**DICCIONARIO BACKEND JR**

En Python, una **Clase** es como un plano o plantilla para crear objetos.

Define **atributos (variables)** y **métodos (funciones)** que los objetos de esa clase tendrán.

Un **Módulo** es simplemente un archivo de Python (.py) que contiene código, clases, funciones y variables que puedes importar y reutilizar en otros archivos. Esto ayuda a organizar el código y evitar archivos gigantes.

Una **instancia** es un objeto concreto creado a partir de una clase o plantilla para crear objetos.

La **base de datos** es un sistema organizado para almacenar, gestionar y recuperar información de manera eficiente.

un **modelo** es la representación de una entidad del mundo real en tu código. Esta representación define tanto los datos (atributos o variables) como el comportamiento (métodos o funciones) de esa entidad.

Una **entidad** es un objeto o concepto del mundo real que es importante para el sistema que estás construyendo y del que necesitas almacenar información. Para un sistema de e-commerce, serían *Cliente, Producto, Pedido y Carrito de compras*.

Una **API** (Application Programming Interface o Interfaz de Programación de Aplicaciones) es un conjunto de reglas y protocolos que permite que dos aplicaciones de software se comuniquen entre sí.

**Introduciendo el Patrón Repositorio**

El **Patrón Repositorio** es una técnica de diseño que nos permite separar la lógica de la base de datos (la capa de persistencia) de la lógica de nuestra aplicación (la capa de negocio). En lugar de que nuestros endpoints de FastAPI interactúen directamente con las funciones de *sqlite3*, crearán una interfaz abstracta que manejará todas las operaciones de la base de datos.

**¿Por qué usarlo?**

* **Desacoplamiento:** Tu API ya no "sabe" que estás usando SQLite. Si decides cambiar a PostgreSQL en el futuro, solo necesitarás reescribir la implementación del Repositorio, dejando tu *main.py* intacto.
* **Código más Limpio:** La lógica de la base de datos se agrupa en un solo lugar, lo que hace que tu código sea más organizado y legible.
* **La capa de negocio:** Lógica de nuestra aplicación.
* **La capa de persistencia:** Lógica de la base de datos

**Próximo Desafío: Inyección de Dependencias**

Ahora que tienes tu repositorio, el siguiente paso es conectar el “*main.py*” con el “*repositories.py*” usando un patrón llamado Inyección de Dependencias.

**¿Qué es la Inyección de Dependencias?**

La **Inyección de Dependencias,** es un patrón que nos permite pasar las "dependencias" (en este caso, la conexión a la base de datos y el repositorio) a las funciones que las necesitan, en lugar de que estas funciones las creen por sí mismas. FastAPI tiene un sistema de Inyección de Dependencias muy potente.

**Autenticación y Seguridad**

Hasta ahora, cualquiera puede crear, ver, actualizar y eliminar usuarios. Para proteger nuestra API, implementaremos un sistema de autenticación basado en **JSON Web Tokens (JWT)**.

**¿Qué es?**

Nunca debes guardar las contraseñas de los usuarios en texto plano en la base de datos. Si tu base de datos fuera comprometida, todas las contraseñas quedarían expuestas. En su lugar, debemos **cifrar** (o "hashear") las contraseñas. Esto es un proceso de un solo sentido: se genera un *hash* a partir de la contraseña, y ese *hash* es el que se guarda. Para verificar la contraseña en el futuro, se cifra el texto que el usuario ingresa y se compara el *hash* resultante con el que está guardado.

**¿Por qué usarlo?**

Es una práctica de seguridad fundamental. Utilizaremos la librería **passlib** para este propósito, ya que es la más recomendada para FastAPI.

**Verificación de Contraseñas**

Ahora que puedes guardar contraseñas de forma segura, el siguiente paso es crear un **endpoint** de inicio de sesión que verifique si la contraseña ingresada por el usuario coincide con la que está guardada en la base de datos.

Para ello, utilizaremos el método *pwd\_context.verify()*. Este método toma la contraseña ingresada por el usuario y el *hash* guardado en la base de datos. No descifra el *hash*, sino que hashea la contraseña de entrada y compara el resultado con el *hash* almacenado.

**Autenticación con JWT**

**¿Qué es un JWT?**

Un JWT es una cadena de texto que codifica información (llamada *payload* o "carga útil"), como el ID del usuario. Este token está firmado digitalmente con una clave secreta del servidor. Cuando el usuario hace una petición a un endpoint protegido, envía este token. Tu API puede verificar si el token es auténtico (es decir, si fue firmado por tu servidor) sin tener que consultar la base de datos en cada petición.

**¿Por qué usarlo?**

* **Sin estado:** El servidor no necesita guardar información sobre la sesión de cada usuario, lo que reduce la carga del servidor y lo hace más escalable.
* **Seguridad:** El token está firmado, lo que garantiza que no fue alterado por un tercero.