Descripción de la aplicación.

Esta es una aplicación de consola que permite la consulta de información sobre usuarios la cual está almacenada en una base de datos que, mediante el web-scraping, se actualiza. Esta aplicación permite el ingreso, validación de credenciales de dos categorías de usuario. Usuarios “admin”, tienen la posibilidad de crear y eliminar usuarios llamados “requesters”. La creación de ambos tipos de usuarios requiere de la asociación de una contraseña. Los usuarios tipo “requesters” son capaces de hacer queries a una base datos que tiene información sensible de usuarios.

Los usuarios “requesters” son capaces de correr queries, bajo ciertas restricciones de palabras para preservar la integridad de la información. Estas restricciones se encuentran en dos tablas. llamadas insiders y outsiders esto con el fin de modificar el nivel de acceso que tienen los requesters siendo parte o no de la compañía pudiendo añadir más restricciones a estos. Las restricciones pueden ser modificadas en dado caso de que se vea necesario por motivos de seguridad o funcionamiento. Para crear un usuario tipo “requester” se pide el nombre completo de una persona natural.

La base de datos está compuesta de cinco tablas “users”, “admins”, “requesters”, “insiders” y “outsiders” hay pocas maneras de modificar cada una de las tablas. Comenzando por la tabla “users”, solo disponible para personas con acceso a la máquina que da host al servidor de la base de datos. La tabla “users” es modificada mediante la función “update()” que corre automáticamente cuando un admin ingresa a la aplicación. Esta función borra todos los datos y vuelve a cargar todos los datos desde la API, lo que esto permite es no mantener información que haya sido eliminada, una de las razones podría ser la cláusula de las políticas de privacidad que permiten a los usuarios la eliminación de sus datos personales. La tabla “requesters” es modificada desde el código si algún usuario “admin” agrega o elimina un usuario “requester”, la tabla “insiders” contiene las palabras restringidas para los usuarios con menor nivel de restricción y la tabla “outsiders” contiene las palabras restringidas para los usuarios con la mayor nivel de restricción, la modificación de estas tablas está restringida para admins. Como valores iniciales ambas tablas tienen como palabra restringida el nombre de las tablas insider y outsider, por lo que como “requester” no se pueden correr queries que añadan, eliminen o modifiquen datos de estas tablas.

Esta aplicación requiere del uso de software como XAMPP el cual permite dar host a una base de datos relacional que usa el lenguaje SQL, también se apoya en otra aplicación llamada SQLBackupAndFTP, que permite realizar copias de seguridad como mecanismo de protección en caso de la destrucción, secuestro, corrupción de los datos. La periodicidad de las copias de seguridad pueden ser modificadas desde la máquina que está dando host a la base de datos. Estas copias de seguridad también tienen un tiempo de vida máximo en el sistema.

Supuestos

Entre las condiciones necesarias o que se asumen disponibles para el usuario encontramos las siguientes.

Supuestos de recursos.

- Un computador que sirva como servidor para hacer de host a la base de datos, debe tener suficiente capacidad de almacenamiento y procesamiento.
- Una persona capacitada en python y SQL que pueda servir como administrador de la aplicación
- Un elemento de acumulación de información en el cual se pueda almacenar una copia de seguridad en formato físico.

supuestos de presupuesto.

- recursos monetarios suficientes para sostener la escalabilidad del proyecto cuando esto sea necesario debido al gran volumen de datos.
- recursos monetarios para la contratación de personal capacitado

Supuesto tecnológico.

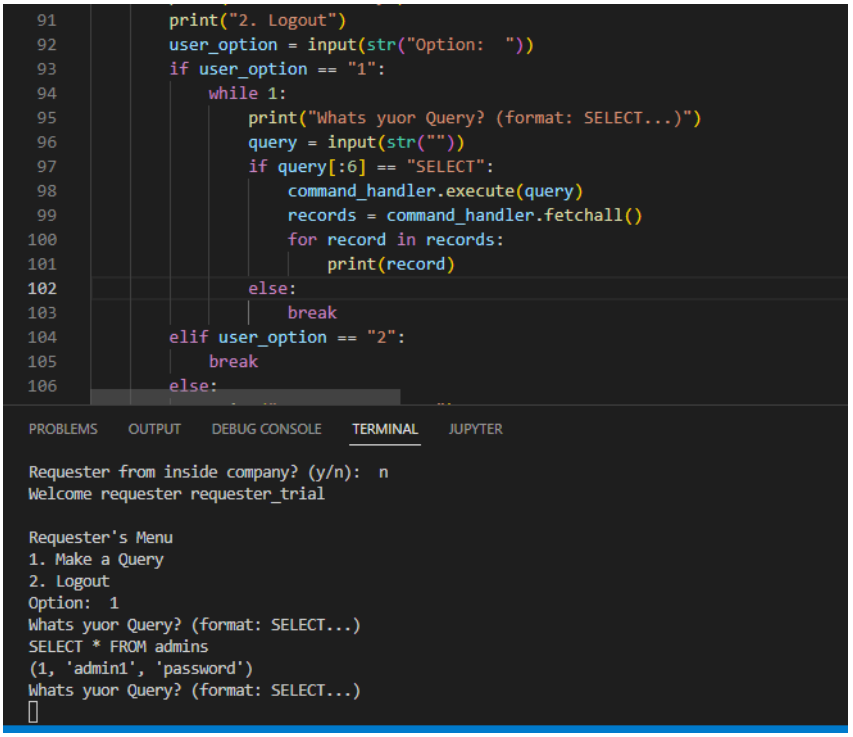
- el mantenimiento y actualización de las aplicaciones de terceros que se usan para el buen funcionamiento de nuestra aplicación.

Problemas y soluciones.

Entre los problemas y soluciones encontramos los siguientes.

Durante el desarrollo de la aplicación se lograron correr los queries que los usuarios de tipo “request” quisieran, sin embargo esto nos llevó a poder acceder a cualquier base de datos, dando información sensible o que podría comprometer el servicio de la aplicación.

En la imágen siguiente se observa cómo fue posible acceder a la base de datos de los administradores, como a sus contraseñas.



la solución para esto se encontró generando una lista de palabras que están prohibidas al momento de correr un query, el programa se encarga de buscar estas palabras y en dado caso de encontrar alguna, termina la operación y pide un nuevo query que sea válido

```
if user_option == "1":
    while 1:
        print("Whats your Query? ")
        query = input(str(""))
        query = query.lower()
        no_allowed = ["admins","requesters","delete"]
        count = 0 #esto podria hacerse como una funcio
        for item in no_allowed:
            if item not in query:
                count = count + 1
        if count == len(no_allowed):
            command_handler.execute(query)
            records = command_handler.fetchall()
            for record in records:
                print("")
                print(record)
            break
        else:
```

Aquí encontramos otro problema, al quedar las restricciones en el código base, esto puede generar fallos de seguridad, la solución que encontramos fue pasar las restricciones a una nueva tabla en la base de datos que tendrá el siguiente formato

#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
1	id	int(11)			No	*Ninguna*		AUTO_INCREMENT	Cambiar Eliminar Más
2	restriction	varchar(40)	utf8mb4_general_ci		No	*Ninguna*			Cambiar Eliminar Más

para esto implementamos que los usuarios “admin” puedan agregar palabras restringidas la la tabla, así como eliminarlas, también existe una diferencia para usuarios tipo “requesters”, esto debido a que queremos generar un diferente nivel de privilegios según si los usuarios están vinculados o no a la compañía

```
def add_restriction():
    ''''
    add restrictions from different tables
    as admin would choose '''
    print("")
    print("Register new retriction")
    insider = input(str("is the restriction for insiders? (y/n):   "))
    print("")
    restriction = input(str("What is the new restricted word:  ")).lower()

    query_vals = (restriction,)
    if insider == "y":
        command_handler.execute("SELECT * FROM insiders WHERE restriction = %s",query_vals )
        if command_handler.rowcount > 0:
            print("WORD ALREADY RESTRICTED")
            return
        command_handler.execute("INSERT INTO insiders (restriction) Values (%s)",query_vals)
        print(restriction + " WORD HAS BEEN ADDED AS A NEW RESTRICTION")
    elif insider == "n":
        command_handler.execute("SELECT * FROM outsiders WHERE restriction = %s",query_vals )
        if command_handler.rowcount > 0:
            print("WORD ALREADY RESTRICTED")
            return
        command_handler.execute("INSERT INTO outsiders (restriction) Values (%s)",query_vals)
        print(restriction + " WORD HAS BEEN ADDED AS A NEW RESTRICTION")
    else:
        ()
    db.commit()
```

También se ha realizado este procedimiento de diferenciación al eliminar restricciones para mantener las dos tablas de forma independiente.

```
def elim_restriction():
    ''''
    eliminate restrictions from different tables
    as admin would choose
    '''
    print("")
    print("Delete existence retriction")
    insider = input(str("is the restriction for insiders? (y/n):   "))
    print("")
    restriction = input(str("What is the restricted word:  ")).lower()
    query_vals = (restriction,)
    if insider == "y":
        command_handler.execute("SELECT * FROM insiders WHERE restriction = %s",query_vals )
        if command_handler.rowcount == 0:
            print(restriction+" WORD IS NOT RESTRICTED YET")
            return
        command_handler.execute("DELETE FROM insiders WHERE restriction = %s",query_vals)
        print(restriction+" HAS BEEN REMOVED FROM RESTRICTED WORDS")
    elif insider == "n":
        command_handler.execute("SELECT * FROM outsiders WHERE restriction = %s",query_vals )
        if command_handler.rowcount == 0:
            print(restriction+" WORD IS NOT RESTRICTED YET")
            return
        command_handler.execute("DELETE FROM outsiders WHERE restriction = %s",query_vals)
        print(restriction+" HAS BEEN REMOVED FROM RESTRICTED WORDS")
    else:
        print("INPUT NOT CLEAR")
    db.commit()
```

y también se modificó el código para que los usuarios “requesters” al hacer un query, tengan las limitaciones basados en la tabla que se ha creado, con su respectiva diferenciación.

```
def requester_query(inside):
    '''
    this function checks if the requester is from inside or outside company
    different restrictions are checked
    '''
    print("")
    print("Whats your Query? ")
    query = input(str(""))
    query = query.lower()
    if inside == "y":
        command_handler.execute("SELECT restriction FROM insiders")
        not_allowed = command_handler.fetchall()
    elif inside == "n":
        command_handler.execute("SELECT restriction FROM outsiders")
        not_allowed = command_handler.fetchall()
    else:
        print("INPUT NOT CLEAR")
        return (0,)
    count = 0

    for item in not_allowed:
        if item[0] in query:
            count = count + 1
    if count == 0:
        command_handler.execute(query)
        records = command_handler.fetchall()
```

Otro problema que se encontró fue con la protección de los datos para los usuarios con privilegios como los administradores. Inicialmente los datos de los administradores se encontraban en el código, esto causaba una vulnerabilidad.
El anterior problema se soluciono creando una nueva base de datos en la cual se guarda la información de los administradores, a esta base de datos solo se puede acceder si se tiene acceso a la máquina que se usa como servidor para la base de datos.

	#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra
☐	1	id 🔑	int(11)			No	*Ninguna*		AUTO_INCREMENT
☐	2	admin_name	varchar(35)	utf8mb4_general_ci		No	*Ninguna*		
☐	3	password	varchar(35)	utf8mb4_general_ci		No	*Ninguna*		

Un problema adicional que se encontró fue eliminar de la base de datos la información que los usuarios habían pedido eliminar. Para lo anterior se decidió que cada vez que se actualizará la base de datos se eliminaría toda la información que ya se tenía. Esta solución se implementó solamente cuando un administrador inicia sesión. lo que permite que los usuarios “requesters” no puedan acceder a esta funcionalidad. Esto ayuda también a que el volumen de solicitudes y el tráfico de datos innecesarios caiga dramáticamente.

```python
def update():
    ''' # tambien es seguro por que elimina lo que ya no se tenga en la base de datos.
    this function updates the db by deleting all information
    and replacing all the information the information that is
    stored in the api.'''
    command_handler.execute("DELETE FROM users")
    db.commit()
    response_api = requests.get("https://62433a7fd126926d0c5d296b.mockapi.io/api/v1/usuarios")
    #print(response_api.status_code)
```

```python
def admin_session():
    update()
    '''
    In this function we have the logic for when the adming
    access, here he can create and delete Requesters.
    '''
    print("")
    print("Login success welcomme admin: ")
```

Uno de los últimos problemas que se encontró es que era posible duplicar los usuarios de tipo “requesters”, al no tener una validación para usuarios existentes.

```python
if user_option == "1":
    print("")
    print("Register new Requester")
    requestername = input(str("Requester name:  "))
    password = input(str("Requester password:  "))
    inside = input(str("Requester from inside company? (y/n):  "))
    responsable = input(str("What is the name of the person in charged?:  "))
    query_vals = (requestername,password,inside,responsable)
    command_handler.execute("INSERT INTO Requesters (requestername, password, inside,responsable)
    db.commit()
    print(requestername + " has been registered as a requester")
```

Esto se soluciono con una búsqueda en la base de datos, que validaba, según el nombre del usuario tipo “requester” que no hubiese ninguno creado.

```python
if user_option == "1":
    print("")
    print("Register new Requester")
    requestername = input(str("Requester name:  "))
    password = input(str("Requester password:  "))
    inside = input(str("Requester from inside company? (y/n):  "))
    responsable = input(str("What is the name of the person in charged?:  "))
    query_vals = (requestername,password,inside,responsable)
    req_vals = (requestername,)
    command_handler.execute("SELECT * FROM requesters WHERE requestername = %s",req_vals)
    if command_handler.rowcount > 0:
        print("the requester "+requestername+" in already registered")
        continue
    command_handler.execute("INSERT INTO Requesters (requestername, password, inside,responsable)
    db.commit()
    print(requestername + " has been registered as a requester")
```