de sesión.

## BOARDS/VIEWS.PY

```
1 from django.http import HttpResponseRedirect
```

```
2 from tokenize import PseudoExtras
3 from django.views.generic import TemplateView
4 from django.shortcuts import render
5 import datetime
6 from .forms import InputForm, WidgetForm, BoardsForm, RegistroUsuarioForm
7 from django.contrib import messages
8 from django.contrib.auth import login, authenticate
9 from django.contrib.auth.forms import AuthenticationForm
10
11 def login_view(request):
12     if request.method == "POST":
13         form = AuthenticationForm(request, data=request.POST)
14         if form.is_valid():
15             username = form.cleaned_data.get('username')
16             password = form.cleaned_data.get('password')
17             user = authenticate(username=username,
18 password=password)
19
20 if user is not None:
21
22     login(request, user)
23
24     messages.info(request, f"Iniciaste sesión como: {username}.")
25
26     return HttpResponseRedirect('/menu')
27
28     else:
29
30     messages.error(request, "Invalido username o password.")
31
32     else:
33
34     messages.error(request, "Invalido username o password.")
35
36     form = AuthenticationForm()
37
38     return render(request=request, template_name="registration/login.html",
39 context={"login_form":form})
40
```

Ahora procedemos a ir a la URL de inicio de sesión, <http://127.0.0.1:8000/login>, e inicie sesión con el usuario de prueba.

## Login

Username\*

admin

Password\*

\*\*\*\*\*

Login

Si nio tienes una cuenta, regístrate en [para crear una cuenta.](#)

## EXERCISE 4: CIERRE DE SESIÓN DE UN USUARIO.

Finalmente, lo que tenemos que manejar es el cierre de sesión del usuario. Colocaremos el enlace de cierre de sesión en la barra de navegación, pero el usuario solo lo verá si está autenticado (es decir, si ha iniciado sesión).

Actualizamos el template menu.html, agregando al final del archivo el siguiente código:

### TEMPLATE/MENU.HTML

```
1 {% extends 'base.html' %} {% load bootstrap5 %} {% block content %}
2 .....
3 .....
4 <div class="collapse navbar-collapse" id="navbarText">
5   <ul class="navbar-nav mr-auto">
6     {% if user.is_authenticated %}
7
8     <li class="nav-item">
9       <a class="nav-link" href="/logout">Salir</a>
10    </li>
11    <li class="nav-item">
12      <a class="nav-link" href="#">Hola, {{user.username}}</a>
13    </li>
14
15    {% else %}
16
17    <li class="nav-item">
18      <a class="nav-link" href="/login">Login</a>
```

```
19         </li>
20
21     {% endif %}
22 </ul>
23 </div>
24 </nav>
25 {% endblock %}
```

La variable de plantilla de Django `{{ usuario }}` almacena el usuario conectado actual, y sus permisos están disponibles en el contexto de la plantilla. Todo lo que tenemos que hacer es agregar una declaración `if/else` que si el usuario está autenticado muestre el enlace de cierre de sesión y su nombre de usuario, de lo contrario muestre un enlace de inicio de sesión.

Agregamos la URL.

#### BOARDS/URL.PY

```
1 from django.urls import path
2
3 from .views import IndexPageView, obtenerFecha, menuView, mostrar,
4 datosform_view, widget_view, boardsform_view, registro_view, login_view,
5 logout_view
6 urlpatterns = [
7     path('', IndexPageView.as_view(), name='index'),
8     path('fecha/<name>', obtenerFecha, name='index'),
9     path('menu/', menuView, name='menu'),
10    path('mostrar/', mostrar, name='mostrar'),
11    path('datosform/', datosform_view, name='datos_form'),
12    path('widgetform/', widget_view, name='widgetform'),
13    path('boardsform/', boardsform_view, name='boardsform'),
14    path('registro/', registro_view, name="registro"),
15    path('logout/', logout_view, name="logout"),
16 ]
```

Agregamos la funcionalidad a la vista:

#### BOARDS/VIEWS.PY

```
1 from django.http import HttpResponseRedirect
```



```
2 from tokenize import PseudoExtras
3 from django.views.generic import TemplateView
4 from django.shortcuts import render
5 import datetime
6 from .forms import InputForm, WidgetForm, BoardsForm, RegistroUsuarioForm
7 from django.contrib import messages
8 from django.contrib.auth import login, authenticate, logout
9
10 from django.contrib.auth.forms import AuthenticationForm
11
12 def logout_view(request):
13     logout(request)
14     messages.info(request, "Se ha cerrado la sesión
15 satisfactoriamente.")
16     return HttpResponseRedirect('/menu')
```

Finalmente, se agregó una función de cierre de sesión al archivo de vistas. La función `logout_view` utiliza la función `logout()` de Django para desconectar al usuario de su cuenta y redirigirlo a la página de inicio cuando se solicita la URL de cierre de sesión.

