**Flask-Login**

Recurso fuente: https://flask-login.readthedocs.io/en/latest/

**Introducción**

**En el presente documento se relacionara como implementar la extensión de flask login para un proyecto hecho con Flask. Se mostraran la secuencia de pasos y requisitos que se deben seguir y tener respectivamente. Los ejemplos serán basados en el proyecto didáctico Minitienda.**

**Flask login:**

El objetivo de la extensión es la de facilitar la implementación de un sistema de autenticación y control de acceso.

**IMPLEMENTACION EN LA APP**

1. Importar modulos :

Para hacer posible la implementación se debe importar desde la app principal (donde se tiene los controladores y routers)

from flask\_login import LoginManager,login\_user,logout\_user,current\_user, login\_required

Dónde:

- **LoginManager:** es la clase API la cual básicamente permite integrar la aplicación con el

El sistema de logueo (login, logout, current\_user etc). Y debe ser de instanciada como

Sigue preferiblemente en la app principal.

login\_manager=LoginManager(app)

\*\***comentario:** Flask login usa sessions como variable para authenticacion. De manera que el proyecto debe tener un secret\_key configurado (generalmente en CONFIG.Py o en la aplicación principal con app.config[“secret\_key”]).

…**SEGUIDO**

[**-@login\_manager.user**](mailto:-@login_manager.user)**\_loade**r: se debe usar el decorador para definir una función que retorne un Usuario , en la aplicación principal preferiblemente ,donde la extensión va a usar lo que retorne para implementar la variable global **current\_user**, el código es tal y como sigue

@login\_manager.user\_loader

def load\_user(user\_id):

    return Usuarios.query.get(int(user\_id))

**\*\*comentario:** el load\_user es cargado con user\_id una vez el usuario se loguéa con el

login\_user (objeto\_usuario).

**-login\_user:** es un método que crea la sesión se le debe pasa como argumento el objeto usuario del suplido por le ORM , a continuación se mostrara un pieza del código de logueo

Donde se muestra subrayado en amarillo el uso de este método.

if form.validate\_on\_submit():

        usuario=Usuarios.query.filter\_by(username=form.username.data).first()

        if usuario==None:

            form.username.errors.append(f"usuario {form.username.data} no existe")

            return render\_template("login.html", form =form, url='log\_in')

        print(form.data)

        if usuario.verify\_password(form.password.data):

            print("LOGUEADO?")

            #crea la session de usuario con los datos del usuario y carga con info del usuario a @login\_manager.userloader

            login\_user(usuario)

            print(f"RESPUESTA:{current\_user.is\_authenticated}")

            return redirect(url\_for("view\_inicio"))

        else:

            form.password.errors.append("Contraseña incorrecta, por favor intente de nuevo")

    return render\_template("login.html", form =form, url='log\_in')

-**logout\_user:** sirve para cerrar session(elimina los datos en sessión). Su implementación

Como sigue en el controlador logout de la app.

@app.route("/logout/", methods=['GET','POST'])

def log\_out():

    logout\_user()

    return redirect(url\_for("view\_inicio"))

**-current user:** Es una variable que se declara con el objeto del usuario logueado, donde se define la funcion quela genera y ademas también en todas las plantillas para acceder al objeto del usuario.ejemplo de uso en la app.

@app.route("/signin/", methods=['GET','POST'])

def signin():

    if current\_user.is\_authenticated:

        return redirect(url\_for("view\_inicio"))

-**LOGIN\_REQUIRED:** este es una function decoradora que se usa sobre los controladores donde se tengan rutas o templates para los cuales es necesario estar logueado. Un Ejemplo

@app.route("/editar/perfil/<int:id\_usuario>", methods=["GET","POST"])

@login\_required

def editar\_perfil(id\_usuario):

    usuario=Usuarios.query.get(id\_usuario)

    form=SigninForm(request.form,obj=usuario)

    del form.password

    del form.cpassword

    if usuario is None:

        abort(404,message='usuario no existe')

    if form.validate\_on\_submit():

        data\_form={key:value

                        for k in Usuarios.\_\_dict\_\_.keys() for key,value in form.data.items() if key==k}

        try:

            Usuarios.query.filter\_by(id=id\_usuario).update(data\_form)

            db.session.commit()

            login\_user(usuario)

            return redirect(url\_for('view\_inicio'))

        except:

            form.username.errors.append("username ya existente")

    return render\_template("signin.html", form=form, editar=True, url="editar\_perfil")

**IMPLEMENTACION EN MODELS.PY**

En el modelo de usuarios es necesario implementar una serie de propiedades y metodo que exige el flask login para su correcta implementacion las cuales son:

 @property

    def is\_authenticated(self):

        return True

    @property

    def is\_anonymous(self):

        return False

    @property

    def is\_active(self):

        return True

    #metodo requisito

    def get\_id(self):

        return self.id

    #extra, NO requisito para Flask login

    def is\_admin(self):

        return self.admin

**ALTERNATIVA A FLASK LOGIN**

Se puede implementar un sistema de autenticación y control de acceso de manera más “nativa” con el uso de **session** de **Flask** , teniéndose que implementar funciones (en caso del proyecto fueron implementadas en el script **Login.py**) para logueo,logout, verificar si esta logueago o no.

La implementación en un script aparte es como sigue .

from flask import session, redirect #importacion de session

from typing import Optional

def set\_login(Usuario: object = ...) -> None :

    """ Este metodo setea keys en el objeto session de Flask como

        id , username , admin"""

    session["id"]=Usuario.id

    session["username"]=Usuario.username

    session["admin"]=Usuario.admin

def set\_logout() -> None :

    """ Este metodo elimina con pop las keys id, username y admin de

        session de flask """

    session.pop("id")

    session.pop("username")

    session.pop("admin")

def is\_login() -> bool:

    """ Mediante la variable session de flask, esta funcion informa si hay un

        usuario logueado """

    if "id" in session:

        return True

    else:

        return False

def is\_admin() -> bool:

    """ Metodo que dice si el usuario logueado es Admin o nó """

    return session.get("admin")

**\*\*comentario:** Metodos a ser implementado en la aplicación principal.

**Caso de uso en la aplicación principal app.py**

**Importar funciones del modulo**

from flask import Flask, render\_template,jsonify,Response, request, url\_for, redirect, abort

from flask\_bootstrap import Bootstrap

from flask\_sqlalchemy import SQLAlchemy

from werkzeug.utils import secure\_filename

from app.login import set\_login, set\_logout

import json

import os

**Uso de set\_login en controlador log\_in para crear la sesión.**

@app.route("/login/", methods=['GET','POST'])

def log\_in():

    form=LoginForm()

    if form.validate\_on\_submit():

        usuario=Usuarios.query.filter\_by(username=form.username.data).first()

        if usuario==None:

            form.username.errors.append(f"usuario {form.username.data} no existe")

            return render\_template("login.html", form =form, url='log\_in')

        print(form.data)

        if usuario.verify\_password(form.password.data):

            set\_login(usuario)

            return redirect(url\_for("view\_inicio"))

        else:

            form.password.errors.append("Contraseña incorrecta, por favor intente de nuevo")

    return render\_template("login.html", form =form, url='log\_in')

**Uso de set\_logout en controlador log\_out para desloguearse**

@app.route("/logout/", methods=['GET','POST'])

def log\_out():

    set\_logout()

    return redirect(url\_for("view\_inicio"))

**\*\*comentario**: is\_login y is\_admin son metodos alternativos al uso de session para comprobar rol y logueo con la difencia de que los primeros tienen que ser pasados al contexto de los templates por un controlador, a diferencia de session que esta **automáticamente disponible en todos** los templates

**Caso de uso de session para verificar rol admin (control de acceso desde el template inicio.html)**

 {% if session["admin"] %}

              <td>

                <a href="{{url\_for('update\_form')}}?id={{e.id}}" > <button class="bg-dark text-white">modificar</button>

                </a>

                <!-- <input class="rounded bg-primary" type="submit" name="modificar" value="modificar" ria-label="modificar"></a> -->

              </td>

              <td>

                <a href="{{url\_for('delete\_articulo',id=e.id)}}"><button class="bg-dark text-white">borrar</button></a>

              </td>

              {% endif %}

**Caso de uso de session para verificar que la sesión exista.**

 {% if session["id"] %}

              <td>

                <a href=#><button class="bg-dark text-white">Comprar</button></a>

              </td>

              {% endif %}

**\*\*comentario:** en realidad se puede usar cualquier variable, ya sea username, id o admin para verificar la creación de la session.

**\*\*comentario:** cabe destacar que se debe impleentar un control de acceso también en el back ya que se puede ingresar manualmente las direcciones url a aquellos recursos restringidos por lo que se debe implementar condicionales que eviten dicho acceso como comprobar **si es admin: si continua con la ejecución del controlador, No se redirecciona el flujo con un abort o un redirect a un template correspondiente.**