ejemplo de arquitectura

readme.md

https://github.com/LillianaU/AgendaORMRelacional

explicación del entregable:

Programación orientado a objetos
Copia de taller de OOP en Python.docx, diseño de proyecto poo uml

Para este entregable, ustedes deberán diseñar **todos los diagramas de clases y de funcionalidad** de su sistema. Estos deben ser **coherentes con la matriz de requerimientos**, ya que así estarán avanzando en dos aspectos a la vez: la entrega de esta fase y la validación del diseño general del sistema.

Es **muy importante** que exista coherencia entre la matriz de requerimientos y los diagramas elaborados.

En esta actividad **solo deberán modelar el UML correspondiente a los diagramas de clases**. No es necesario realizar el diagrama de paquetes en esta entrega, ya que ese tema puede resultar más complejo. Si el tiempo lo permite durante la formación, les enseñaré cómo elaborarlo para que lo puedan implementar más adelante.

Por ahora, lo que deben entregar son los **diagramas de clases** que incluyan:

- **Herencia**
- **Encapsulamiento**
- **Polimorfismo** (si aplica al proyecto, ya que no todos lo requieren)
- **Relaciones**: uno a muchos, muchos a muchos, **composición** y **agregación**.

En resumen, este entregable corresponde al **modelado UML enfocado en los diagramas de clases**, asegurando que el diseño represente correctamente la estructura y relaciones del sistema.

Diagramas de clase de proyecto oop 3 módulos(esto se replanteo por que los proyectos son grades)

Hoy, 24 de octubre, estuvimos observando los sistemas que ustedes han desarrollado. Con la experiencia que tengo en programación —y también en documentación— puedo decirles que este software, como pudieron notar, cuenta con bastantes clases, alrededor de ocho o diez aproximadamente.

¿Qué les voy a pedir entonces? No les voy a exigir nada relacionado con **seguridad ni autenticación**, ya que esas clases pertenecen al **patrón de arquitectura** solicitado. Lo que sí deben entregar son **las clases diseñadas en UML**, únicamente.

Por ejemplo, si alguno de ustedes tiene **16 clases**, puede realizar una **integración parcial**. A medida que vayan mostrando avances durante el proceso, podemos **negociar entregas**. Sin embargo, si no observo progreso, **no habrá negociación** y deberán presentar **todos los diagramas** que correspondan.

Recuerden que ya me mostraron las **bases de datos** —excepto un grupo—, porque algunos solo entregaron los **procedimientos almacenados**, y a través de ellos uno puede identificar cuántas tablas están siendo manipuladas.

Chicos, avancemos, porque esta práctica es muy valiosa para su formación. Les servirá cuando trabajen con **frameworks como Django**, que utiliza **programación orientada a objetos**, **manejo de clases** y **programación funcional**. Todo esto fortalecerá su **lógica de desarrollo**.

Aclaro también que **hay posibilidad de reajustar el entregable**, siempre y cuando se evidencie **avance, compromiso e integridad**. Si observo que algunos se están cubriendo entre sí o que no están cumpliendo con su parte, entonces no habrá flexibilidad en la entrega.

retomando:

Ya se vio 🙂

- ☐ Herencia 🖌️
- ☐ Relaciones uno a uno, muchos a mucho, cero a muchos🖌️
- ☐ Encapsulamientos métodos y atributos🖌️
- ☐ Objetos pero no lo hicimos en diagramas🖌️

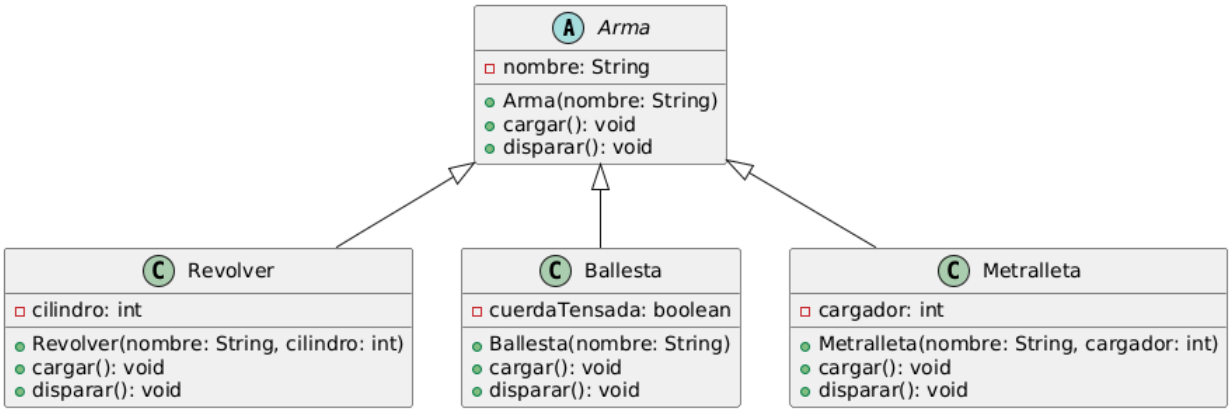
para hoy 24 de octubre del 2025 es

- ☐ Composición 🖌️
- ☐ agregación🖌️
- ☐ Polimorfismo 🖌️
- ☐ Diagrama de objetos🖌️

https://es.scribd.com/document/570180343/UML-2-5-Iniciacion-ejemplos-y-ejercicios-corregidos-4ta-edicion

https://www.lucidchart.com/pages/uml-class-diagram

https://support.microsoft.com/es-es/topic/crear-un-diagrama-de-clase-uml-de6be927-8a7b-4a79-ae63-90da8f1a8a6b



porque no me queda Bueno con lo que ustedes más le gusta vamos a hacer un ejemplo de videojuegos que son los tipos de armas no hace falta que tenga una ballesta que tengo una un revólver una metralleta cierto ustedes saben que tenemos la clase de arma dentro de la clase de arma va a heredar unos atributos 1 m 2 y encontramos que en cada tipo de arma que en este caso sería revolver ballesta y metralleta vemos que hay un dos polimorfismos cuál carga y disparar genial o sea está fantástico vamos a estar no el polimorrimo es que usted tiene un atributo vea la ballesta Se dispara muy diferente que una que un que una ametralladora pero checos vamos a estar mirando para cargarla si es diferente porque una vez balita por balita que ese revolver la maestra ya la ametralladora es un cartucho Y la ballesta es una flechita lo que vamos a estar analizando es disparar es igual Bueno aquí con mis jugadores profesionales el disparar una ballesta es igual vamos a analizar las imágenes

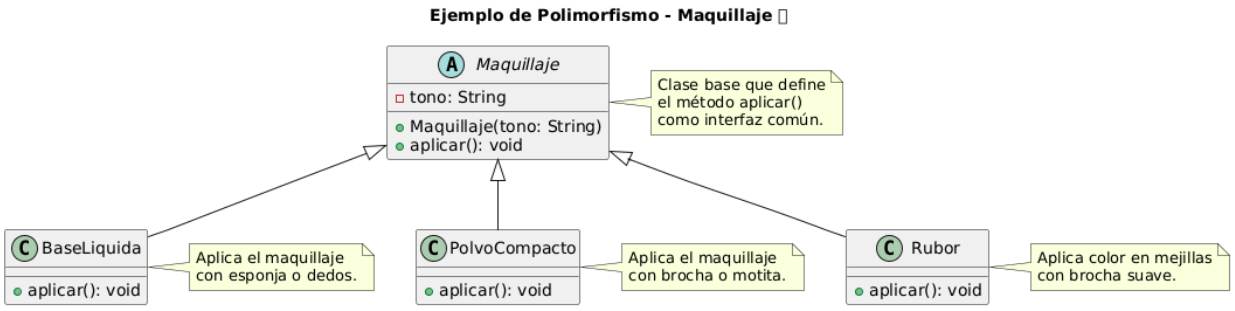
revolver

Clase	Método `disparar()`	Qué hace (simulación)
Revolver	`disparar()`	Simula un **disparo único**, como una acción puntual del cilindro.
Ballesta	`disparar()`	Solo dispara **si la cuerda está tensada**; luego la **destensa**, cambia su estado.
Metralleta	`disparar()`	Ejecuta una **ráfaga simulada**, varios disparos continuos, representando una acción repetitiva.

💡 Explicación textual

- **Clase base:** `Maquillaje` → define el método `aplicar()` (interfaz común).
- **Subclases:**
  - `BaseLiquida`: simula aplicarse con esponja.
  - `PolvoCompacto`: simula aplicarse con brocha.
  - `Rubor`: simula aplicarse en las mejillas con una brocha más suave.

Todas comparten el método `aplicar()`, pero cada una **actúa distinto**, aunque se llame igual.



Bueno chicos, hasta el momento vamos a tener en cuenta algunas **consideraciones importantes para elaborar el diagrama de clases** de su proyecto formativo.

Primero, recuerden que los **diagramas de clases no se normalizan**. Es decir, las clases que representan entidades creadas únicamente por procesos de normalización **no se consideran como clases independientes** en el diagrama.

Por ejemplo, si ustedes tienen una clase llamada **Usuario**, que representa la tabla `usuario` en la base de datos, y esta tiene una relación con **Roles**, deben saber que **Roles**, en la mayoría de modelos UML, **se maneja como un atributo del Usuario**, no como una clase aparte. En este caso, se considera un **atributo compuesto** o **relacional**, no una entidad.

Por eso es muy positivo que ustedes se estén apoyando en herramientas como **ChatGPT** para orientarse, ya que a veces lo que se ve en la base de datos no se refleja igual en el diagrama de clases, porque **UML trabaja desde la lógica de objetos**, no desde la estructura relacional.

Otro ejemplo: el **detalle de factura**. Este elemento hace parte de una **tercera forma normal** dentro del modelo relacional, pero **no se incluye como una clase independiente**, sino que se considera **parte de la clase Factura**. En otras palabras, todas las tablas creadas solo para lograr la normalización —como las tablas intermedias de muchos a muchos— **no deben modelarse como clases**. Son más bien **atributos o asociaciones** dentro de las clases principales.

Ahora bien, si un elemento de estos llega a tener **demasiados atributos propios o comportamientos específicos**, ahí sí se puede considerar como una clase aparte. Pero en principio, no.

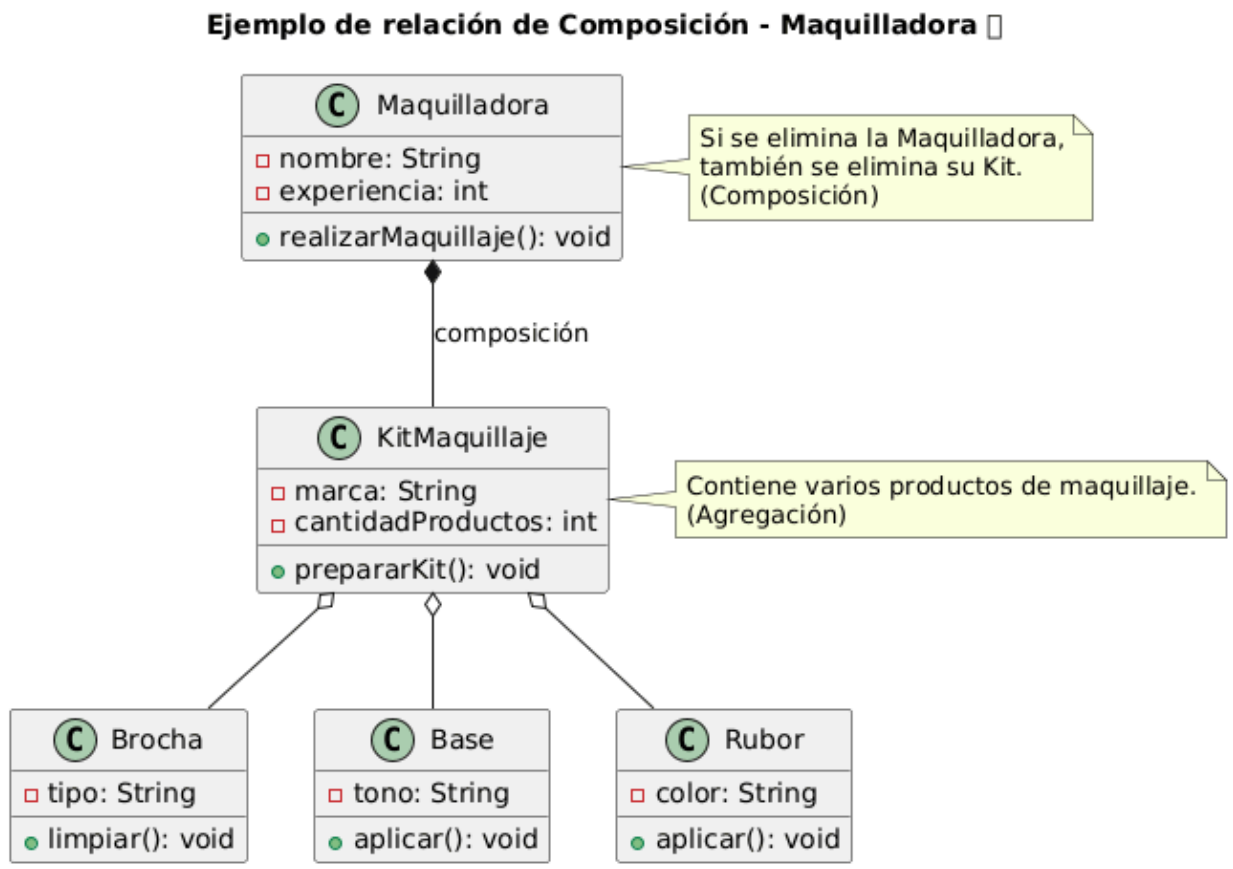
El **diagrama de base de datos** nos ayuda a identificar las clases “candidatas”, como una especie de borrador o pastel de ideas; sin embargo, quien realmente define las características de los atributos —si son públicos, privados o protegidos, o si hacen parte de una relación polimórfica— **es la matriz de requerimientos**. Por eso es fundamental que esa matriz esté **ya revisada y corregida**.

Quiero también aclarar algo sobre la evaluación:
Yo me comprometo a revisar hasta un **75 u 80 % de corrección** sobre las evidencias que ustedes me entreguen a tiempo. Si presentan sus entregas **de manera tardía**, no puedo garantizar que tengan los insumos suficientes para su **entrega de fase**.

Recuerden que **quien define el porcentaje y los criterios de la entrega de fase** es su **Gestor** y el **equipo ejecutor**. En la plataforma que les compartí hace ocho días está la **ruta de aprendizaje**, donde se especifican los porcentajes y requerimientos para la devolución del proyecto.

Finalmente, si el líder del programa considera que deben mostrar **avances adicionales**, dependerá de su competencia o del resultado de aprendizaje que estén desarrollando. En eso **yo no interfiero**, porque respeto la libertad de cátedra y los lineamientos del contrato de prestación de servicios.

Por eso, yo simplemente me guío por **lo que está definido en la ruta de aprendizaje y en la metodología**, y por las competencias que me corresponde acompañar. Los **diagramas de clases** son una herramienta lógica que les permite **desarrollar pensamiento orientado a objetos**, y por eso son parte esencial de su proceso de formación en programación.



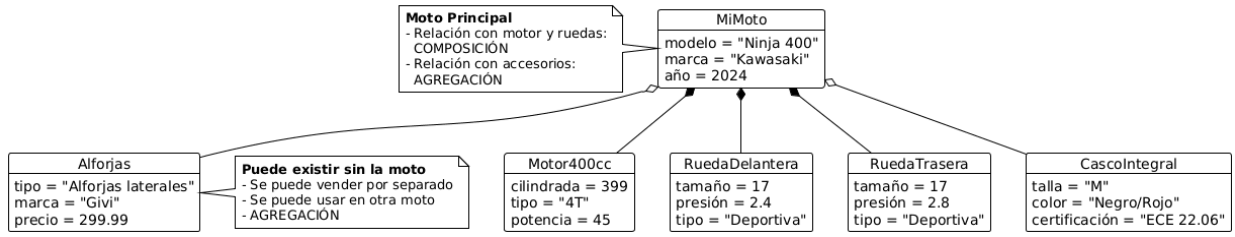
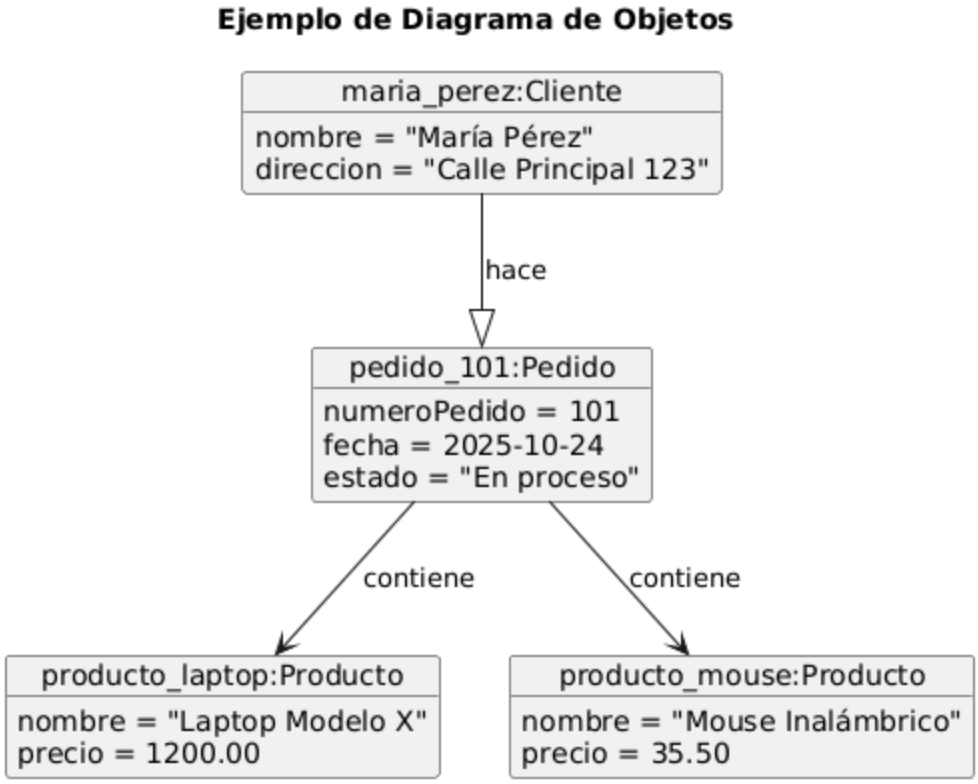
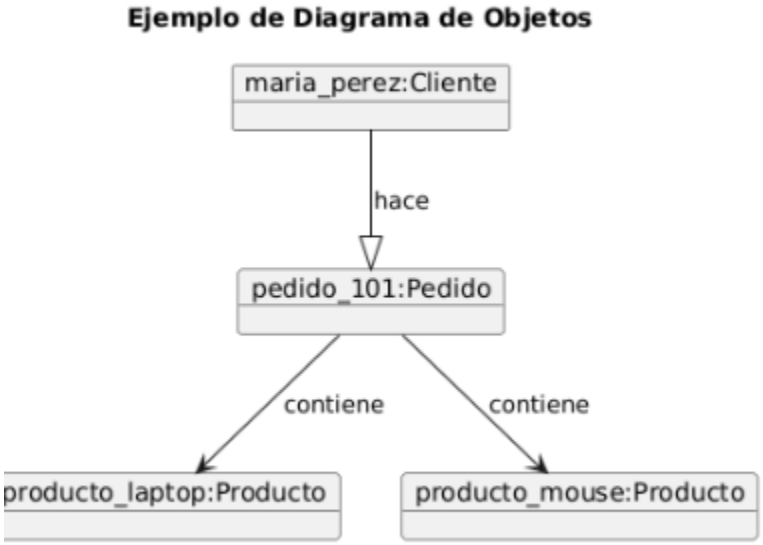


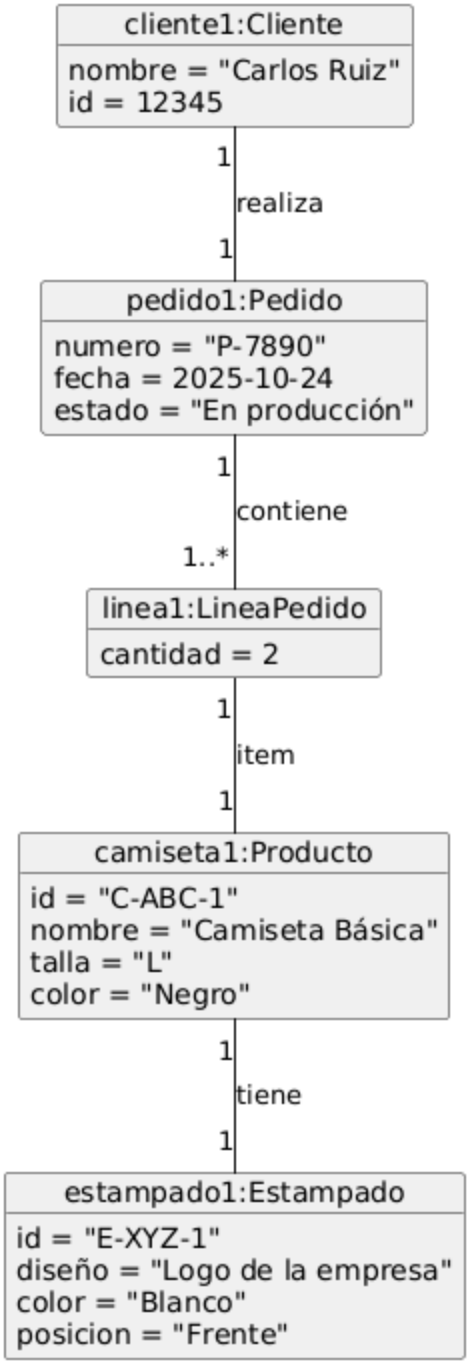
diagrama de objetos

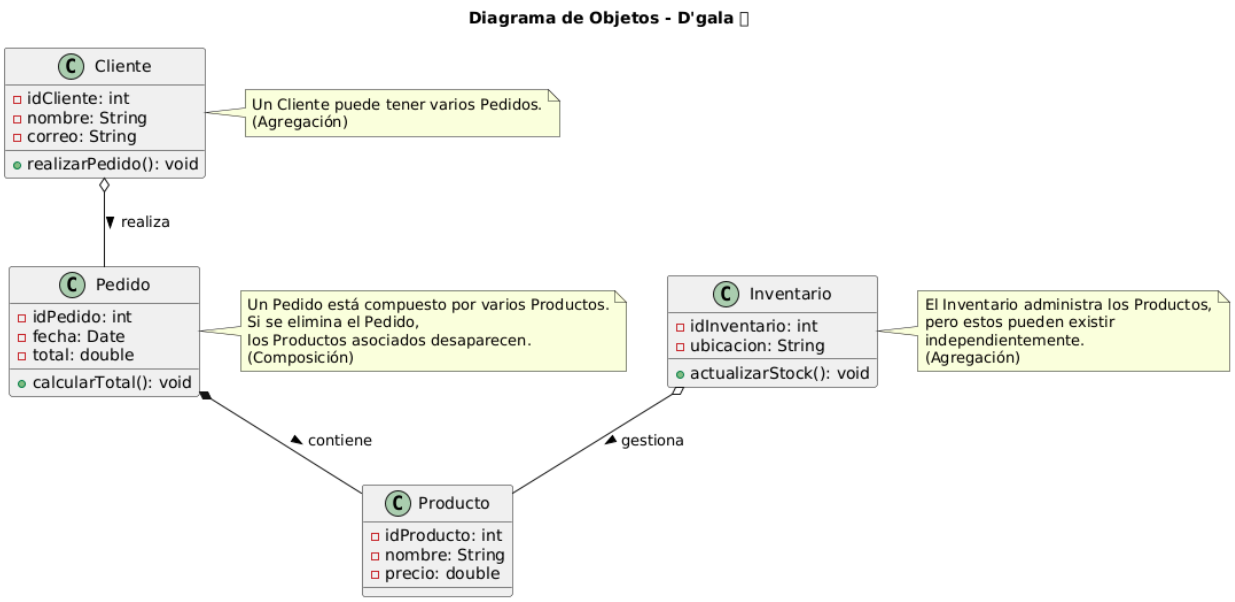




https://www.planttext.com/

Diagrama de Objetos: Pedido con Estampado





Chicos, con el ejemplo que ustedes ya tienen, voy a perfeccionar el pequeño modelo de arquitectura monolítica que les mostré, elaborando tanto la **documentación técnica** como el **manual de usuario**.
Esto lo hago para que ustedes tengan una **referencia clara** de cómo debe ir quedando su proyecto. Es un **plus** que les estoy brindando como guía, porque si los requerimientos no están bien desarrollados o corregidos, **los planes de prueba y las métricas de seguridad no saldrán adecuadamente**.

En este momento estamos trabajando en **arquitecturas de desarrollo de software**, y todas ellas tienen un componente fundamental: el **modelo**.
Ese es precisamente el módulo que estoy solicitando ahora, ya que no alcanzamos a implementar una arquitectura completa.
En este caso, trabajaremos bajo el enfoque **MVT (Modelo–Vista–Template)**.

No se preocupen por las demás clases, ya que su diseño dependerá del tipo de arquitectura que cada uno esté implementando.
Hago este paréntesis porque después me dicen:

> “Profe, ¿cómo hiciste lo de la autenticación?”

Y ahí es importante aclarar: en **Python** existen varios tipos de autenticación; personalmente me gusta **JWT**, que es muy útil aunque tiene su complejidad (por ejemplo, cuando uno olvida las claves y toca usar herramientas externas para gestionarlas).
Todas esas cosas que hacen parte del día a día del **desarrollo en TIC**, se las voy a enseñar más adelante.

Pero ojo: si ven que las clases están incompletas, es porque en este momento **solo les estoy pidiendo el módulo correspondiente al modelo dentro de la arquitectura**, ya que aún no sabemos con precisión cuál se aplicará en cada proyecto según su naturaleza.

Por ejemplo, proyectos pequeños —como el de “Mi Chucito”— pueden crecer rápidamente. Al principio parecen simples, pero con el aumento de usuarios o pedidos, la arquitectura **monolítica** deja de ser funcional, ya que no soporta tantas solicitudes.

En esos casos, se recomienda migrar hacia **arquitecturas por capas o por componentes**, que son más escalables y mantenibles.

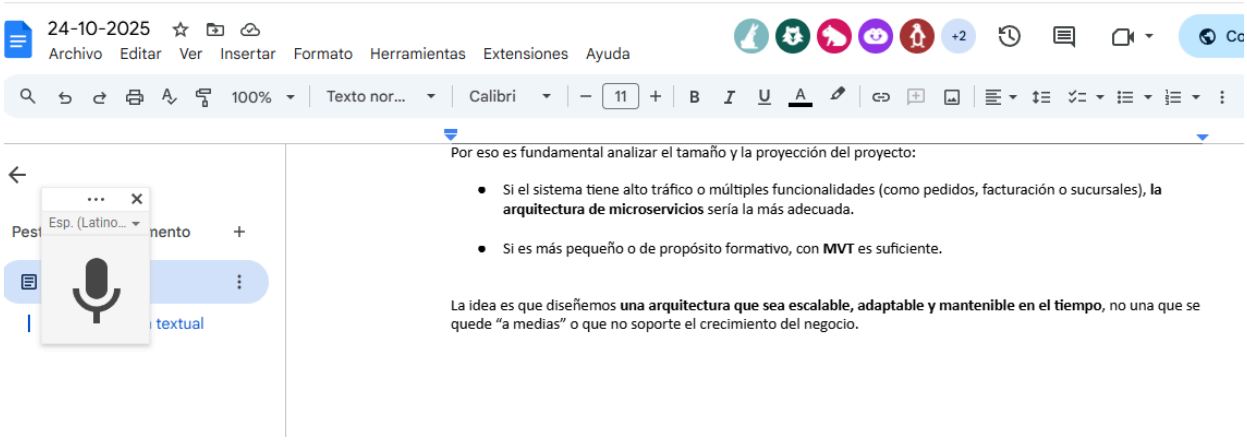
Por eso es fundamental analizar el tamaño y la proyección del proyecto:

- Si el sistema tiene alto tráfico o múltiples funcionalidades (como pedidos, facturación o sucursales), **la arquitectura de microservicios** sería la más adecuada.

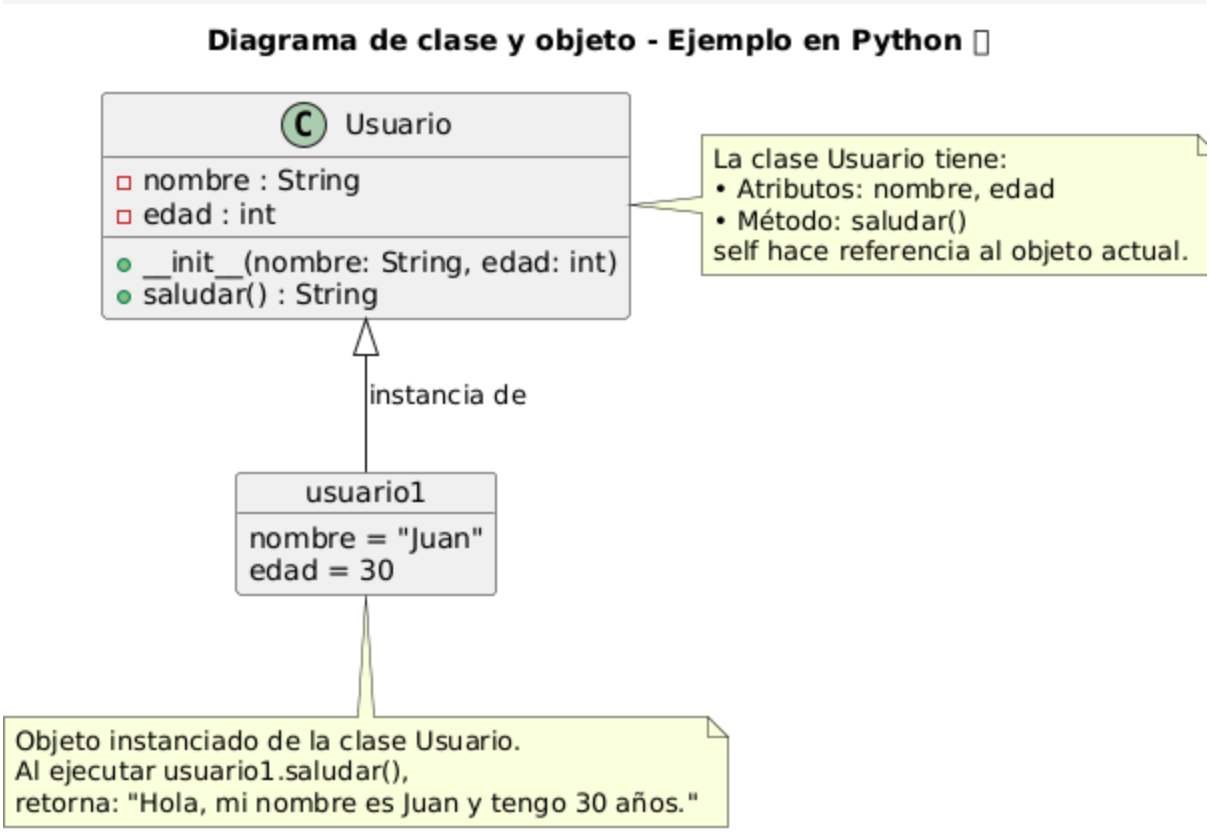
- Si es más pequeño o de propósito formativo, con **MVT** es suficiente.

La idea es que diseñemos **una arquitectura que sea escalable, adaptable y mantenible en el tiempo**, no una que se quede “a medias” o que no soporte el crecimiento del negocio.

evidencia de archivo compartido



https://www.youtube.com/watch?v=UocucJbOVno




```
#ejemplo de clase usuario con un objeto

class Usuario:

    def __init__(self, nombre, edad):# Constructor de la clase Usuario

        self.nombre = nombre# Atributo nombre

        self.edad = edad# Atributo edad



    def saludar(self):# Método para saludar

        return f"Hola, mi nombre es {self.nombre} y tengo {self.edad}
años."

# self es una referencia al objeto actual de la clase Usuario
```

```python
# objeto usuario1 de la clase Usuario

usuario1 = Usuario("Juan", 30)

print(usuario1.saludar())  # Llamada al método saludar del objeto usuario1
```