Example

De: *Jackeline*

Nota: Este es un ejemplo sobre como crear una aplicación web con Flask y socket, también contiene bases de datos en relación a ciertos id, para comunicación de las tablas, usando como modulo sqlite3, todo en Python3.

Modulos y su intalación:

En python3, existen varios módulos para programar aplicaciones web, una de ellas es Flask, su instalación es muy sencilla:

$ pip3 install flask

Aparte de este módulo se usó otro, que es sockets para flask, su instalación es:

$ pip3 install flask_socketio

Y otro más, que es sqlite3:

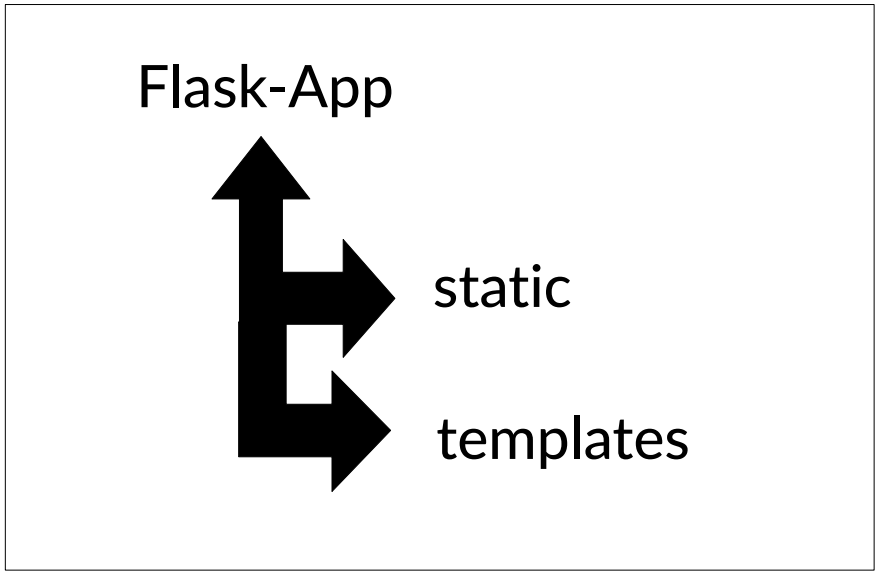
$ pip3 install sqlite3

Estos serían los tres principales, aparte de eso, utilizamos otros módulos que vienen por defecto:

- time
- os
- random
- re

Carpetas estáticas:

El módulo de flask trabaja con dos carpetas que se mantienen estáticas, aunque claro, existe módulos que cambian esta forma de hacerlo, pero aún así, usaremos esto, pues lo otro es un concepto más avanzado, y el árbol de carpetas es esto:



static:

-En esta carpeta generalmente se coloca archivos, como imagenes, videos, css, js, etc.

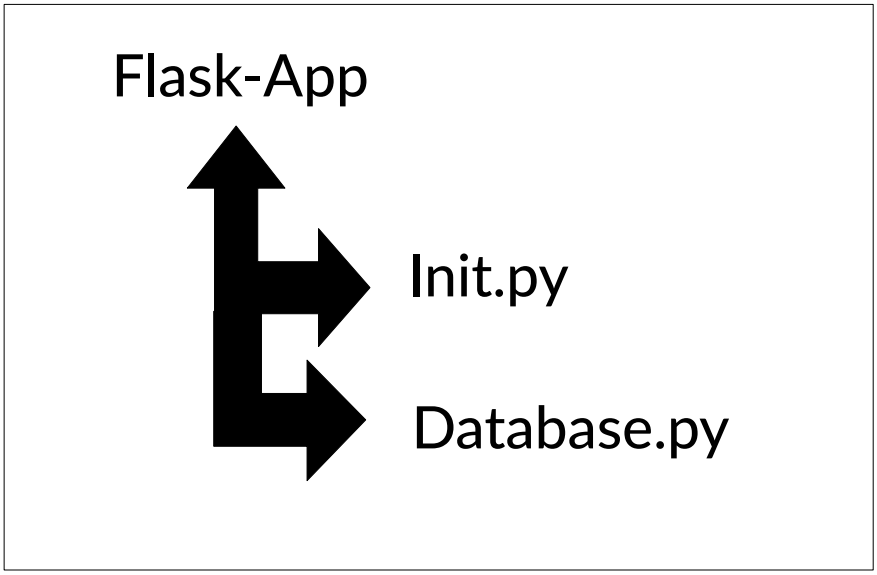
Templates:

-En esta carpeta se coloca la plantillas html para ser cargadas en el navegador.

Recomiendo hacerlo en este orden al comenzar a hacer una aplicación web, de esta forma, también podrás manejar fácilmente los datos para poder modificarlos y así no te perderás en un sin fin de archivos.

Archivos:

Para que la aplicación funcione, en mi caso dividí la programación en dos archivos:



De esta forma, hago que el código sea más claro, para que no se confunda entre las bases de datos, y el código de la aplicación con Flask.

init.py:

-Este archivo contiene los datos de la aplicación de flask, carga los archivos html, css, y lo sube a un servidor local.

database.py:

-En este archivo coloqué las funciones para crear, leer, actualizar y eliminar los datos de la base de datos, utilizando expresiones regulares y sqlite3.

Bases de datos:

La base de datos utiliza **sqlite3** y regex, o expresiones regulares con el módulo **re**:

```
#regex o expresiones regulares
nick_re     = re.compile(r"^[a-zA-Z0-9]{4,12}$")
age_re      = re.compile(r'^((19|20)\d\d)-(0?[1-9]|1[012])-(0?[1-9]|[12][0-9]|3[01])$')
email_re    = re.compile(r"\b[\w.%+-]+@[\w.-]+\.[a-zA-Z]{2,6}\b")
password_re = re.compile(r"^[a-zA-Z0-9\W\w]{7,30}$")
url_re      = re.compile(r"^(https?:\/\/)?([\da-z\.-]+)\.([a-z\.]{2,6})([\/\w \.-]*)*\/?$")
ip_re       = re.compile(r"^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$")
profile_re  = re.compile(r"^(images)/([a-zA-Z0-9\W]{1,60})+\.(webp|jpeg|png|jpg)$")
tel_peru_re = re.compile(r"^.+(51)([0-9\w]{9})$")
direction_re= re.compile(r"^[a-zA-Z0-9]{3,30}$")
#---------
```

De esta forma, comenzamos tenemos listo nuestras principales variables, lo siguiente es crear la base de datos:

```
def createDB():
    db_app = sqlite3.connect("database.db", check_same_thread=False)
    cursor = db_app.cursor()
    cursor.execute("""CREATE TABLE "accounts" (
                        "user_name" TEXT NOT NULL,
                        "user_surname"  TEXT NOT NULL,
                        "email" TEXT NOT NULL UNIQUE,
                        "password"  TEXT NOT NULL,
                        "phone" INT NOT NULL UNIQUE,
                        "direction" TEXT NOT NULL,
                        "age"   TEXT NOT NULL,
                        "user_id"   INT NOT NULL UNIQUE);""")

    db_app.commit()
    db_app.close()
```

Y así sucesivamente el resto de las bases de datos, para el manejo de errores, lo dividimos en diferentes funciones, y lo instanciamos en otra función:

```
def createAllDB():
    try:
        createDB()
        computerDB()
        moveDB()
        incidentsDB()
        imagesDB()
    except Exception as e:
        raise e
```

Bases de datos:

[redacted]

Creamos los primeros algoritmos conectados a la base de datos, en primer coso, uno conectado la tabla accounts:

Tables (5)		
accounts		CREATE TABLE "accounts" ("user_nar
user_name	TEXT	"user_name" TEXT NOT NULL
user_surname	TEXT	"user_surname" TEXT NOT NULL
email	TEXT	"email" TEXT NOT NULL UNIQUE
password	TEXT	"password" TEXT NOT NULL
phone	INT	"phone" INT NOT NULL UNIQUE
direction	TEXT	"direction" TEXT NOT NULL
age	TEXT	"age" TEXT NOT NULL
user_id	INT	"user_id" INT NOT NULL UNIQUE
computerDB		CREATE TABLE "computerDB" ("codig
imagesDB		CREATE TABLE "imagesDB" ("url" TEX
incidentsDB		CREATE TABLE "incidentsDB" ("codig
moveDB		CREATE TABLE "moveDB" ("codigo" T

Así que hacemos una función que nos ingresa 8 datos:

```
def account_signup(user_name, user_surname, email, password, telephone, direction, age):
    db_app = sqlite3.connect("database.db", check_same_thread=False)
    cursor = db_app.cursor()
    if nameRETURN(email)==None and str(nick_re.search(user_name))!=None and str(nick_re.search(user_surn
        user_id=random.randint(99999,99999999)
        content=[(user_name), (user_surname), (email), (password), (telephone), (direction), (age), (use
        cursor.execute("INSERT or IGNORE INTO accounts (user_name,user_surname, email, password, phone,
        db_app.commit()
        db_app.close()
        return True
    else:
        db_app.close()
        return False
```

Estos datos son analizados con el modulo re y las expresiones regulares y otros módulos para datos que no deben ser ingresados dos veces, como un mismo correo, luego de esto, los datos son guardados en la tabla, y retorna True, si es los datos no son correctos, retorna False.

Bases de datos:

La siguiente función proporciona el nombre en la cual se tiene registrado un correo, así como sirve para retornar el nombre, también sirve para verificar si existe un correo ya registrado:

```
def nameRETURN(correo):
    db_app = sqlite3.connect("database.db", check_same_thread=False)
    cursor = db_app.cursor()
    account_data=cursor.execute('SELECT * FROM accounts ORDER BY _rowid_').fetchall()
    for i in account_data:
        if correo==i[2]:
            db_app.close()
            return i[0]
        elif correo==i[0]:
            db_app.close()
            return i[0]
    db_app.close()
```

La siguiente función sirve para editar los datos de una tabla:

```
def accountUPDATE(data_update, code_user, data_modific, password): #return True or None
    db_app = sqlite3.connect("database.db", check_same_thread=False)
    cursor = db_app.cursor()
    for i in return_all():
        if  i[3]==str(password) and str(i[6])==str(code_user):
            cursor.execute("UPDATE accounts set {0}='{1}' where user_id='{2}'".format(d
            db_app.commit()
            db_app.close()
            return True

    db_app.close()
```

Si todo fue correcto, retorna True, de lo contrario retorna None. ahora queda implementarlo a Flask, ya que tenemos la primera parte de la base de datos, entonces creamos los primeros componentes en el archivo:

-init.py

Bases de datos y flask:

Creamos el login:

```python
@app.route("/login", methods=["GET","POST"])
def login():
    if 'emailXZK' in session:
        usuario=nameRETURN(escape(session["emailXZK"]))
        return render_template("home/home.html", user_name=usuario)
    else:
        if request.method == "POST":
            email=request.form.get('email');
            password=request.form.get('password');
            if password != None or email != None:
                AL=account_login(email=email, password=password)
                if AL == None:
                    return render_template("login/login.html", style_warning="text-warning text-red", text_warning="La cuenta no
                elif AL == True:
                        session['emailXZK']=email

                        if "nickname" in session:
                            return redirect('/home')
                        else:
                            return render_template("login/login.html", style_warning="text-warning text-red", text_warning="Ocurr
                else:
                    return render_template("login/login.html", style_warning="text-warning text-white", text_warning="Ocurrió un
        return render_template("login/login.html")
```

Al principio establecemos una url, que es /login, luego de esto, establecemos una condicional que verifica si existe coockies, si existe, nos manda redirecciona a /home, digamos que cuando se ejecuta una url, entra adentro de la función, luego de esto, si no es así, nos muestra en el navegador la página web, html, y existe nuevamente una condicional, si el método http envía POST, entra en esa condicional, que obtiene datos de los formularios email y password, estos datos son verificados nuevamente con una condicional, si no están vacíos, ejecuta una función de la base de datos que arriba no fue mencionado, pero esa función básicamente verifica si los datos son correctos, y retorna True o False, si retorna True, crea un coockie y nos manda a /home.

Bases de datos y flask:

El registro es básicamente lo mismo:

```
@app.route("/signup", methods=["GET","POST"])
def signup():
    if request.method == "POST":

        user_name=request.form.get('user_name');
        user_surname=request.form.get('user_surname');
        email=request.form.get('email');
        password=request.form.get('password');
        telephone=request.form.get('telephone');
        direction=request.form.get('direction');
        age=request.form.get('age');

        if user_name!=None or user_surname!=None or email!=None or password!=None or telephone!=None or
            signup=account_signup(user_name, user_surname, email, password, telephone, direction, age)
            if signup==True:
                return render_template("login/login.html", style_warning="text-warning text-green", text
            else:
                return render_template("signup/signup.html", style_warning="text-warning text-red", text

    return render_template("signup/signup.html", style_warning="text-warning text-white", text_warning="
```

Si pudiste comprender lo otro, también podrás comprender esto, solo que aquí utilizamos la función de la base de datos del registro anteriormente mencionado, si todo fue correcto, nos envía al /login con un mensaje, si algo falló, recarga nuevamente el /signup

```
@app.route("/home", methods=["GET","POST"])
def home():
    if 'emailXZK' in session:
        usuario=nameRETURN(escape(session["emailXZK"]))
        return render_template("home/home.html", user_name=usuario)
    else:
        return redirect("/login")
```

En el /login, cuando creemos una cuenta en /signup, podemos logearnos con el correo y contraseña, para así poder acceder al /home, que verifica si existe coockies, si existe, nos muestra la página con nuestro nombre, si no hay coockies, nos reenvía nuevamente al /login.

localhost: [redacted]

Aun no acabo, pero enviaré este documento para que lo veas.