

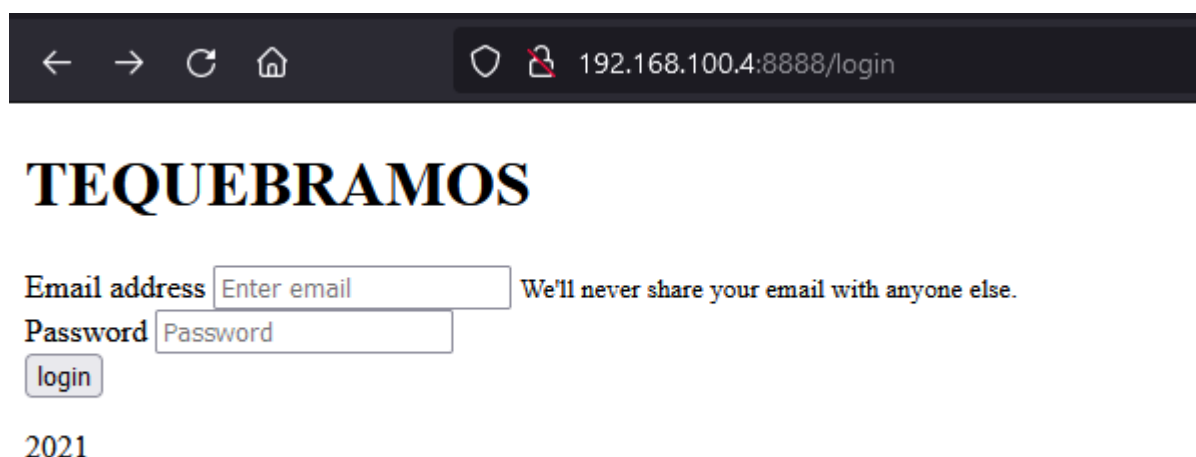
**Implementar un módulo que permita filtrar intentos de ataques de SQL Injection evitando consultas vulnerables y que incorpora funciones para escapar caracteres anómalos en las cadenas SQL, debe evidenciar la vulnerabilidad de inyección y la contramedida.**

>>

Una entidad bancaria acaba de abrir su página web, para ingresar en dicha página un usuario debe de poner su correo y contraseña.

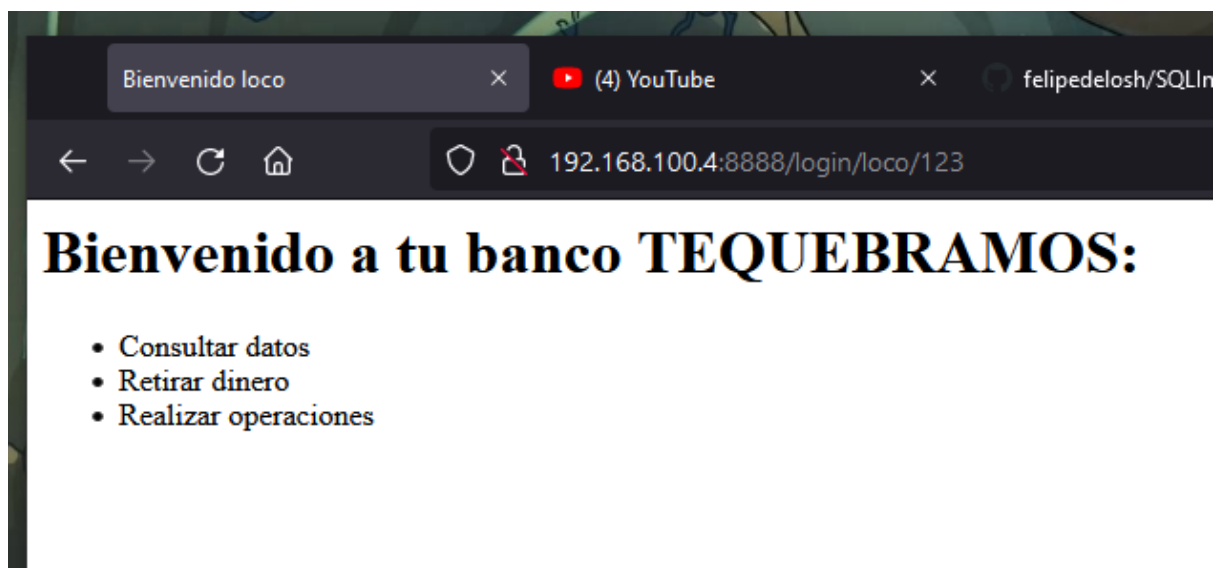
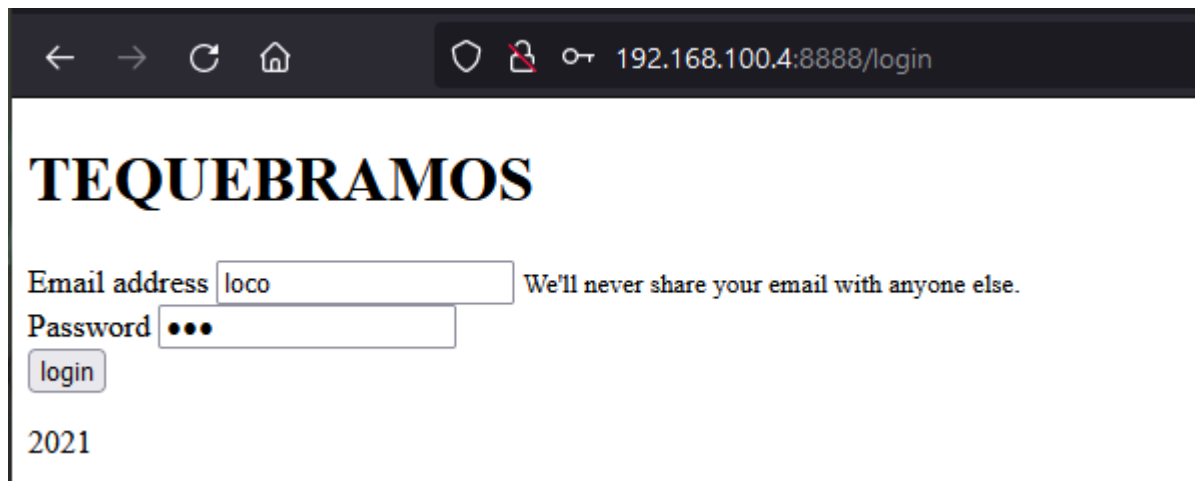
El modelo para ingresar es el siguiente:

1 -> Se pone a correr el servidor python Flask  
IPSERVER+":"+8888+"/login"



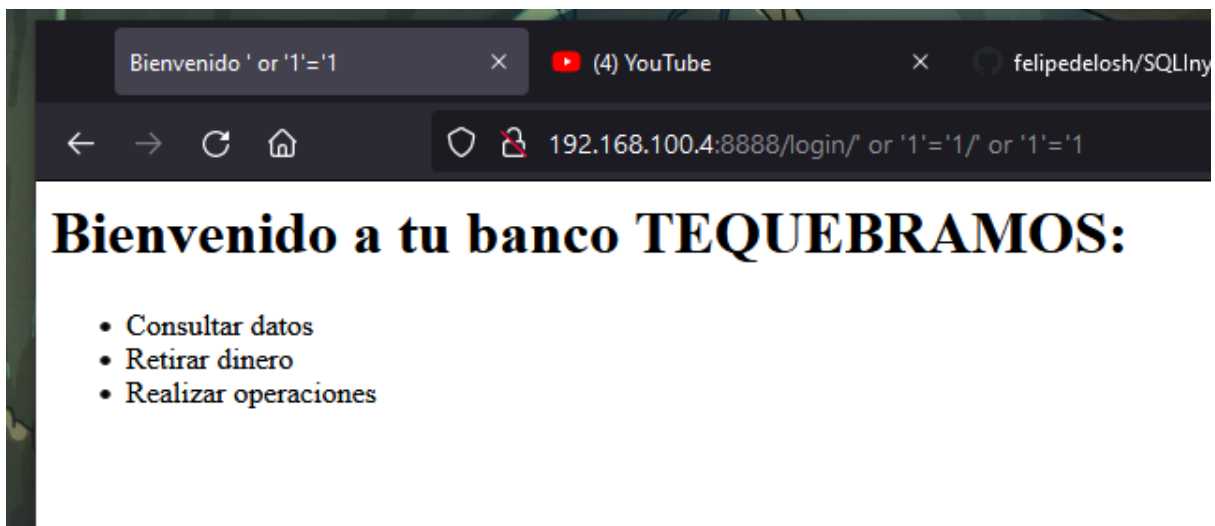
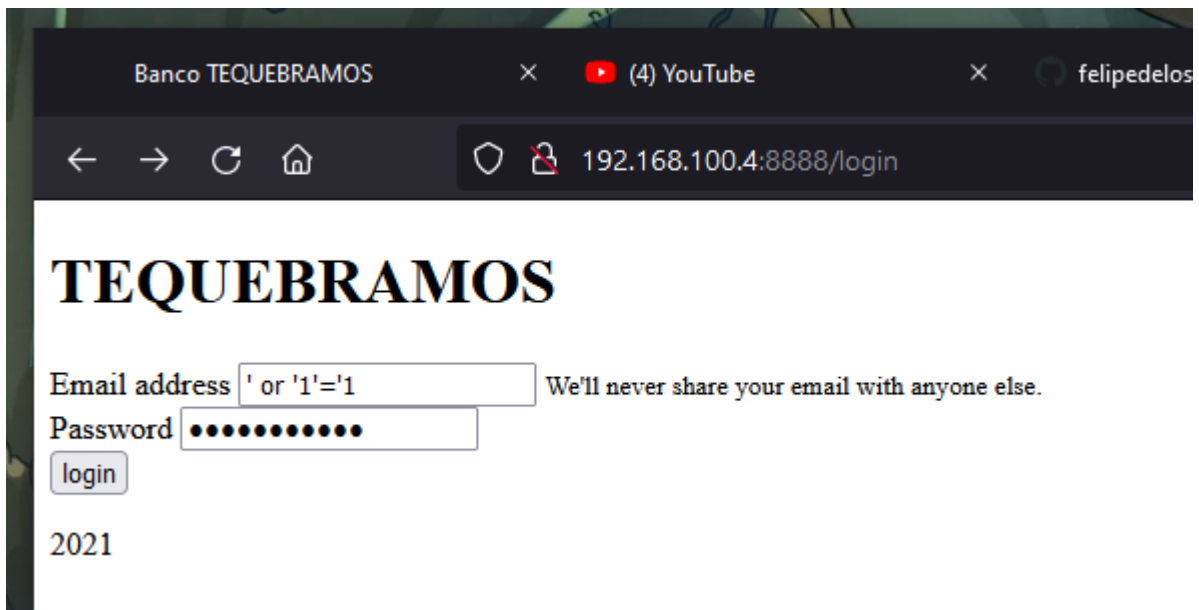
The screenshot shows a web browser window with a dark address bar displaying the URL '192.168.100.4:8888/login'. The page content features the title 'TEQUEBRAMOS' in a large, bold, serif font. Below the title, there is a login form with two input fields: 'Email address' with a placeholder 'Enter email' and 'Password' with a placeholder 'Password'. To the right of the email field, a text string reads 'We'll never share your email with anyone else.' Below the password field is a 'login' button. At the bottom left of the form area, the year '2021' is displayed.

2 -> Se procede a verificar el funcionamiento: por defecto la base de datos contiene el user loco con contraseña 123



3 -> Se procede a verificar una inyección SQL para ello introducimos usuario y contraseña :

**' or '1'='1**



Esto sucede por que el login tiene la siguiente consulta SQL

```
def loginUser(self, strusername, strpassword):
    conn = sqlite3.connect(self.dataBasename, check_same_thread=False)
    cur = conn.cursor()
    sql = "select * from user where username='"+strusername+"' and password='"+strpassword+"';"
    cur.execute(sql)
    return len(cur.fetchall()) > 0
```

y dicha consulta produce:

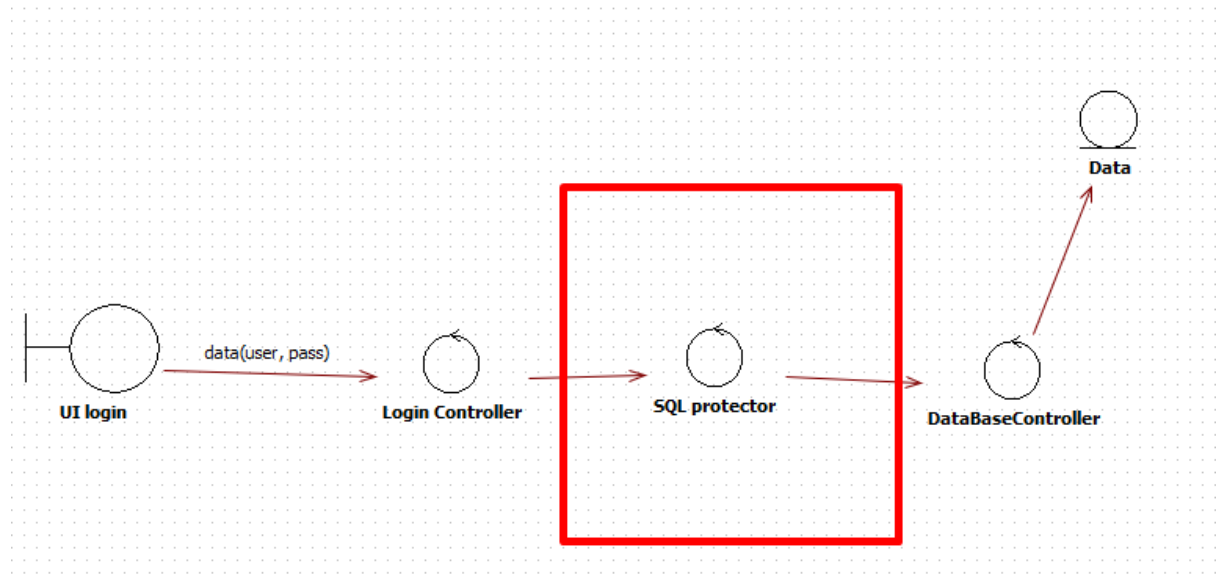
```
loginUser("' or '1'='1", "' or '1'='1'")
```

Lo que se traduce en:

```
select * from user where username=" or '1'='1' and password=" or '1'='1';
```

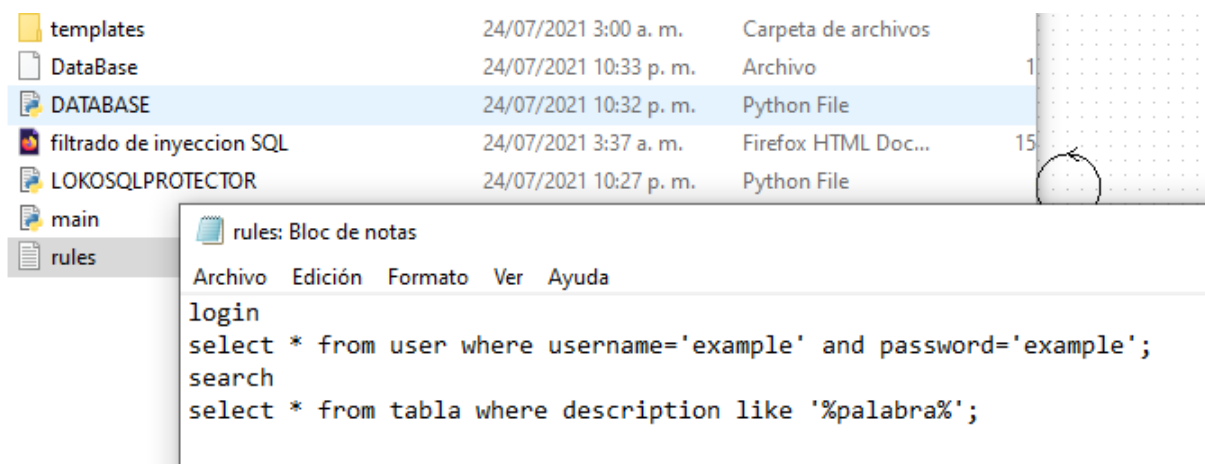
# Solución:

Se crea un agente que proteja las consultas, el módulo va a estar basado en reglas y tiene la siguiente estructura



Se trata de un controlador que verifica las consultas antes de ser enviadas al motor de base de datos. y funciona de la siguiente manera:

Se debe tener un archivo con una consulta de ejemplo:



Dicha consulta se usa para pasar 3 simples pruebas:

**1 -> las consultas de ejemplo y las consultas que se envían desde HTML tienen que tener la misma cantidad de palabras.**

Comparemos la consulta falsa con la consulta verdadera:

```
select * from user where username='loco' and password='123';  
select * from user where username="" or '1'='1' and password="" or '1'='1';
```

La consulta falsa no va a pasar el test porque tienen diferente longitud de palabras.

**2 -> la consulta ejemplo y la consulta enviada desde el HTML deben de tener las mismas palabras reservadas.**

```
select * from user where username='loco' and password='123';  
select * from user where username="" or '1'='1' and password="" or '1'='1';
```

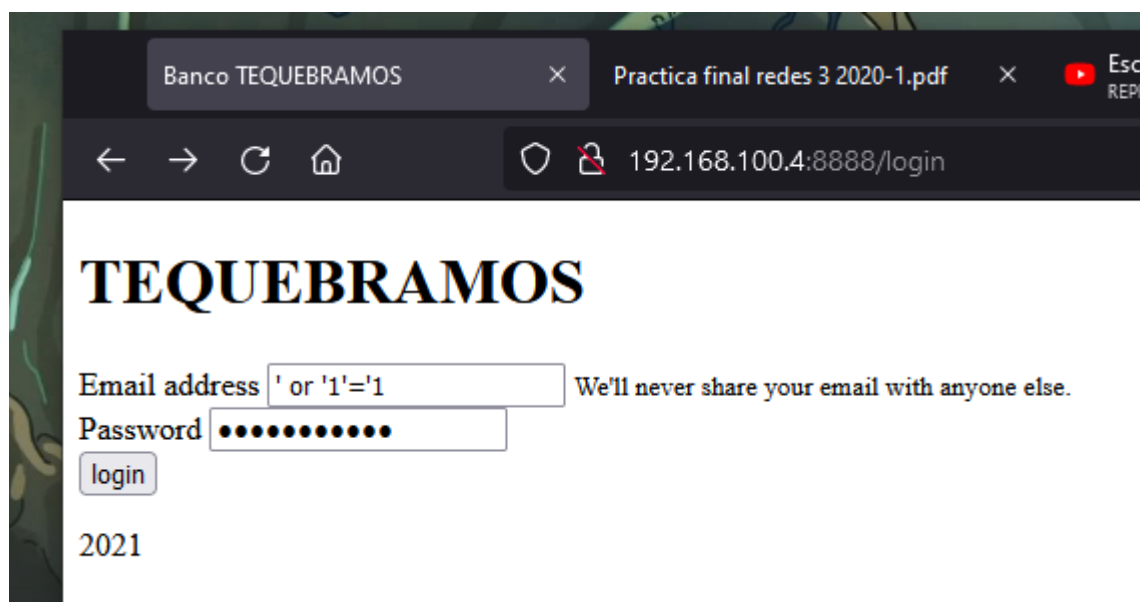
La consulta falsa no va a pasar el test porque tiene 2 “or”, los cuales no existen en la consulta de ejemplo.

**3 -> La consulta ejemplo y la consulta enviada desde el HTML deben de tener la misma cantidad de condicionales.**

```
select * from user where username='loco' and password='123';  
select * from user where username="" or '1'='1' and password="" or '1'='1'
```

la consulta falsa no va a pasar el test por que tiene 2 igualdades más.

## ***Funcionamiento:***





Código:

```
def loginUser(self, strusername, strpassword):
    conn = sqlite3.connect(self.dataBasename, check_same_thread=False)
    cur = conn.cursor()
    sql = "select * from user where username='"+strusername+"' and password='"+strpassword+"';"
    if self.sqlProtector.verifySQL("login", sql):
        cur.execute(sql)
        return len(cur.fetchall()) > 0
    else:
        return False
```