SISTEMAS DE GESTIÓN EMPRESARIAL

CFGS Desarrollo de Aplicaciones Multiplataforma
Informática y Comunicaciones

PROYECTO PYTHON

Nombre y Apellidos: Estela de Vega Martín.
Curso: 2024/2025

Índice

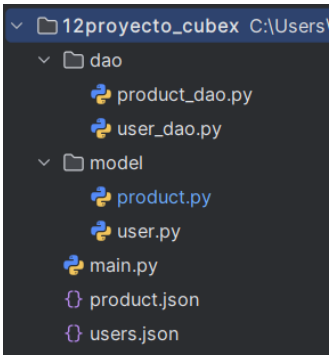
Introducción

Se ha ampliado la idea de una tienda de cubos de Rubik en javafx, así que se ha desarrollado una aplicación en Python que permita la gestión de productos y usuarios en una tienda de cubos de Rubik. Los usuarios tienen la capacidad de crear, modificar, listar y eliminar productos, así como gestionar su propia cuenta. La aplicación también incluye la compra de productos.

Entorno de trabajo

Se ha utilizado el lenguaje de programación Python junto con el entorno de desarrollo PyCharm. Para la gestión de datos, se ha empleado la biblioteca json. Además, se implementó Git para hacer un seguimiento de los cambios realizados en el código.

Estructura de clases



- Main: donde tendrá todo el código principal.
- Product: donde se definirá la clase de producto. Dispondrá de los atributos de nombre, precio, stock, categoría, propietario y los métodos para escribir la información del objeto y el método para convertir de objeto a diccionario.
- User: donde se definirá la clase de usuario. Tendrá los atributos de nombre, email, y contraseña. Además de los métodos para escribir la información del objeto y el método para convertir de objeto a diccionario.
- User_dao: donde se definirá todo el crud de usuario. Gestionará el listar usuarios, crear, iniciar sesión, modificar y eliminar.
- Product_dao: se definirá todo el crud de producto. Gestionará el listar la información de productos, crear, modificar, eliminar y comprar producto.

Documentación técnica

La estructura principal se ha organizado en el archivo main, el cual contiene cuatro funciones, además del código que inicia la aplicación. De esta forma, se permite tener una mayor legibilidad a la hora de leer el código, dividendo las funcionalidades según las opciones elegidas por el usuario, evitando que el código se convierta en un único bloque enorme.

Las funciones desarrolladas contienen la lógica para:

- **Iniciar sesión.**
- **Crear una cuenta.**
- **El menú del apartado de usuarios.**

- **El menú del apartado de productos.**

Gracias a esta forma, el código principal se mantiene más limpio y claro. Además de la legibilidad de código, se ha implementado un manejo de excepciones para gestionar posibles errores que puedan pasar durante la ejecución de la aplicación.

```
try:
    # INICIAR SESION
    while True:
        print("¿Cómo quieres entrar a nuestra tienda?\n1-Iniciando sesión\n2-Creando una cuenta")
        metodo_login = int(input())
        # CONTROLAR QUE PONGA LA OPCION 1 O 2
        while metodo_login > 2 or metodo_login < 1:
            metodo_login = int(input("Introduzca una opcion entre 1 y 2\n"))

        if metodo_login == 1:
            email = method_login()  # LLAMAR AL METODO PARA INICIAR SESION
            if email is not None: # SI NO ES NULO SE SALE DEL BUCLE (se encontro usuario)
                break
        elif metodo_login == 2:
            email = method_signup()  # LLAMAR AL METODO PARA CREAR CUENTA
            if email is not None: # SI SE CREA BIEN LA CUENTA SALE DEL BUCLE
                break

    # MENU PRINCIPAL DESPUES DEL INICIO DE SESION
    print("\n\n¡Bienvenido a CubeX! ¿Qué desea hacer en nuestra tienda?" +
          "\n1-Ir al apartado de usuarios" +
          "\n2-Ir al apartado de cubos")
    opcion_apartado = int(input())
    if opcion_apartado == 1:
        # APARTADO USUARIOS
        apartado_usuario()  # LLAMAR AL METODO DE APARTADO USUARIOS

    elif opcion_apartado == 2:
        # APARTADO CUBOS
        apartado_productos()  # LLAMAR AL METODO DE APARTADO PRODUCTOS
except Exception as e:
    print(f"Ocurrió un error: {e}")
print(";Vuelva pronto!")
```

Aplicación

Inicio

Al iniciar la aplicación, lo primero que le saldrá será la forma en la que quiere acceder a la aplicación, si iniciando sesión o creando una cuenta. Esta opción estará validada para que no introduzca una opción no valida.

Inicio de sesión

Cuando el usuario elige la opción de iniciar sesión, se invoca a la función que solicita su mail y contraseña. Esta función utiliza también el método dao que abre el archivo json donde se almacenan los usuarios, el programa recorre la lista de usuarios y busca un registro que coincida con el correo y contraseña proporcionados. Si se encuentra el usuario, devuelve true, si no, devuelve false.

Si el inicio de sesión es exitoso, el sistema guarda en un atributo el atributo que devuelva la función de iniciar sesión (el mail). Si no encuentra el usuario, se mostrará un mensaje de error y volverá a mostrar el menú de acceder a la aplicación.

```python
# FUNCION PARA LOGGEAR USUARIOS
def log_in(user_obj: user):  1 usage  estelaV9
    user_data = leer_archivo()

    for users in user_data:
        if users['Email'] == user_obj.email and users['Password'] == user_obj.password:
            # SI ENCUENTRA AL USUARIO CON ESE MAIL Y ESA CONTRASEÑA ENTONCES SE PODRA LOGGEAR
            return True

    return False  # SI NO HA ENCONTRADO USUARIO DEVOLVERA FALSE
```

Crear cuenta

Sigue un proceso similar al método de iniciar sesión. Si se selecciona esta opción, invoca a la función que gestiona la creación de cuentas, solicitando el nombre, mail y contraseña. La función verifica si ya existe un usuario con esos datos, si existe, devuelve false, si no, se añade a la lista y se actualiza el archivo json. Si la creación de cuenta fue exitosa se le dará un mensaje de bienvenida con un menú de la tienda, de lo contrario, se le mostrará nuevamente el menú de acceso a la aplicación.

```python
# FUNCION PARA CREAR USUARIOS Y GUARDARLOS EN JSON
def create_user(user_obj: user):  1 usage  estelaV9
    user_data = user_obj.to_dict()  # CONVERTIMOS EL OBJETO EN UN DICCIONARIO
    users = leer_archivo()

    # BUCLE PARA SABER SI EL USUARIO YA TENIA UNA CUENTA
    for user_exist in users:
        if user_exist['Email'] == user_obj.email and user_exist['Password'] == user_obj.password:
            # SI ENCUENTRA AL USUARIO CON ESE MAIL Y ESA CONTRASEÑA RETORNA FALSE Y NO SE CREARA LA CUENTA
            return False

    users.append(user_data)  # AGREGAR EL NUEVO USUARIO A LA LISTA

    # ESCRIBIR LA LISTA DE USUARIO EN EL ARCHIVO DE JSON
    with open('users.json', 'w') as file:
        json.dump(users, file, indent=4)  # Indent para mejor legibilidad
    return True
```

Tienda

Una vez que accede el usuario, ya sea mediante el inicio de sesión o la creación de una cuenta, se le mostrara un menú de dos opciones: apartado de usuario y apartado de productos.

Cada uno de ellos contiene la lógica relacionada con la gestión de usuario o a productos.

Apartado usuarios

El apartado de usuarios gestiona todas las operaciones relacionadas con la información de los usuarios registrados en la tienda. Cuando el usuario seleccione esta opción, se llamará a la función que contiene la lógica del apartado usuarios, donde se le mostrara el menú con las siguientes opciones:

1 - Ver todos los usuarios de la tienda.

Esta opción permite visualizar todos los **usuarios registrados** en la tienda. Se llama a una función que leerá todo el archivo json de usuarios y recorrerá la lista mostrando la información de cada usuario.

```python
# FUNCION PARA IMPRIMIR TODOS LOS USUARIOS
def list_user():  1 usage  estelaV9
    list_user_show = leer_archivo()

    # RECORRER LA LISTA Y MOSTRARLOS
    for user in list_user_show:
        print(user.__str__())
```

2 - Ver la información de tu usuario.

Se llamará a la función de **listar información del usuario**. Esta función leerá todo el archivo json de usuarios y recorrerá la lista hasta encontrar el nombre de usuario deseado, y después le mostrará su información específica.

```python
# FUNCION PARA IMPRIMIR LA INFO DEL USUARIO
def list_user_only(user_name):  1 usage  estelaV9
    list_user_show = leer_archivo()  # GUARDAR LOS USUARIOS

    # RECORRER LA LISTA Y MOSTRAR SU INFO
    for user in list_user_show:
        if user['Name'] == user_name:  # SI ES EL NOMBRE DE USUARIO ENTONCES MUESTRA SU INFORMACION
            print(user.__str__())
```

3 - Modificar datos del usuario.

Se llama a la función de **modificar** usuarios. La función leerá todo el archivo json de usuarios hasta encontrar el usuario y le solicitara los nuevos datos, los cuales se actualizan y se escriben nuevamente en el archivo.

```python
def modify_user(user_name):  1 usage  estelaV9
    users = leer_archivo()  # GUARDAR LOS USUARIOS

    # RECORRER LA LISTA
    for users in users:
        if users['Name'] == user_name:
            # MODIFICAR DATOS
            new_name = input("Nombre: ")
            new_mail = input("Email: ")
            new_password = input("Password: ")
            # SETTEAR NUEVOS VALORES
            users['Name'] = new_name
            users['Email'] = new_mail
            users['Password'] = new_password
            break

    # ESCRIBIR LA NUEVA INFORMACION EN EL FICHERO
    with open('users.json', 'w') as file:
        json.dump(users, file, indent=4)
    print(f"Usuario '{user_name}' se ha modificado exitosamente.")
```

4 - Eliminar usuario.

Al seleccionar esta opción, se invoca a la función de **eliminar** usuario. Esta función leerá todo el archivo json de usuarios y recorrerá la lista y cuando encuentre al nombre del usuario a eliminar, lo elimina de la lista y actualiza el archivo con la nueva información.

```python
# FUNCION PARA ELIMINAR USUARIO
def delete_user(user_name):  1 usage  estelaV9
    list_user = leer_archivo()

    for users in list_user:
        # SI EL NOMBRE DE USUARIO ES EL INTRODUCIDO SE ELIMINARA
        if users['Name'] == user_name:
            list_user.remove(users)  # SE ELIMINA EL USUARIO
            break  # SE SALE DEL BUCLE

    with open('users.json', 'w') as file:
        json.dump(list_user, file, indent=4)

    print(f"Usuario '{user_name}' se ha eliminado exitosamente.")
```

0 - Salir de la tienda.

Esta opción permite al usuario **salir** la aplicación.

Apartado productos

El apartado de productos gestiona todas las operaciones relacionadas con los productos de la tienda de cubos de Rubik. A través de este menú, los usuario pueden realizar acciones como visualizar, crear, modificar, eliminar productos y comprar.

Opciones del Menú

1 - Ver todos los productos disponibles

Esta opción permite a los usuarios visualizar una **lista de todos los productos disponibles en la tienda**. Se llama a una función que lee el archivo JSON donde están almacenados los productos, recorre la lista y muestra la información de cada uno. Se muestran todos los productos disponibles de la tienda.

```python
# FUNCION PARA LISTAR TODOS LOS PRODUCTOS
def list_product():  1 usage  estelaV9
    products = leer_archivo()
    # RECORRER LA LISTA
    for product in products:
        print(product.__str__()) # IMPRIMIR LOS DATOS
```

2 - Ver tus productos

Al seleccionar esta opción, los usuarios pueden ver la **lista de productos que han creado**. Se implementa una función que filtra los productos basándose en el usuario logueado y muestra solo aquellos que le pertenecen.

```python
# FUNCION PARA LISTAR LOS PRODUCTOS DE UN USUARIO
def list_product_user(user_name):  1 usage  estelaV9
    products = leer_archivo()
    # RECORRER LA LISTA
    for product in products:
        if product['owner'] == user_name:
            print(product.__str__()) # RETORNA LOS DATOS
```

3 - Comprar productos

Esta funcionalidad permite a los usuarios realizar **compras** de productos disponibles. Se implementa un sistema que verifica la disponibilidad del producto, reduce el stock y actualiza el inventario en el archivo JSON correspondiente. Esto incluye validaciones para garantizar que el producto esté en stock.

```python
# FUNCION PARA COMPRAR PRODUCTO
def buy_product(product_name):  1 usage  estelaV9
    product_list = leer_archivo()

    # BUSCAR NOMRBE PRODUCTO
    product_found = False  # SABER SI SE HA ENCONTRADO EL PRODUCTO
    stock_disponible = True  # SABER SI HAY STOCK DISPONIBLE

    for product in product_list:
        if product['nombre'] == product_name:  # SI HAY UN PRODUCTO CON ESE NOMBRE
            product_found = True

            # SI EL STOCK NO ES CERO SE MODIFICA EL STOCK
            if product['stock'] != 0:
                new_stock = product['stock'] - 1  # SE LE RESTA 1 AL STOCK
                product['stock'] = int(new_stock)  # SE MODIFICA EL STOCK
                break
            else:
                stock_disponible = False  # SI NO HAY STOCK DEVUELVE FALSE

    if product_found and stock_disponible:  # SI EXISTE ESE PRODUCTO Y HAY STOCK DISPONIBLE ENTONCES SE GUARDA
        # SE GUARDA LA LISTA ACTUALIZADA DE VUELTA AL JSON
        with open('product.json', 'w') as file:
            json.dump(product_list, file, indent=4)
        print(f"Producto '{product_name}' comprado exitosamente.")
    elif not stock_disponible:  # SI NO HAY STOCK DISPONIBLE SALTA UN MENSAJE
        print("No hay stock disponible.")
    elif not product_found:  # SI ESE PRODUCTO NO EXISTE, SALTA UN MENSAJE
        print(f"No se encontró el producto '{product_name}'")
```

4 - Crear productos

Los usuarios pueden **agregar** nuevos productos a la tienda utilizando esta opción. Se llama a una función que solicita al usuario la información necesaria para el nuevo producto y valida los datos de tipo numérico con una función lambda.

Para escribir el valor de las categorías, se pasa una tupla de las categorías que hay, ya que no se pueden modificar. Si los datos son correctos, se añade el nuevo producto al archivo JSON y se muestra un mensaje de confirmación.

```python
# FUNCION PARA CREAR UN PRODUCTO
def create_product(email):  1 usage  estelaV9
    product_name = input("Nombre del producto: ")
    product_price = input("Precio del producto: ")
    # VALIDAR LOS DATOS NUMERICOS QUE SEAN POSITIVOS Y NUMEROS
    if not validar_numero(product_price):
        print("El valor ingresado no es un número positivo válido.")
        return False
    stock = int(input("Stock disponible: "))
    # VALIDAR LOS DATOS NUMERICOS QUE SEAN POSITIVOS Y NUMEROS
    if not validar_numero(product_price):
        print("El valor ingresado no es un número positivo válido.")
        return False

    # MOSTRAR LAS CATEGORIAS PARA QUE ELIJA
    # RECORRER LA TUPLA CON LA FUNCION DE 'enumerate' LA CUAL DEVUELVE UN OBJETO ENUMERADO
    for idx, category in enumerate(categories, 1):
        print(f"{idx}. {category}")

    # SE VALIDA QUE INTRODUZCA BIEN EL NUMERO DE LA CATEGORIA
    while True:
        try:
            category_option = int(input("Introduce el número de la categoría: "))
            # SE COMPRUEBA QUE LA CATEGORIA SEA MAYOR O IGUAL A 1 Y QUE SEA MENOR O IGUAL A LA LONGUITUD DE LA TUPLA
            if 1 <= category_option and category_option <= len(categories):
                categoria = categories[category_option - 1] # SI ES CORRECTO SE AÑADE LA OPCION
                break
            else:
                print("Elección inválida, elige un número válido.") # SI NO ES CORRECTO SALTA UN ERROR
        except ValueError:
            print("Debes ingresar un número válido.") # SE VALIDA QUE NO PONGA LETRAS

    # CREAR EL NUEVO PRODUCTO
    new_product = product(product_name, product_price, stock, categoria, email)

    with open('product.json', 'r') as file:
        list_product_create = json.load(file)  # GUARDAR TODOS LOS PRODUCTOS EN LISTA

    product_data = new_product.to_dict()
    list_product_create.append(product_data)  # GUARDAR EL NUEVO PRODUCTO EN FORMA DE DICCIONARIO EN LA LISTA

    with open('product.json', 'w') as file:
        json.dump(list_product_create, file, indent=4)  # Indent para mejor legibilidad
```

5 - Modificar productos

Esta opción permite a los usuarios **modificar** la información de productos existentes. Se llama a una función que solicita el nombre del producto a modificar y los nuevos datos. Después de realizar las validaciones, se actualiza la información en el archivo JSON.

```python
def modify_product(product_name):  1 usage  estelaV9
    # BUSCAR NOMBRE PRODUCTO
    product_found = False  # SABER SI SE HA ENCONTRADO EL PRODUCTO
    for product in product_list:
        if product['nombre'] == product_name:  # SI HAY UN PRODUCTO CON ESE NOMBRE SE MODIFICA
            product_found = True

            # SE SOLICITA LOS NUEVOS DATOS PARA EL PRODUCTO
            new_name = input("Nombre: ")
            new_price = input("Precio: ")
            # VALIDAR LOS DATOS NUMERICOS QUE SEAN POSITIVOS Y NUMEROS
            if not validar_numero(new_price):
                print("El valor ingresado no es un número positivo válido.")
                return False
            new_stock = input("Stock disponible: ")
            # VALIDAR LOS DATOS NUMERICOS QUE SEAN POSITIVOS Y NUMEROS
            if not validar_numero(new_stock):
                print("El valor ingresado no es un número positivo válido.")
                return False

            # MOSTRAR LAS CATEGORIAS PARA QUE ELIJA
            # RECORRER LA TUPLA CON LA FUNCION DE 'enumerate' LA CUAL DEVUELVE UN OBJETO ENUMERADO
            for idx, category in enumerate(categories, 1): # EMPIEZA EN UNO
                print(f"{idx}. {category}")

            # SE VALIDA QUE INTRODUZCA BIEN EL NUMERO DE LA CATEGORIA
            while True:
                try:
                    new_category = int(input("Introduce el número de la categoria: "))
                    # SE COMPRUEBA QUE LA CATEGORIA SEA MAYOR O IGUAL A 1 Y QUE SEA MENOR O IGUAL A LA LONGUITUD DE LA TUPLA
                    if 1 <= new_category and new_category <= len(categories):
                        categoria = categories[new_category - 1]  # SI ES CORRECTO SE AÑADE LA OPCION
                        break
                    else:
                        print("Elección inválida, elige un número válido.")  # SI NO ES CORRECTO SALTA UN ERROR
                except ValueError:
                    print("Debes ingresar un número válido.")  # SE VALIDA QUE NO PONGA LETRAS

            # SE MODIFICAN LOS ATRIBUTOS
            product['nombre'] = new_name
            product['precio'] = float(new_price)
            product['stock'] = int(new_stock)
            product['categoria'] = categoria

            break  # SE SALE DEL BUCLE DESPUES DE MODIFICAR

    if product_found:
        # SE GUARDA LA LISTA ACTUALIZADA DE VUELTA AL JSON
        with open('product.json', 'w') as file:
            json.dump(product_list, file, indent=4)
```

6 - Eliminar productos

Los usuarios pueden **eliminar** sus productos de la tienda utilizando esta opción. Se llama a una función que solicita el nombre del producto a eliminar, se recorre la lista y, si encuentra ese nombre lo elimina y reescribe el archivo json.

```python
# FUNCION PARA ELIMINAR PRODUCTO POR NOMBRE
def delete_product(product_name):  1 usage  estelaV9
    list_product_delete = leer_archivo()
    # BUSCAR NOMBRE PRODUCTO
    product_found = False  # SABER SI EL PRODUCTO EXISTE O NO

    for product in list_product_delete:
        if product['nombre'] == product_name:  # SI HAY UN PRODUCTO CON ESE NOMBRE SE ELIMINA
            list_product_delete.remove(product)  # ELIMINAR EL PRODUCTO DE LA LISTA
            product_found = True
            break  # SALIR DEL BUCLE

    if product_found:
        # SE GUARDA LA LISTA ACTUALIZADA DE VUELTA EN EL JSON
        with open('product.json', 'w') as file:
            json.dump(list_product_delete, file, indent=4)
        print(f"Producto '{product_name}' eliminado exitosamente.")
    else:
        print(f"No se encontró el producto '{product_name}'.")
```

0 - Salir de la tienda

Esta opción permite a los usuarios **salir** del apartado de productos y regresar al menú principal de la aplicación. Se asegura que todas las operaciones en curso se completen antes de cerrar el apartado de productos.

Conclusiones

Como conclusiones finales, cabe recalcar que Python me ha demostrado que es un lenguaje de programación más complejo de lo que en teoría parece, ya que incluso una correcta elección del entorno de desarrollo es importante para poder tener más comodidades a la hora de hacer la aplicación.

Ampliaciones

Se podría implementar una interfaz gráfica inspirada en el ejercicio de JavaFX, ya que esto proporcionaría una experiencia de usuario más visual y atractiva, mejorando así, su interacción con la aplicación en comparación con la entrada de datos a través de pantallas de texto.

También se podría crear un apartado de récords actuales por categorías o las próximas competiciones que se vayan a planificar. Pero como no encontré ningún api de cubos de Rubik, la ampliación se centraría en crear yo el fichero json y ahí meter los datos.

Además, se podría implementar las funciones lambdas de orden superior para filtrar cubos o usuarios, las cuales había intención en implementar para este proyecto.

Bibliografía

1. **Ejercicios de clase.** Antes de empezar tue que volverme a mirar como creaban tuplas, set, lo que era un diccionario…
2. https://chatgpt.com. Antes de empezar y viendo que mi proyecto se orientaba sobre todo a la gestión de datos de los ficheros json, primero tuve que aprender a manejarme con ellos. Asiqué, una vez adquirido ese conocimiento, pude empezar con el proyecto.
3. https://j2logo.com/python/tutorial/tipo-tuple-python/#tuple-for

Enlace a GitHub

Para acceder al código de la práctica 12, pincha en el siguiente enlace:

https://github.com/estelaV9/SistemasGestionEmpresarial/tree/master/Tema3_Python/Python_SGE/12proyecto_cubex