# **Evaluacion N° 4**

# Programación orientada a objetos

NOMBRE: Nicolás Anrique Benitez

Javier Arriagada Contreras

Maximiliano Díaz Marchant

Mario Alejandro Mora Vega

Gabriel Riquelme Mery

CARRERA: Analista Programador

ASIGNATURA: Programación Orientada a Objetos

PROFESOR:Roberto Fica

FECHA: 16/12/2023

Contenido

[**Evaluacion N° 4** 1](#_Toc153651048)

[Programación orientada a objetos 1](#_Toc153651049)

[1 Introducción 3](#_Toc153651050)

[2 Desarrollo 3](#_Toc153651051)

[2.1 Modifica usuario 3](#_Toc153651052)

[2.2 Gestionar ingreso de productos/servicios 3](#_Toc153651053)

[2.3 Transacción de negocio completa 3](#_Toc153651054)

[3 Conclusión 4](#_Toc153651055)

# Introducción

En este documento, presentamos las actualizaciones realizadas en nuestro programa de transporte de carga, desarrollado con el framework Django. Hemos realizado una serie de mejoras y adiciones para mejorar la funcionalidad y la eficiencia del programa. En primer lugar, implementaremos una funcionalidad de modificación de usuario, para así extender lo hecho en la tercera evaluación. Siguiendo esta línea, crearemos la funcionalidad de ingreso de carga, que también funciona como una transacción comercial.

Todas estas funciones se han probado exhaustivamente y se han verificado con capturas de pantalla del HTML visto por el navegador para garantizar su correcto funcionamiento. Continuaremos trabajando para mejorar y expandir las capacidades de nuestro programa de transporte de carga. Gracias por su apoyo continuo.

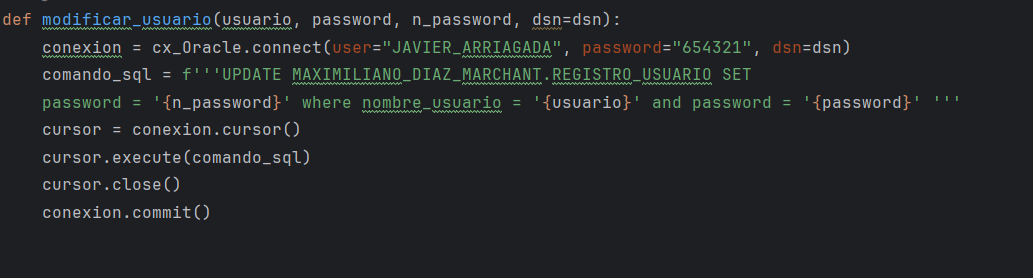
# Desarrollo

## Modifica usuario

### Función de conexión a base de datos

De acuerdo a la funcionalidad establecida en la anterior evaluación, hemos creado la función “modificar\_usuario” para que esta se conecte a nuestra base de datos de Oracle y actualice la tabla REGISTRO\_USUARIO con los datos ingresados. La función pide como argumentos las cadenas de “usuario“, “password” y “nueva password”. Luego, esta se conecta a la base de datos, instanciando los objetos de conexión (connect) , dados por el módulo Oracle\_cx. Para construir este objeto hemos ingresado la contraseña y el usuario reales de la base de datos como cadena estáticas. Una vez instanciada la conexión, construimos un objeto cursor (también dado por Oracle\_cx) y con este ejecutamos los comandos sql “UPDATE “ que actualiza la tabla con los datos ingresados.

Captura:



**Código:**

def modificar\_usuario(usuario, password, n\_password, dsn=dsn):  
 conexion = cx\_Oracle.connect(user="JAVIER\_ARRIAGADA", password="654321", dsn=dsn)  
 comando\_sql = f'''UPDATE MAXIMILIANO\_DIAZ\_MARCHANT.REGISTRO\_USUARIO SET  
 password = '{n\_password}' where nombre\_usuario = '{usuario}' and password = '{password}' '''  
 cursor = conexion.cursor()  
 cursor.execute(comando\_sql)  
 cursor.close()  
 conexion.commit()

### Función view

Hemos creado una nueva función en el archivo view, llamada “mod\_usuario”, esta toma un objeto tipo request, que responde a la url “mod-usuario”. Se genera un diccionario a partir del POST del html correspondiente. Este es convertido a un diccionario y mediante un get conseguimos los valores de usuario, password y nueva password. Estos valores son posteriormente bajados a la función “modificar\_usuario”, que los usa para actualizar nuestra base de datos.

**Captura**



**Código:**

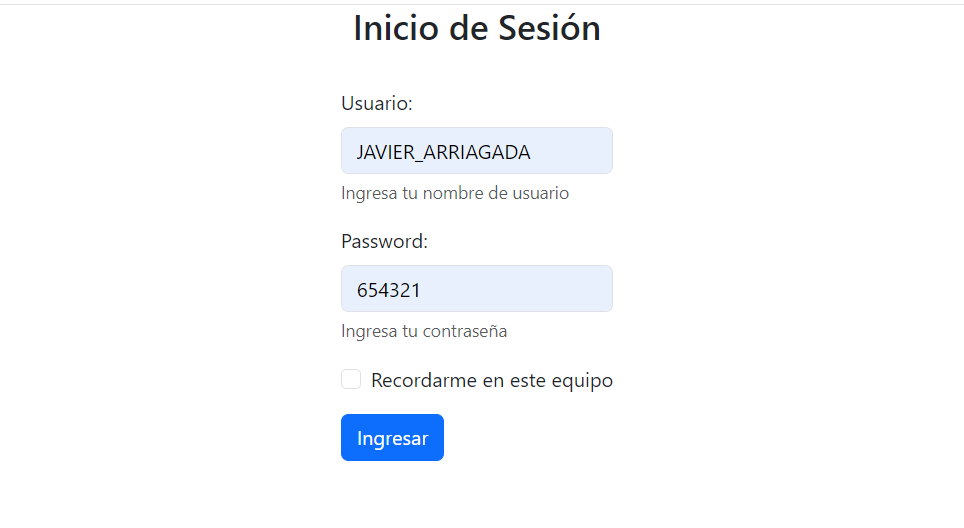
def mod\_usuario(request):  
 if request.POST:  
 data = request.POST.dict()  
 usuario = data.get("usuario")  
 password = data.get("password")  
 nueva\_password = data.get("nueva\_password")  
 modificar\_usuario(usuario, password,nueva\_password)  
 print(data)  
 return render(request, "crud/mod-usuario.html")

**Path url:**

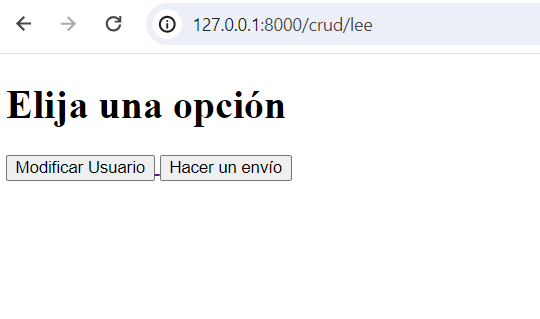


### Resultado de implementación en HTML

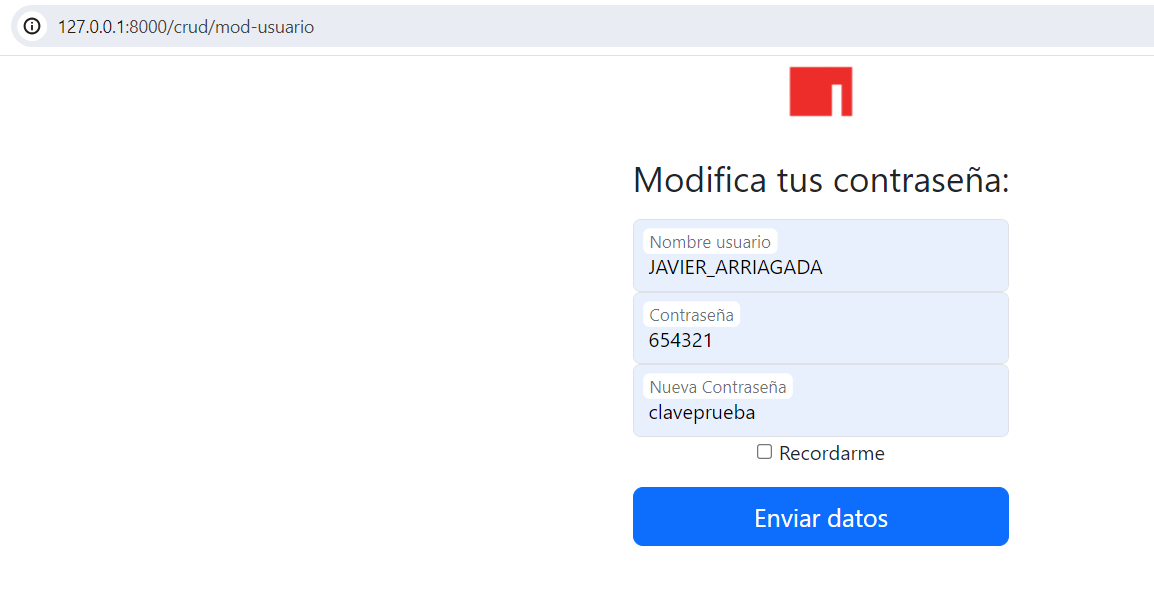
En primer lugar, iniciamos sesión usando la funcionalidad que implementamos en la tercera evaluación.



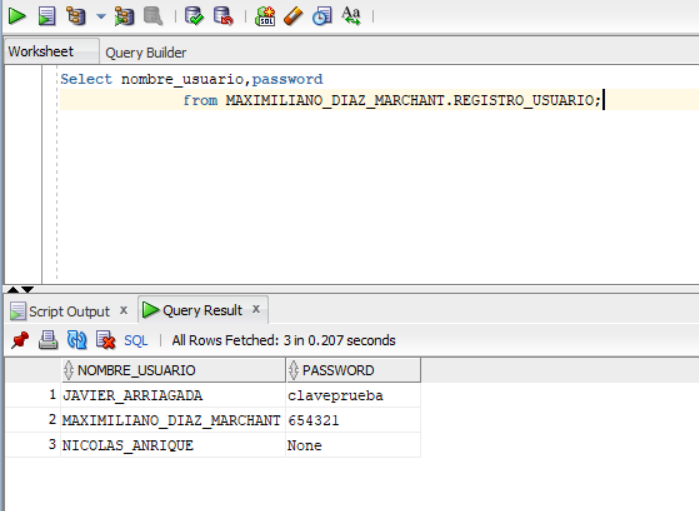
Si el usuario y contraseña son reconocidos en la base de datos, entramos a un menú de opciones:



Al hacer clic en modificar usuario, accedemos al formulario para actualizar la base de datos:



Una vez ingresados los datos, comprobamos si la tabla ha sido actualizada.



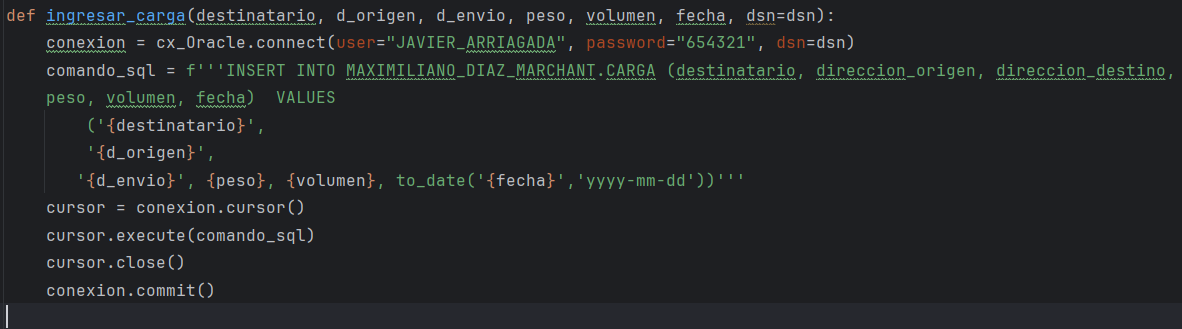
## Gestionar ingreso de productos y Transacción comercial

**Nota: Hemos centrado nuestros esfuerzos en el ingreso de una carga, con todos sus datos correspondientes. Como somos una empresa de transporte, el ingreso de nuestro servicio de transporte y el proceso de registro de una transacción constituyen la misma funcionalidad.**

### Función de conexión a base de datos

De acuerdo a la funcionalidad establecida en la anterior evaluación, hemos creado la función “ingresar\_carga” para que esta se conecte a nuestra base de datos de Oracle y registre el ingreso de una carga. La función pide como argumentos las cadenas de “destinatario”, “d\_origen”, “d\_destino”, y los enteros “peso”,”volumen. Luego, esta se conecta a la base de datos, instanciando los objetos de conexión (connect), dados por el módulo Oracle\_cx. Para construir este objeto hemos ingresado la contraseña y el usuario reales de la base de datos como cadena estáticas. Una vez instanciada la conexión, construimos un objeto cursor (también dado por Oracle\_cx) y con este ejecutamos los comandos sql “INSERT INTO“ que ingresa a la tabla CARGA los datos ingresados.

**Captura**



**Codigo:**

def ingresar\_carga(destinatario, d\_origen, d\_envio, peso, volumen, fecha, dsn=dsn):  
 conexion = cx\_Oracle.connect(user="JAVIER\_ARRIAGADA", password="654321", dsn=dsn)  
 comando\_sql = f'''INSERT INTO MAXIMILIANO\_DIAZ\_MARCHANT.CARGA (destinatario, direccion\_origen, direccion\_destino,  
 peso, volumen, fecha) VALUES   
 ('{destinatario}',  
 '{d\_origen}',  
 '{d\_envio}', {peso}, {volumen}, to\_date('{fecha}','yyyy-mm-dd'))'''  
 cursor = conexion.cursor()  
 cursor.execute(comando\_sql)  
 cursor.close()  
 conexion.commit()

### Función view

Hemos creado una nueva función en el archivo view, llamada “formulario\_carga”, esta toma un objeto tipo request, que responde a la url “hacer-envio”. Se genera un diccionario a partir del POST del html correspondiente. Este es convertido a un diccionario y mediante un get conseguimos los valores de destinatario, “d\_origen”,” d\_envio”, peso y volumen. También se crea la variable “hoy” para instanciar el objeto date y generar la fecha actual de la transacción. Una vez capturados estos valores, los bajamos a la función “ingresar\_carga”, que se usan para actualizar nuestra base de datos.

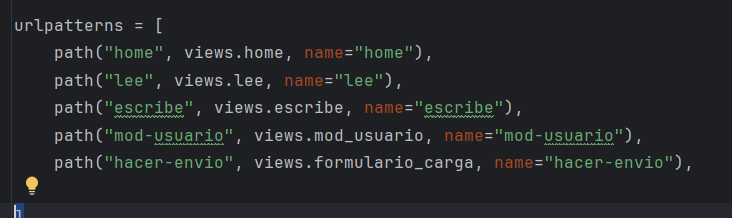
**Captura:**



**Código:**

def formulario\_carga(request):  
 if request.POST:  
 data = request.POST.dict()  
 destinatario = data.get("destinatario")  
 d\_origen = data.get("d\_origen")  
 d\_envio = data.get("d\_envio")  
 peso = data.get("peso")  
 volumen = data.get("volumen")  
 hoy = datetime.date.today()  
 ingresar\_carga(destinatario, d\_origen, d\_envio, peso, volumen, hoy)  
 print(data)  
 return render(request, "crud/hacer-envio.html")

**URL patterns:**

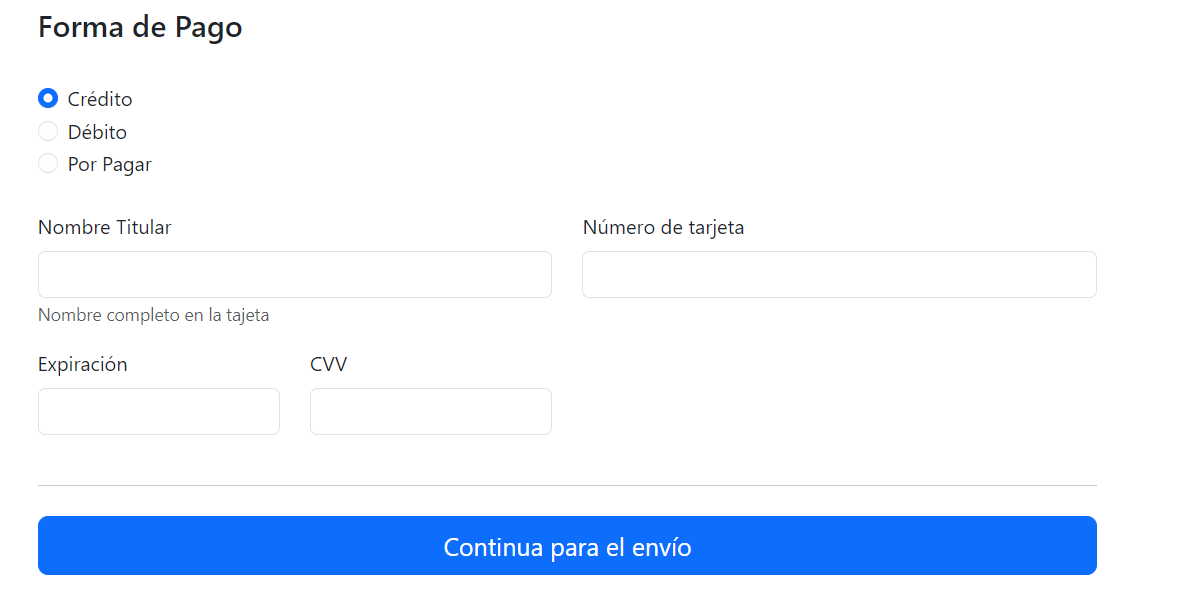


### Implementación en html

Hemos modificado la estética de los html con un poco de css.

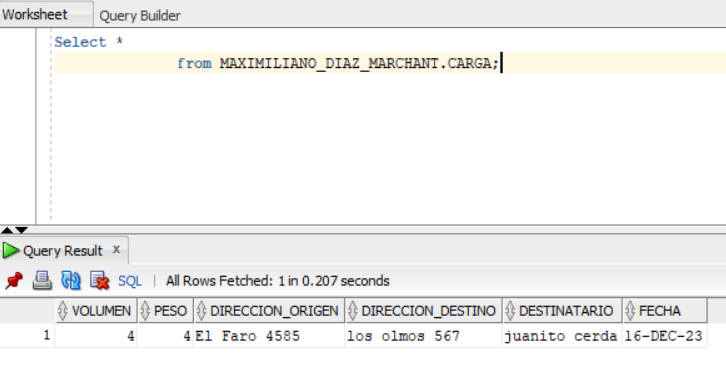
**Ingreso de datos:**





NOTA: hemos escrito este html de simulación de pago, pero no hemos configurado esta funcionalidad en la base de datos todavía.

**Registro en base de datos:**



# Conclusión

Finalmente, este documento explica la implementación y las actualizaciones del programa de transporte de carga, desarrollado con Django. Una función para editar la información del usuario, además de una función de transacción e ingreso de carga.. Todas estas actualizaciones y funciones han sido probadas y verificadas para asegurarse de que funcionen correctamente. El equipo seguirá trabajando en Django para mejorar las funciones del programa de transporte de carga.