PROYECTO FINAL PYTHON FULL STACK DEVELOPER IBM

AUTOR: RAFAEL SOLÍS LÓPEZ
PERFIL: www.linkedin.com/in/rafael-solis-lopez
GITHUB: **https://github.com/RAFASOLIS/python_full_stack**

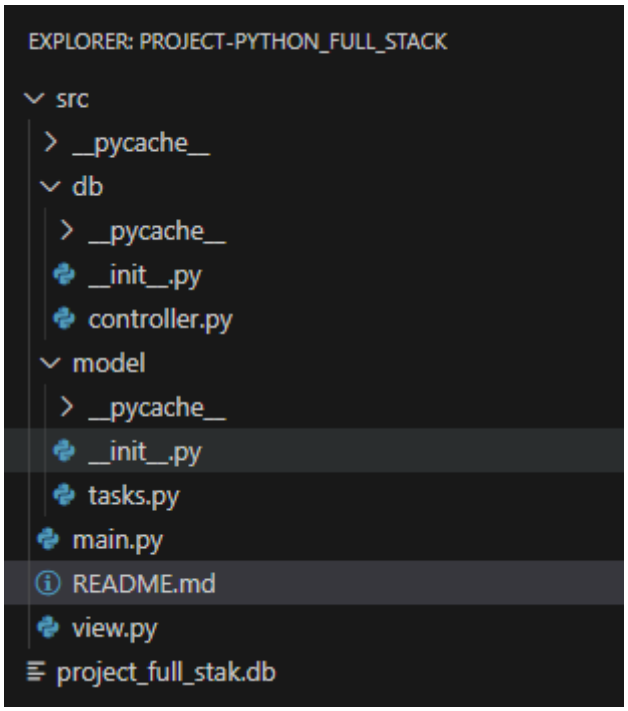
ÍNDICE

1. DESCRIPCIÓN DEL PROYECTO

El proyecto desarrolla en Python a través del IDE Visual Studio Code una aplicación que permite realizar operaciones CRUD en una base de datos embedida de tipo SQLite. El usuario final puede interactuar con la aplicación a través de la consola siguiendo las instrucciones de un menú de opciones que se muestra al ejecutar el script “main.py”.

2. ESTRUCTURA DEL PROYECTO

El proyecto se ha estructurado de la siguiente forma siguiendo un patrón MVC (Modelo, Vista, Controlador). La lógica de cada modelo se ha encapsulado en una clase.



3. DETALLE DEL PROYECTO

A continuación se detalla el código con comentario de cada uno de los scripts que componen el proyecto.

Módulo Controlador"

```
import os
import sqlite3 as sql

class Controller():
    """Clase que contiene los métodos para interactuar con la BBDD SQLite
Embedida"""
    def __init__(self):
        self.CLASS_NAME="Controller"
        self.db_name="project_full_stak.db"

    def deployDB(self):
        """Método público.Método que crea la bbdd, la tabla tarea y
realiza los inserts iniciales"""
        fun_name="deployDB"
        try:
            self.createDB() #Creamos BBDD
            self.createTable() #Creamos tabla TAREA
            self.initialInserts() # Creamos inserts iniciales
            print("Executed {}/{}".format(self.CLASS_NAME,fun_name))
        except Exception as e:
            raise Exception("ERROR in  {}/{} msg:
{}".format(self.CLASS_NAME,fun_name,e))

    def createDB(self):
        """Método que crea la BBDD"""
        fun_name="createDB"
        try:
            conn=sql.connect(self.db_name)#Conectamos a la BBDD
            conn.commit()#Efectuamos la confirmación para generarl el
fichero "project_full_stack_db"
            conn.close() #Cerramos conexión
            print("Executed {}/{}".format(self.CLASS_NAME,fun_name))
        except Exception as e:
           raise Exception("ERROR in  {}/{} msg:
{}".format(self.CLASS_NAME,fun_name,e))
```

```
    def createTable(self):
        """Método público.Método que crea la tabla TAREA en la BBDD """
        fun_name="createTable"
        try:
            ddl="""
            CREATE  TABLE TAREA (
            ID_TASK INTEGER PRIMARY KEY AUTOINCREMENT,
            TASK_NAME VARCHAR(100),
            STATUS VARCHAR(50),
            CREATED_DATE DATETIME,
            UPDATE_DATE DATETIME
            );"""
            conn=sql.connect(self.db_name) #Creamos conexión a BBDD
            cursor=conn.cursor() #Creamos un cursos para poder ejecutar
sentencias SQL
            cursor.execute("DROP TABLE IF EXISTS TAREA") #Borramos lat
abla si existe previamente
            cursor.execute(ddl) #Crea la tabla
            conn.commit() # Confirma las instrucciones
            conn.close() #Cerramos conexión
            print("Executed {}/{}".format(self.CLASS_NAME,fun_name))
        except Exception as e:
           raise Exception("ERROR in  {}/{} msg:
{}".format(self.CLASS_NAME,fun_name,e))


    def initialInserts(self):
        """Método público.Método que genera los inserts iniciales sobre
la tabla TAREA"""
        fun_name="initialInsert"
        try:
            conn=sql.connect(self.db_name)#Creamos conexión a BBDD
            cursor=conn.cursor() #Creamos cursor para poder ejecutar
instrucciones SQL sobre la BBDD
            #Lanzamos las diferentes instrucciones SQL
            cursor.execute("DELETE FROM TAREA;")
            cursor.execute("INSERT INTO TAREA
(TASK_NAME,STATUS,CREATED_DATE,UPDATE_DATE)
VALUES('TAREA1','PENDING',DATETIME('now'),DATETIME('now'));")
            cursor.execute("INSERT INTO TAREA
(TASK_NAME,STATUS,CREATED_DATE,UPDATE_DATE)
VALUES('TAREA2','PENDING',DATETIME('now'),DATETIME('now'));")
            cursor.execute("INSERT INTO TAREA
(TASK_NAME,STATUS,CREATED_DATE,UPDATE_DATE)
VALUES('TAREA3','PENDING',DATETIME('now'),DATETIME('now'));")
            cursor.execute("INSERT INTO TAREA
(TASK_NAME,STATUS,CREATED_DATE,UPDATE_DATE)
VALUES('TAREA4','PENDING',DATETIME('now'),DATETIME('now'));")
```

```
            cursor.execute("INSERT INTO TAREA
(TASK_NAME,STATUS,CREATED_DATE,UPDATE_DATE)
VALUES('TAREA5','PENDING',DATETIME('now'),DATETIME('now'));")
            conn.commit() #Confirmamos las instrucciones
            conn.close() #Cerramos conexión
            print("Executed {}/{}".format(self.CLASS_NAME,fun_name))
        except Exception as e:
           raise Exception("ERROR in  {}/{} msg:
{}".format(self.CLASS_NAME,fun_name,e))

    def __get_stmt(self,type_stmt,task_name="",status="",id_task=-1):
        """Función privada.Función que recibe el timpo de instrucción SQL
a ejecutar y devuelve su sentencia SQL montada"""
        fun_name="get_stmt"
        try:
            stmt_select="SELECT *  FROM TAREA WHERE ID_TASK
={}".format(id_task)
            stmt_insert="INSERT INTO TAREA
(TASK_NAME,STATUS,CREATED_DATE,UPDATE_DATE)
VALUES('{}','PENDING',DATETIME('now'),DATETIME('now'));".format(task_name
)
            stmt_update="UPDATE TAREA SET
STATUS='{}',UPDATE_DATE=DATETIME('now') WHERE ID_TASK
={}".format(status,id_task)
            stmt_delete="DELETE FROM TAREA WHERE ID_TASK
={}".format(id_task)
            #Creamos un diccionario para casar tipo de sentencia con su
isntrucción SQL generada

stmt={"select":stmt_select,"insert":stmt_insert,"update":stmt_update,
"delete":stmt_delete}
            print("Executed {}/{}".format(self.CLASS_NAME,fun_name))
            return stmt[type_stmt] #rdevolvemos la sentencia SQL a
ejecutar
        except Exception as e:
           raise Exception("ERROR in  {}/{} msg:
{}".format(self.CLASS_NAME,fun_name,e))

    def list_all(self):
        """Método público.Método que lista todos los registros de la
tabla TAREA"""
        fun_name="list_all"
        try:
            conn=sql.connect(self.db_name)#Creamos conexión
            cursor=conn.cursor() # Creamos cursos para ejecutar la
instrucción SQL
            query=f"SELECT * FROM TAREA" #Definimos la instrucción SQL
            cursor.execute(query) #Ejecutamos la instrucción SQL
            rows=cursor.fetchall() #Recuperamos los datos desde BBDD
```

```
            conn.commit() #Confirmamos la ejecución
            conn.close() #Cerramos conexión
            print("Executed {}/{}".format(self.CLASS_NAME,fun_name))
            return rows
        except Exception as e:
           raise Exception("ERROR in  {}/{} msg:
{}".format(self.CLASS_NAME,fun_name,e))




    def execute_stmt(self,type_stmt,task_name="",status="",id_task=-1):
        fun_name="execute_stmt"
        try:

stmt=self.__get_stmt(type_stmt,task_name=task_name,status=status,id_task=
id_task)
            conn=sql.connect(self.db_name)
            cursor=conn.cursor()
            cursor.execute(stmt)
            if type_stmt:
                rows=cursor.fetchall()
            else:
                rows=None
            conn.commit()
            conn.close()

            print("Executed {}/{}".format(self.CLASS_NAME,fun_name))
            return rows
        except Exception as e:
           raise Exception("ERROR in  {}/{} msg:
{}".format(self.CLASS_NAME,fun_name,e))


if __name__=="__main__":
    print("controller")
```

Módulo “Modelo”

```
from db.controller import Controller

class Model():
    """Clase que gestiona la lógica del proyecto entre el ususario y la
BBDD"""
    def __init__(self):
        """Constructor"""
```

```
        self.CLASS_NAME="Model"
        self.controller=Controller()#Instanciamos un objecto de clase
Controller() para comunicarnos con BBDD

    def __print_rows(self,rows):
        """Método privado.Este método printa las finas recibidas por
argumento"""
        frame="*"*100
        print(frame)
        print("Table: TAREA")

headers="ID_TASK","TASK_NAME","STATUS","CREATED_DATE","UPDATE_DATE"
        print(headers)
        for row in rows:
            print(row)
        print(frame)

    def list_all(self):
        """Método público.Método que recupera todas la filas de la tabla
TAREA y las printa en consola"""
        fun_name="list_all"
        try:
            print("Executing {}/{}".format(self.CLASS_NAME,fun_name))
            rows=self.controller.list_all()#Recuperamos los datos
            self.__print_rows(rows)#Printamos los datos
            print("Executed {}/{}".format(self.CLASS_NAME,fun_name))

        except Exception as e:
            raise Exception("ERROR in  {}/{} msg:
{}".format(self.CLASS_NAME,fun_name,e))

    def show_task(self,id_task):
        """Método público.Método que recupera la fila asociada al código
de tarea recibido por arguemnto"""
        fun_name="show_task"
        try:
            print("Executing {}/{}".format(self.CLASS_NAME,fun_name))

rows=self.controller.execute_stmt(type_stmt="select",id_task=id_task)#Rec
upera datos
            self.__print_rows(rows)#Printa los datos
            print("Executed {}/{}".format(self.CLASS_NAME,fun_name))
        except Exception as e:
            raise Exception("ERROR in  {}/{} msg:
{}".format(self.CLASS_NAME,fun_name,e))


    def update_task(self,id_task,status):
```

```
        """Método públio.Método que actualiza en la BBDD la tarea
asociada al código de tarea reacibido"""
        fun_name="update_task"
        try:
            print("Executing {}/{}".format(self.CLASS_NAME,fun_name))

stmt=self.controller.execute_stmt(type_stmt="update",id_task=id_task,stat
us=status)#Ejecuta la instrucción en BBDD
            print("Executed {}/{}".format(self.CLASS_NAME,fun_name))
        except Exception as e:
            raise Exception("ERROR in  {}/{} msg:
{}".format(self.CLASS_NAME,fun_name,e))


    def insert_task(self,task_name):
        """Método público.Método que inserta una nueva tarea en la tabla
TAREA"""
        fun_name="insert_task"
        try:
            print("Executing {}/{}".format(self.CLASS_NAME,fun_name))

stmt=self.controller.execute_stmt(type_stmt="insert",task_name=task_name)
#Ejecuta la instrucción en BBDD
            print("Executed {}/{}".format(self.CLASS_NAME,fun_name))
        except Exception as e:
            raise Exception("ERROR in  {}/{} msg:
{}".format(self.CLASS_NAME,fun_name,e))

    def delete_task(self,id_task):
        """Método público.Método que borra la tarea asociada al código de
tarea redibido por arugmento"""
        fun_name="delete_task"
        try:
            print("Executing {}/{}".format(self.CLASS_NAME,fun_name))

stmt=self.controller.execute_stmt(type_stmt="delete",id_task=id_task)#Eje
cuta la instrucción en BBDD
            print("Executed {}/{}".format(self.CLASS_NAME,fun_name))
        except Exception as e:
            raise Exception("ERROR in  {}/{} msg:
{}".format(self.CLASS_NAME,fun_name,e))
```

Módulo “Vista”.

```
from model.tasks import Model
from db.controller import Controller


class Menu():
    def __init__(self):
        self.CLASS_NAME="View"
        self.model=Model()
        self.TEXT_MENU="""
            MENU:

            0. Crear BBDD (esta opción borrará registros
insertados/actualizados desde la creación anterior).
            1. Listar todas las tareas.
            2. Listar una tarea.
            3. Añadir una tarea.
            4. Actualizar una tarea.
            5. Borrar Una tarea.
            6. Salir del menú.
            """

    def start(self):
        fun_name="start"
        try:
            op=-1
            while op != 6:
                print(self.TEXT_MENU)
                op=int(input("Introduzca opción:"))
                if(op==0):
                    self.__op_0_createDB()
                elif(op==1):
                    self.__op_1_list_all_tasks()
                elif(op==2):
                    self.__op_2_list_taks()
                elif(op==3):
                    self.__op_3_insert_task()
                elif(op==4):
                    self.__op_4_update_task()
                elif(op==5):
                    self.__op_5_delete_task()
                elif(op<0  or op>6):
                    print("El número de opción no está incluida en el
menú.")
        except Exception as e:
                raise Exception("ERROR in  {}/{} msg:
{}".format(self.CLASS_NAME,fun_name,e))


    def __op_0_createDB(self):
```

```
        controller=Controller()
        controller.deployDB()

    def __op_1_list_all_tasks(self):
        fun_name="__op_1_list_all_tasks"
        try:
            self.model.list_all()
        except Exception as e:
                raise Exception("ERROR in  {}/{} msg:
{}".format(self.CLASS_NAME,fun_name,e))

    def __op_2_list_taks(self):
        fun_name="__op_2_list_taks"
        try:
            id_task=int(input("Introduzca el código de la tarea a
mostrar:"))
            self.model.show_task(id_task=id_task)
        except Exception as e:
                raise Exception("ERROR in  {}/{} msg:
{}".format(self.CLASS_NAME,fun_name,e))

    def __op_3_insert_task(self):
        fun_name="__op_3_insert_task"
        try:
            task_name=input("Introduzca el nombre de la tarea a
insertar:")
            self.model.insert_task(task_name=task_name)
            self.model.list_all()
        except Exception as e:
                raise Exception("ERROR in  {}/{} msg:
{}".format(self.CLASS_NAME,fun_name,e))

    def __op_4_update_task(self):
        fun_name="__op_4_update_task"
        try:
            id_task=int(input("Introduzca el código de la tarea a
actualizar:"))
            status=input("Introduzca el nuevo estado de la tarea a
actualizar:")
            self.model.update_task(id_task=id_task,status=status)
            self.model.list_all()
        except Exception as e:
                raise Exception("ERROR in  {}/{} msg:
{}".format(self.CLASS_NAME,fun_name,e))

    def __op_5_delete_task(self):
        fun_name="__op_5_delete_task"
        try:
```

```
            id_task=int(input("Introduzca el código de la tarea a
borrar:"))
            self.model.delete_task(id_task=id_task)
            self.model.list_all()
        except Exception as e:
                raise Exception("ERROR in  {}/{} msg:
{}".format(self.CLASS_NAME,fun_name,e))
```

Clase principal

```
from view import Menu
""" Script principal """
menu=Menu() #Instanciamos un objecto de la clase Menu
menu.start() #Arrancamos el proceso1
```

4. DEMO DEL PROYECTO

Introducimos la opción 0

```
            MENU:

            0. Crear BBDD (esta opción borrará registros insertados/actualizados desde la creación anterior).
            1. Listar todas las tareas.
            2. Listar una tarea.
            3. Añadir una tarea.
            4. Actualizar una tarea.
            5. Borrar Una tarea.
            6. Salir del menú.

 Introduzca opción:0
 Executed Controller/createDB
 Executed Controller/createTable
 Executed Controller/initialInsert
 Executed Controller/deployDB
```

Introducimos la opción 1

```
Introduzca opción:1
Executing Model/list_all
Executed Controller/list_all
*********************************************************************************************
Table: TAREA
('ID_TASK', 'TASK_NAME', 'STATUS', 'CREATED_DATE', 'UPDATE_DATE')
(1, 'TAREA1', 'PENDING', '2024-05-22 10:22:08', '2024-05-22 10:22:08')
(2, 'TAREA2', 'PENDING', '2024-05-22 10:22:08', '2024-05-22 10:22:08')
(3, 'TAREA3', 'PENDING', '2024-05-22 10:22:08', '2024-05-22 10:22:08')
(4, 'TAREA4', 'PENDING', '2024-05-22 10:22:08', '2024-05-22 10:22:08')
(5, 'TAREA5', 'PENDING', '2024-05-22 10:22:08', '2024-05-22 10:22:08')
*********************************************************************************************
Executed Model/list_all
```

Introducimos la opción 2

```
Introduzca opción:2
Introduzca el código de la tarea a mostrar:2
Executing Model/show_task
Executed Controller/get_stmt
Executed Controller/execute_stmt
********************************************************************************************************
Table: TAREA
('ID_TASK', 'TASK_NAME', 'STATUS', 'CREATED_DATE', 'UPDATE_DATE')
(2, 'TAREA2', 'PENDING', '2024-05-22 10:22:08', '2024-05-22 10:22:08')
********************************************************************************************************
Executed Model/show_task
```

Introducimos la opción 3

```
Introduzca opción:3
Introduzca el nombre de la tarea a insertar:"TAREA6"
Executing Model/insert_task
Executed Controller/get_stmt
Executed Controller/execute_stmt
Executed Model/insert_task
Executing Model/list_all
Executed Controller/list_all
********************************************************************************************************
Table: TAREA
('ID_TASK', 'TASK_NAME', 'STATUS', 'CREATED_DATE', 'UPDATE_DATE')
(1, 'TAREA1', 'PENDING', '2024-05-22 10:22:08', '2024-05-22 10:22:08')
(2, 'TAREA2', 'PENDING', '2024-05-22 10:22:08', '2024-05-22 10:22:08')
(3, 'TAREA3', 'PENDING', '2024-05-22 10:22:08', '2024-05-22 10:22:08')
(4, 'TAREA4', 'PENDING', '2024-05-22 10:22:08', '2024-05-22 10:22:08')
(5, 'TAREA5', 'PENDING', '2024-05-22 10:22:08', '2024-05-22 10:22:08')
(6, '"TAREA6"', 'PENDING', '2024-05-22 10:23:26', '2024-05-22 10:23:26')
********************************************************************************************************
```

Introducimos la opción 4

```
Introduzca opción:4
Introduzca el código de la tarea a actualizar:1
Introduzca el nuevo estado de la tarea a actualizar:"SUCCESS"
Executing Model/update_task
Executed Controller/get_stmt
Executed Controller/execute_stmt
Executed Model/update_task
Executing Model/list_all
Executed Controller/list_all
********************************************************************************************************
Table: TAREA
('ID_TASK', 'TASK_NAME', 'STATUS', 'CREATED_DATE', 'UPDATE_DATE')
(1, 'TAREA1', '"SUCCESS"', '2024-05-22 10:22:08', '2024-05-22 10:24:13')
(2, 'TAREA2', 'PENDING', '2024-05-22 10:22:08', '2024-05-22 10:22:08')
(3, 'TAREA3', 'PENDING', '2024-05-22 10:22:08', '2024-05-22 10:22:08')
(4, 'TAREA4', 'PENDING', '2024-05-22 10:22:08', '2024-05-22 10:22:08')
(5, 'TAREA5', 'PENDING', '2024-05-22 10:22:08', '2024-05-22 10:22:08')
(6, '"TAREA6"', 'PENDING', '2024-05-22 10:23:26', '2024-05-22 10:23:26')
********************************************************************************************************
```

Introducimos la opción 5

```
Introduzca opción:5
Introduzca el código de la tarea a borrar:1
Executing Model/delete_task
Executed Controller/get_stmt
Executed Controller/execute_stmt
Executed Model/delete_task
Executing Model/list_all
Executed Controller/list_all
********************************************************************************************
Table: TAREA
('ID_TASK', 'TASK_NAME', 'STATUS', 'CREATED_DATE', 'UPDATE_DATE')
(2, 'TAREA2', 'PENDING', '2024-05-22 10:22:08', '2024-05-22 10:22:08')
(3, 'TAREA3', 'PENDING', '2024-05-22 10:22:08', '2024-05-22 10:22:08')
(4, 'TAREA4', 'PENDING', '2024-05-22 10:22:08', '2024-05-22 10:22:08')
(5, 'TAREA5', 'PENDING', '2024-05-22 10:22:08', '2024-05-22 10:22:08')
(6, '"TAREA6"', 'PENDING', '2024-05-22 10:23:26', '2024-05-22 10:23:26')
********************************************************************************************
```

Introducimos la opción 7

```
                0. Salir del menú.

Introduzca opción:7
El número de opción no está incluida en el menú.
```

Introducimos la opción 6

```
Introduzca opción:6
PS C:\Users\Rafael\Desktop\Python FS\project-python_full_stack>
```