Mantenedor de Dimensiones Web

Gestionando la parametría de usuario de manera ágil

Acerca del Equipo

Brenda Lippo

Analista de Sistemas
(CAECE)
+10 años en BI

Javier Mastropaolo

Lic. Sistemas de Información
(UBA)
Diplomatura BI (UTN)
+10 años en BI

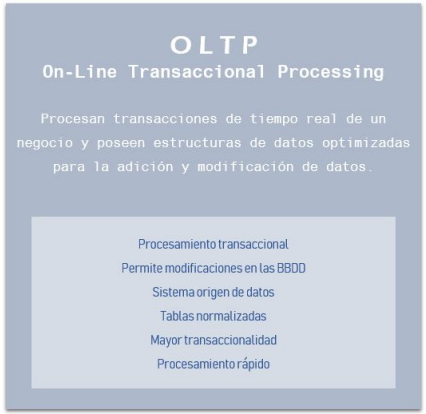
Diego Joannas

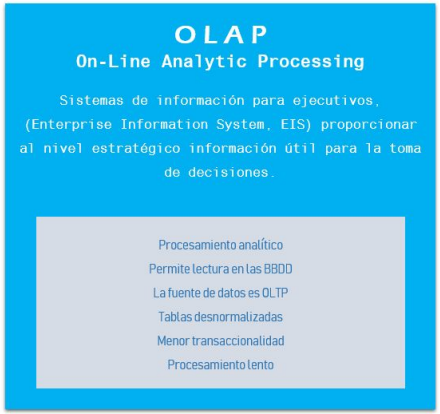
Lic. Economía (UNLP)
+10 años en BI

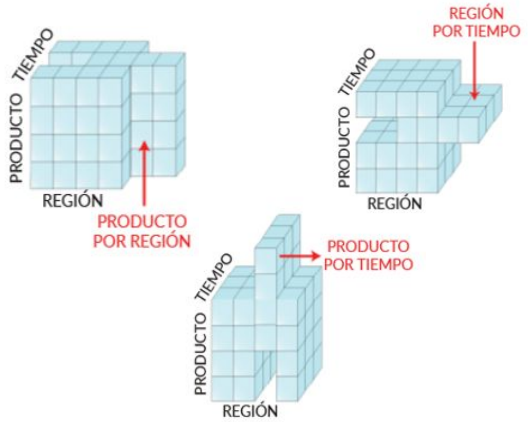
Descripción de la Necesidad

Una Entidad Financiera nos contacta para desarrollar un aplicativo web para gestionar la **parametría de usuario** en su Datawarehouse.

Existe una parametría específica, que excede al mundo OLTP, dado que se **genera y administra en el mundo OLAP**. Esta parametría es requerida por los usuarios de negocio para generar información de gestión relevante y consiste generalmente, en renombrar/reasignar/agrupar valores provenientes de los sistemas transaccionales, creando **dimensiones de análisis** conforme a definiciones de negocio.







Diseño de la Solución

Se proponer al cliente, avanzar con un **MVP** de la solución abarcando la siguientes funcionalidades:

- **Visualización de Entidades** sobre las que se requiere aplicar parametría de usuario (*)
- Desarrollar el flujo de un **ABM completo** sobre una Entidad
- Desarrollar **registro/log in/log out** sobre la Aplicación

(*) La Aplicación deberá conectarse a la Base de Publicación del Datawarehouse, y deberán mapearse aquellas tablas maestro de cada Entidad sobre la que se aplique parametría de usuario. Para el MVP se simula este punto conectándose contra una base en SQLite.

Vista Inicial

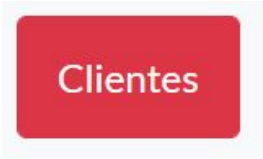
ENTIDADES

USUARIO

Funcionalidades

CRUD
(para Entidad Cliente)

Registración
Log-In
Log-Out



[C]reate

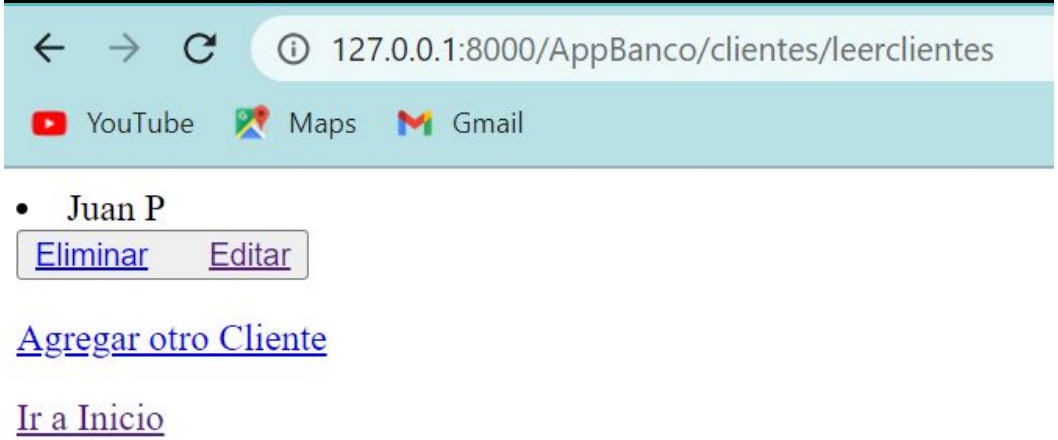
Para dar de alta un Cliente, debe ingresar el código (ej. para este MVP se puede cargar el CUIT y en un desarrollo posterior, se puede desarrollar algún secuencial interno), nombre y email. Todos los campos son de carga obligatoria, aunque sólo se valida la estructura de datos de “Código Cliente” (numérico) y “Email”.

```python
def clientes(request):

    if request.method == 'POST':
        formulario=ClienteForm(request.POST)
        if formulario.is_valid():
            informacion=formulario.cleaned_data
            cliente = Clientes(codigo_cliente=informacion['codigo_cliente'], nombre=informacion['nombre'],
            cliente.save()
            return render(request,"AppBanco/inicio.html")
    else:
        formulario=ClienteForm()
    return render(request,"AppBanco/clientes.html",{'formulario':formulario})
```

Clientes

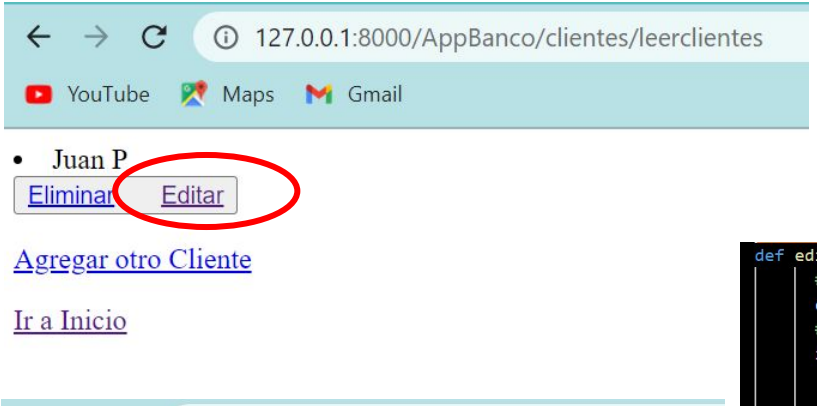
[R]ead


```python
def leerclientes(request):

    clientes = Clientes.objects.all() #trae todos los clientes
    contexto = {"clientes": clientes}
    return render(request, "AppBanco/leerclientes.html", contexto)
```



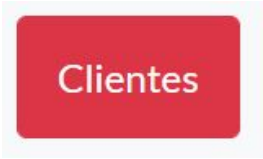
Clientes

[U]pdate

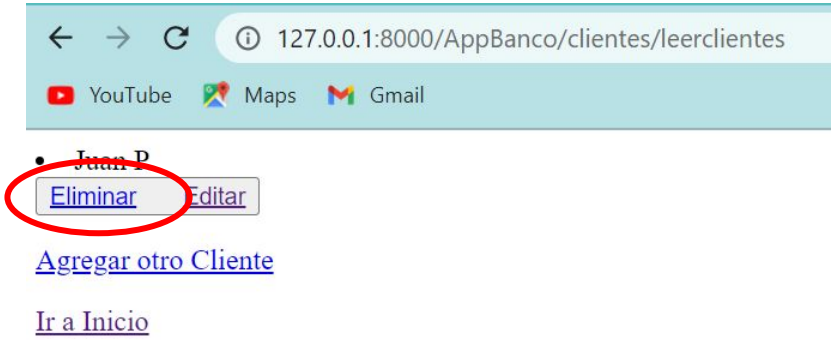


Es posible modificar cualquier campo y las validaciones que se realizan para un update, son las mismas que aplican para el create.

```python
def editarcliente(request, cliente_nombre):
    #Recibe el nombre del cliente que vamos a modificar
    clientes = Clientes.objects.get(nombre=cliente_nombre)
    #Si es metodo POST hago lo mismo que el agregar
    if request.method == 'POST':
        formulario = ClienteForm(request.POST)
        print(formulario)
        if formulario.is_valid:   #Si pasó la validación de Django
            informacion = formulario.cleaned_data
            clientes.codigo_cliente = informacion['codigo_cliente']
            clientes.nombre = informacion['nombre']
            clientes.email = informacion['email']
            clientes.save()
            clientess=Clientes.objects.all()
            contexto={'clientess':clientess}
            return render(request, "AppBanco/inicio.html",contexto)
    #En caso que no sea post
    else:
        #Creo el formulario con los datos que voy a modificar
        formulario= ClienteForm(initial={'codigo_cliente': clientes.codigo_cliente, 'nombre':clientes.
        'email':clientes.email})
    #Voy al html que me permite editar
    return render(request, "AppBanco/editarcliente.html", {"formulario":formulario, "cliente_nombre":cli
```



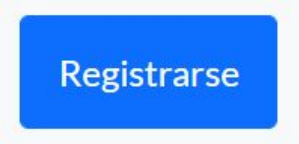
[D]elete



Al elegir delete para un cliente, retorna al homepage.

Si hubiera más de un cliente, cada cliente aparece con su opción “eliminar”/”editar”.

```python
def eliminarclientes(request, cliente_nombre):

    clientes = Clientes.objects.get(nombre=cliente_nombre)
    clientes.delete()

    #vuelvo al menú
    clientes = Clientes.objects.all() #traigo todos los clientes
    contexto = {"clientes":clientes}

    return render(request, "AppBanco/inicio.html",contexto)
```



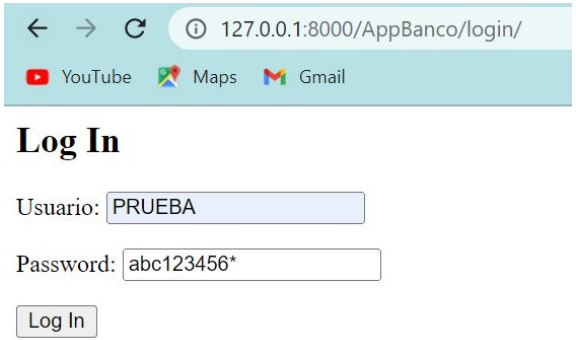
Registración de Usuario

El usuario deberá darse de alta con username, email y contraseña. El método solo valida la estructura de la contraseña. En caso de ser válida, retorna a homepage. Caso contrario, expone requisitos de la contraseña para que el usuario corrija contraseña.

- This password is too short. It must contain at least 8 characters.
- This password is too common.
- This password is entirely numeric.

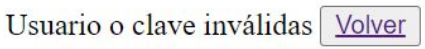
```
def register(request):
    if request.method == 'POST':
        formulario=RegistrationForm(request.POST)
        if formulario.is_valid():
            username=formulario.cleaned_data.get('username')
            formulario.save()
            return render(request,"AppBanco/inicio.html", {'mensaje':"Usuario registrado con éxito"})

    else:
        formulario=RegistrationForm()


    return render(request,"AppBanco/registro.html", {'formulario':formulario})
```

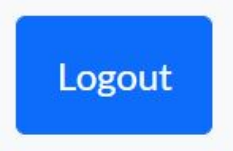
Logueo de Usuario



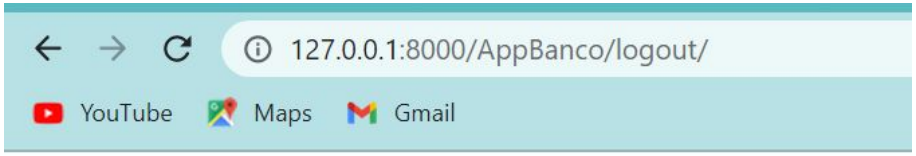
El usuario deberá ingresar con username y contraseña. En caso de ser correcta la combinación de valores, retonar al homepage y se visualiza “Bienvenido + Username”. Caso contrario, se expone mensaje de error.

```python
def login_view(request):
    if request.method == 'POST':
        formulario=LoginForm(request.POST)
        if formulario.is_valid():
            username=formulario.cleaned_data.get('username')
            password=formulario.cleaned_data.get('password')
            # Authenticate user
            user = authenticate(username=username, password=password)
            if user is not None:
                login(request, user)
                # A backend authenticated the credentials
                print("Login successfuly")
                # return render(request,"AppBanco/inicio.html")
                return inicio(request)
            else:
                # No backend authenticated the credentials
                # Return an 'invalid login' error message.
                return render(request,"AppBanco/login.html", {'mensaje':"Usuario o clave inválidas"})
        else:
            return render(request,"AppBanco/inicio.html", {'mensaje':"Formulario Inválido"})

    formulario=LoginForm()
    return render(request, "AppBanco/login.html", {'formulario':formulario})
```





Deslogueo de Usuario




```python
def logout(request):
    logout(request)
    return render(request,"AppBanco/logout.html")
```

AppBanco > Templates > AppBanco > logout.html > html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Logout</title>
</head>
<body>
    <h1>Te has deslogueado, espera...</h1>

    <script>
        // after 3 seconds redirects to index.html
        setTimeout(function(){
            window.location.href = "{% url 'inicio' %}";
        }, 3000);

    </script>
</body>
</html>
```

Al desloguearse, se definió que a los 3 segundos, el método retorne al homepage.

Gracias