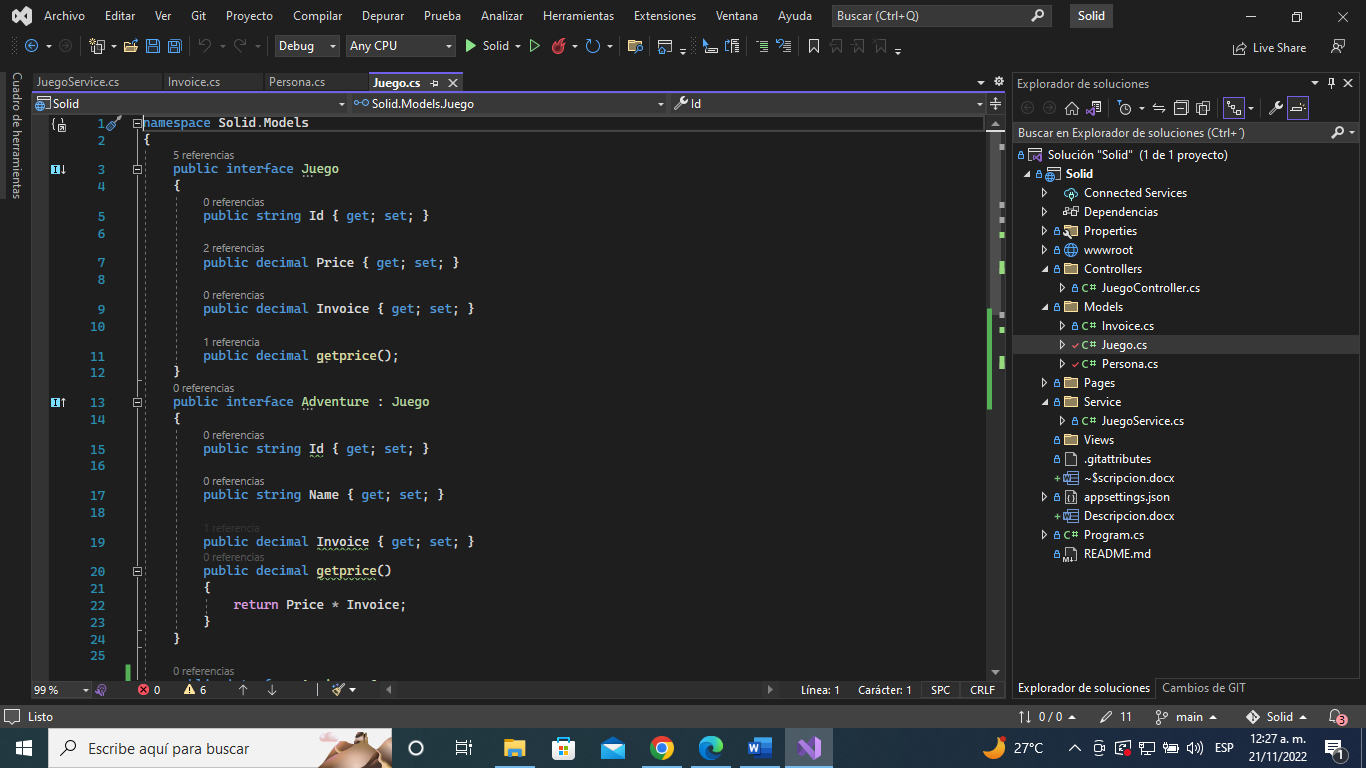
Hice un solo proyecto en el cual se implementaron los principios SOLID

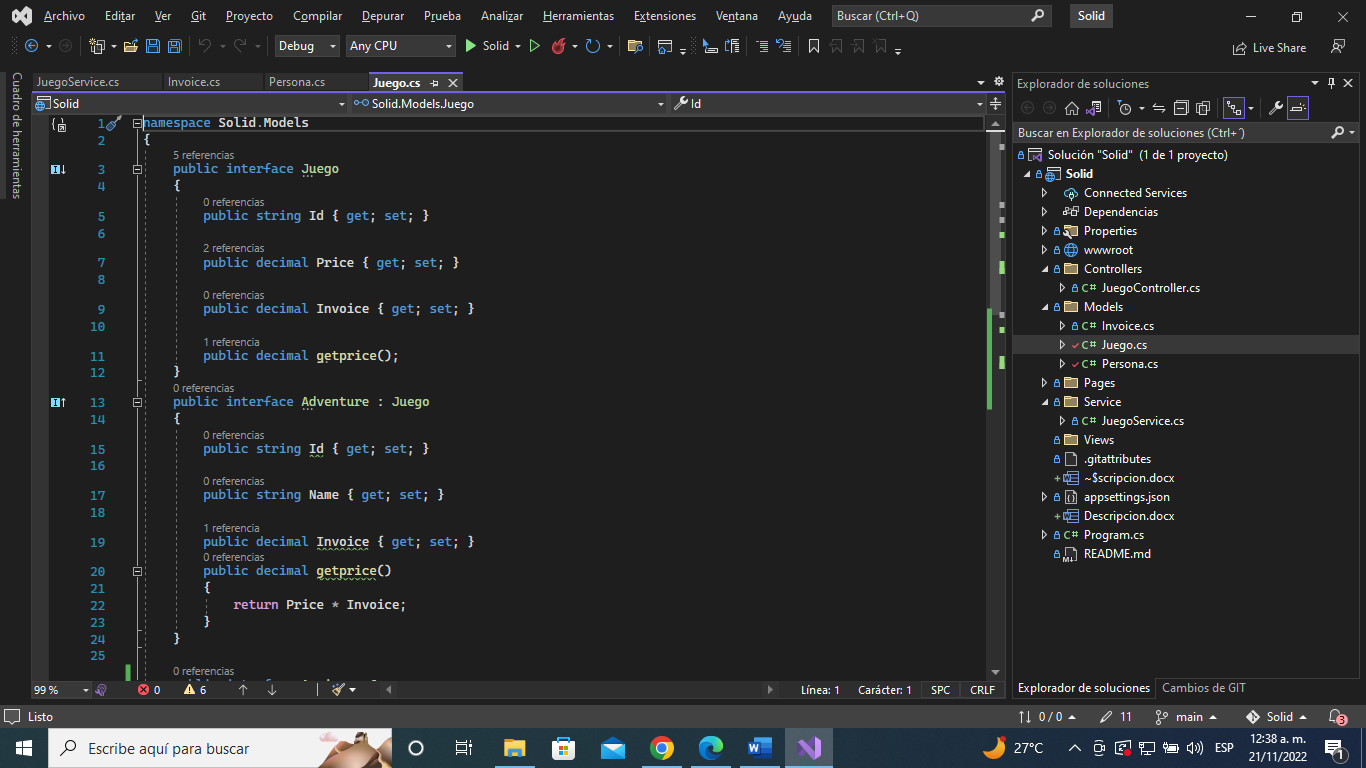
Para El Principio de Responsabilidad Única SRP nos dice que cada clase debe tener una sola responsabilidad y esto se debe a que, si se desea hacer un cambio en un futuro de los requerimientos o del funcionamiento solo se deba ir a un solo lugar, esto nos ayuda a la reutilización de código. A continuación, vemos como tengo dos clases, una que es el modelo de la clase y otra que se encarga del servicio de modificación en la base de datos.



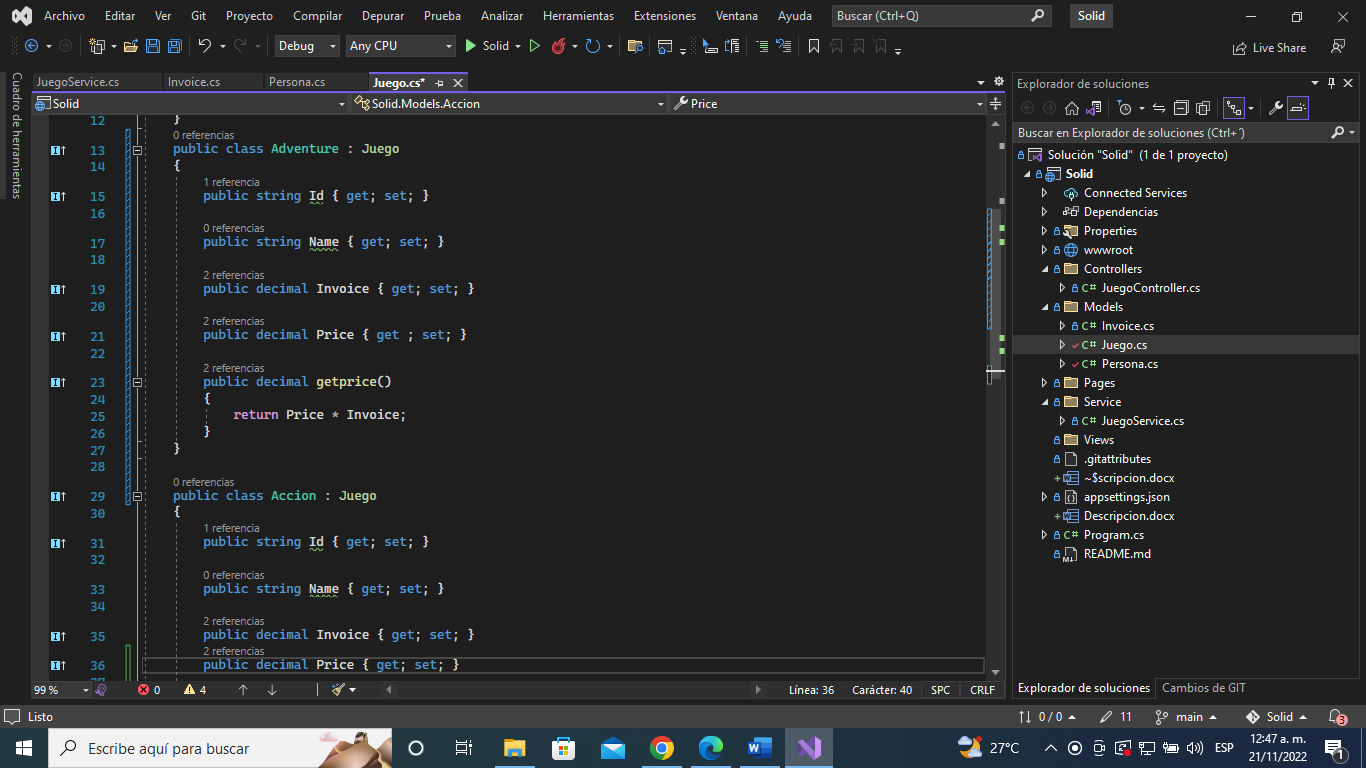
Captura de pantalla de computadora

Descripción generada automáticamente

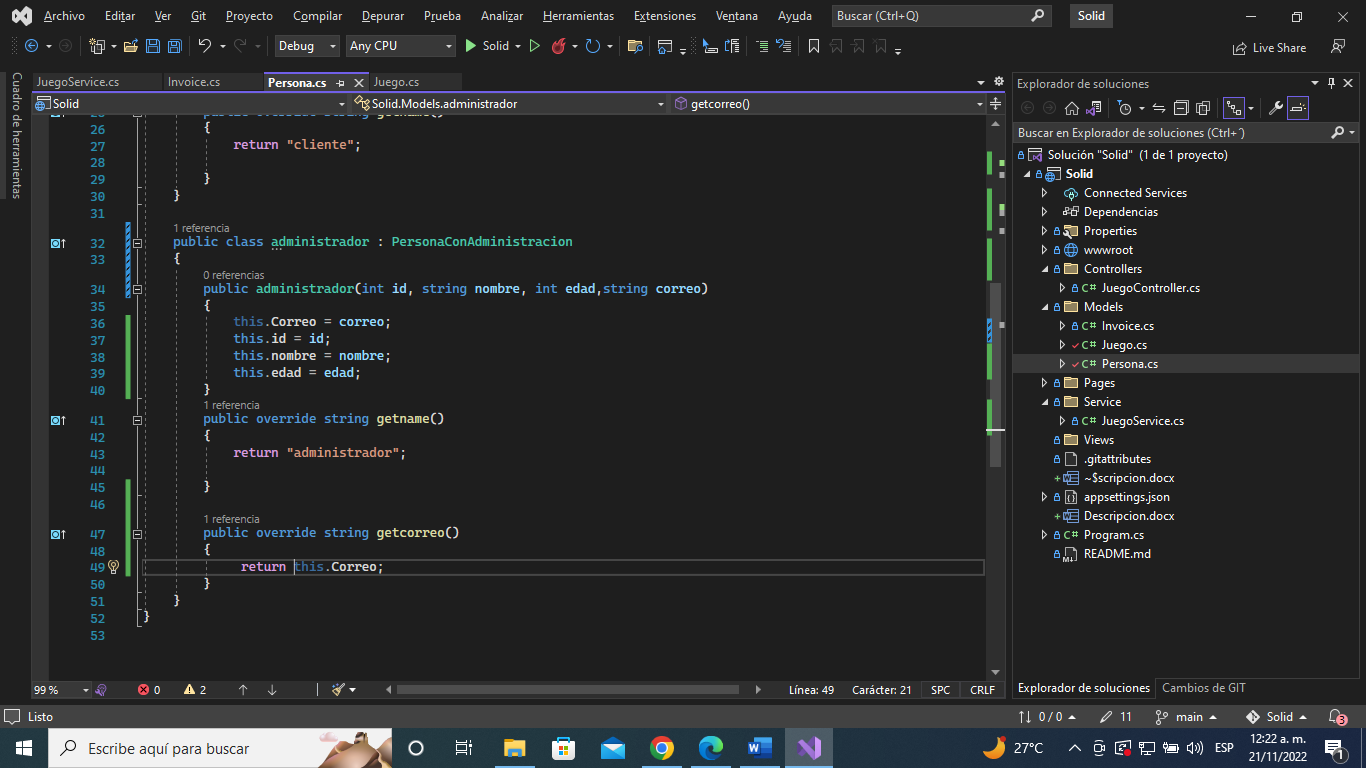
Para el Principio de Abierto/Cerrado establece que al crear una entidad de software esta debe quedar abierta para su extensión o cerrada para su modificación, ya que en teoría si se crea una clase no se debería volver a modificar. Cree un contrato (interface) que nos obliga a que cada vez que se crea una clase esta debe obedecer el contrato, esto hace que las clases tengan un contrato ligadas a ellas. A continuación se muestra como implemente este principio, lo que hice fue crear una interface que me obliga a que todo lo que se considere un juego debe cumplir con ciertas características:



Y luego cree 2 clases que siguen este contrato



Para el principio de Principio de Sustitución de Liskov LSP Cada clase hija de otra clase puede usarse como su padre, es decir cada clase puede sustituir al padre esto lo podemos ver en nuestro código cuando la clase PersonaAdmistracion sustituye a la clase Persona al crear un Administrador siendo que PersonaAdmistracion hereda de Persona.



Para el Principio de Segregación de Interfaces ISP para esta practica se nos recomienda no tener interfaces o contratos muy grandes ya tener muchos métodos puede ser contraproducente. Para implementar este principio podemos dividir una gran interfaz en varias que se comportan como interfaz A, Interfaz B o Interfaz C. Tampoco es recomendado hacer una interfaz por cada método, pero es buena practica saber cuándo separa las interfaces. Como vemos acontinuacion se crea otra interfaz para los videojuegos llamada juegosdatos:

Captura de pantalla de computadora

Descripción generada automáticamente

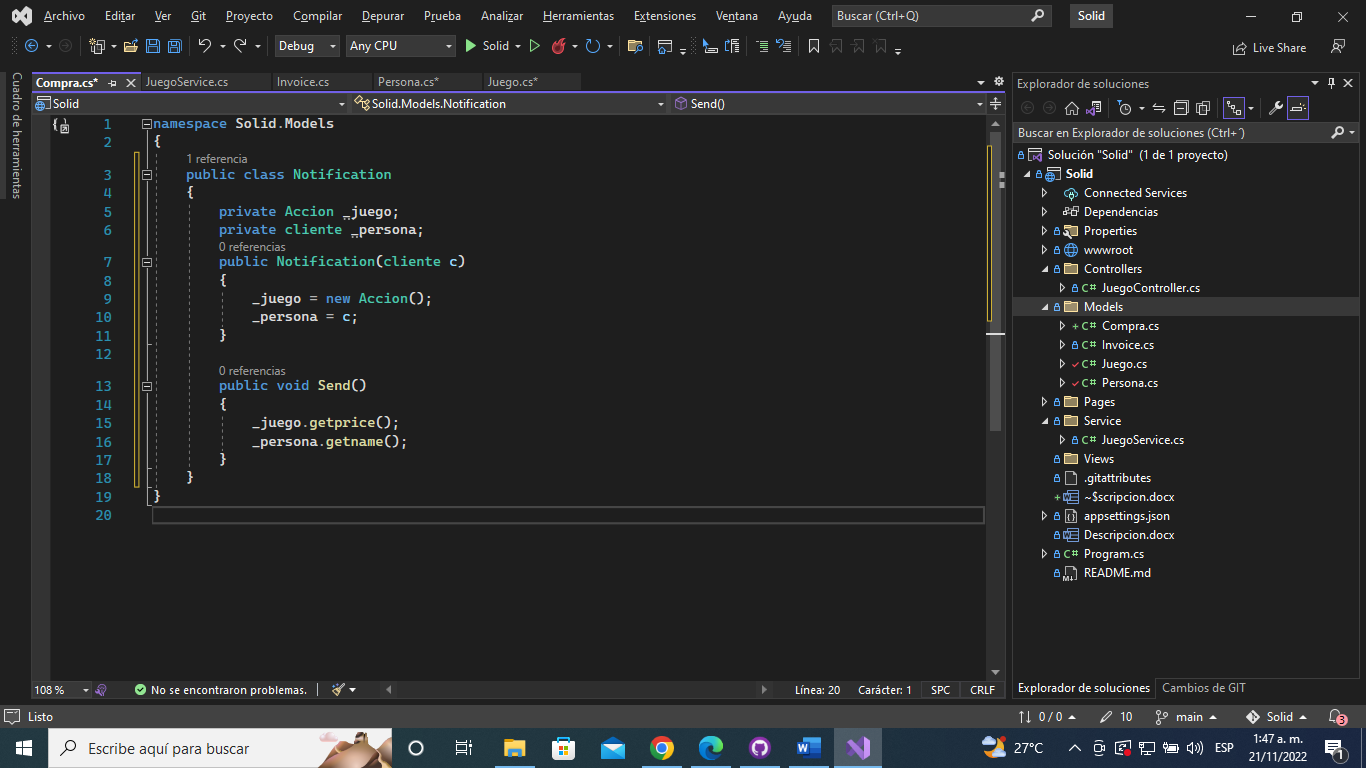
Y se utiliza en una clase para hacer que cumpla con el contrato:

Captura de pantalla de computadora

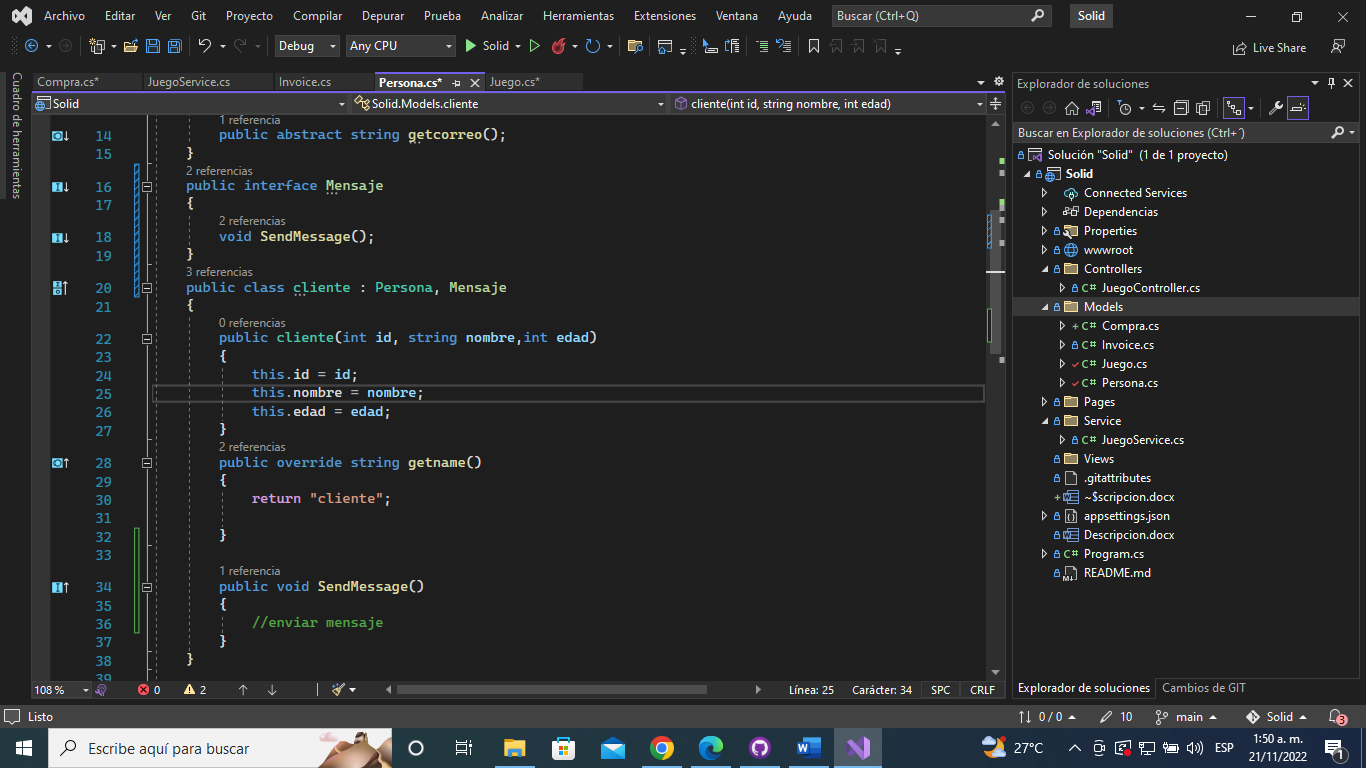
Descripción generada automáticamente

Y por ultimo para el Principio de Inversión de Dependencia DIP este principio nos dice que los módulos de alto nivel no deberían depender de los módulos de bajo nivel, las herramientas de bajo nivel son los módulos que te permiten hacer cosas como conexiones a base de datos, para poder hacer una conexión a un servicio o de encriptación, y el principio dije que estas

herramientas no deberían afectar si son cambiadas. A continuación, se ve como yo tenía una clase llamada compras que tenía dependencia en otras:



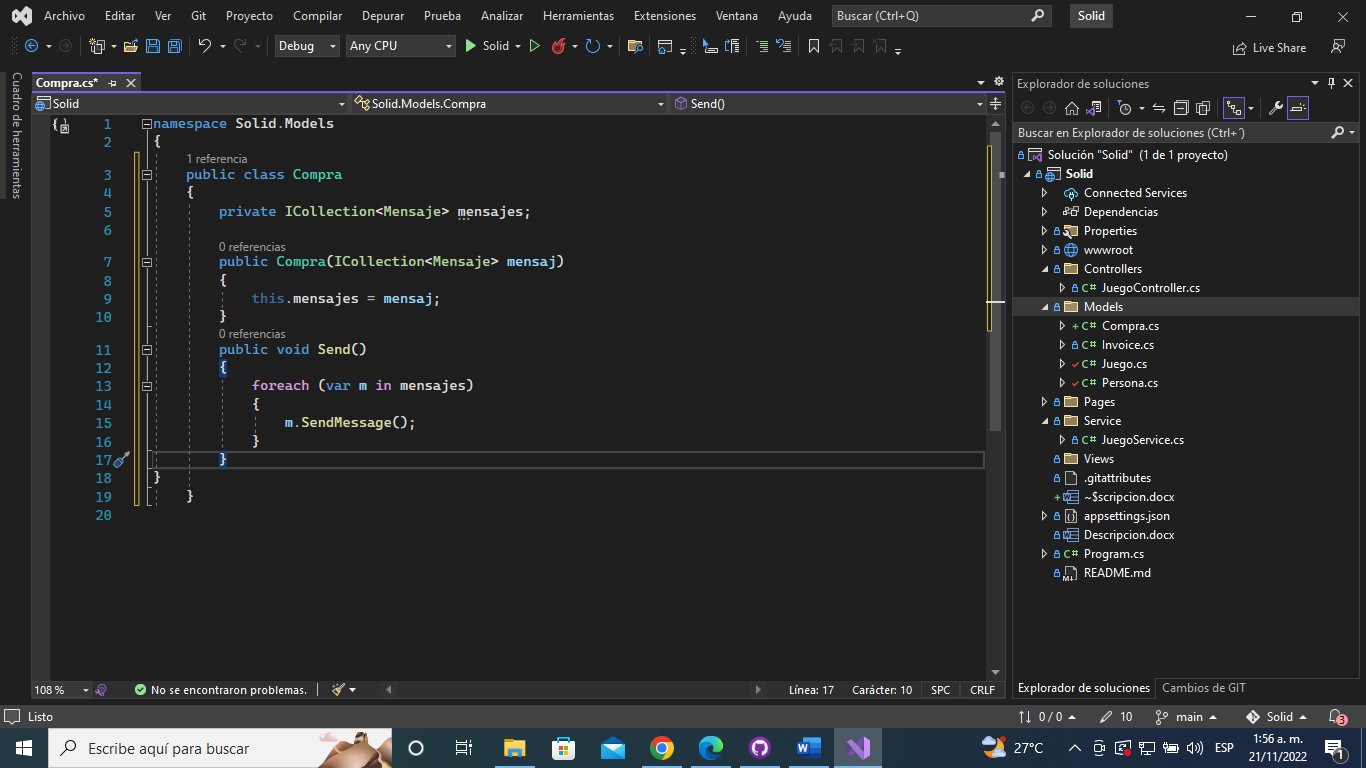
Lo que hice para ejecutar el principio fue crear una interfaz que me regresara lo que necesitara y agregarla a las clases de las cuales tenia dependencia que son cliente y acción:



Captura de pantalla de computadora

Descripción generada automáticamente

Después de esto modifique el código en la clase compras de forma que la clase compras ahora depende de una abstracción (IMessage) en vez de una clase.



En otras palabras, que los módulos de alto nivel (compras) y los de bajo nivel (acción y cliente) se basaran en abstracciones lo que cumple con el principio.