1. **En el método Crear de la clase JugadorService, ¿por qué se utiliza SCOPE_IDENTITY() en la consulta SQL y qué beneficio aporta al código?**

Scope_IDENTITY se utiliza debido a que como lo dice su nombre “apunta” a la identidad de lo que se acaba de agregar, en nuestro caso la identidad es el ID del jugador, y esta función nos permite devolver un valor escalar el cual nos permite hacer otras cosas, en este caso mostrar un mensaje más personalizado.

2. En el método Eliminar del servicio de jugadores, ¿por qué se verifica la existencia de elementos en el inventario antes de eliminar un jugador y qué problema está previniendo esta comprobación?

Verifica la existencia de elementos en el inventario, debido a que el jugador es una llave foránea en esa tabla, por lo que para poder eliminarse sin problemas, requiere que no hayan más datos en otras tablas que este relacionadas a ese ID

3. ¿Qué ventaja ofrece la línea using var connection=dbManager.GetConnection(); frente a crear y cerrar la conexión manualmente? Menciona un posible problema que podría ocurrir si no se usara esta estructura.

La ventaja que ofrece es que permite que se cierre automáticamente cuando se deje de utilizar, al no utilizar una estructura como esta puede que se olvide de cerrar la conexión haciendo que creen múltiples de estas si se vuelve a crear otra conexión, haciendo que se desperdicien recursos.

4. En la clase DatabaseManager, ¿por qué la variable _connectionString está marcada como readonly y qué implicaciones tendría para la seguridad si no tuviera este modificador?

Readonly lo que implica es que solo se puede asignar al momento de la creación de la variable o en el constructor, si no se tuviere este modificador, se podría cambiar la línea de conexión haciendo que ocurran errores

5. Si quisieras agregar un sistema de logros para los jugadores, ¿qué cambios realizarías en el modelo de datos actual y qué nuevos métodos deberías implementar en los servicios existentes?

Para agregar un sistema de logros, pensaría de una manera que no afecte mucho al código existente para no ocasionar problemas técnicos, principalmente en este tipo de sistema como el que realizamos, sería basado en el nivel del jugador, y los objetos que se puedan obtener, por ejemplo: un logro de llegar al nivel 50, o tener 5 stacks de hierro en el inventario. De los cambios con el enfoque que tendría sería el crear mínimo una tabla donde se tengan los logros, y este relacionada a la tabla de jugadores e inventario, con lo métodos debería hacer un sistema que detecte cuando se cumplen con los requisitos, agregar más logros, consultar la lista de logros etc.

6. ¿Qué sucede con la conexión a la base de datos cuando ocurre una excepción dentro de un bloque using como el que se utiliza en los métodos del JugadorService?

Se cierra automáticamente cuando ocurre una excepción

7. En el método ObtenerTodos() del JugadorService, ¿qué ocurre si la consulta SQL no devuelve ningún jugador? ¿Devuelve null o una lista vacía? ¿Por qué crees que se diseñó de esta manera?

Se devuelve una lista vacía, creo que se realizo de esta manera para evitar realizar errores en la lógica, ya que si se devuelve una lista vacía se puede suponer que simplemente no hay nada.

8. Si necesitaras implementar una funcionalidad para registrar el tiempo jugado por cada jugador, ¿qué cambios harías en la clase Jugador y cómo modificarías los métodos del servicio para mantener actualizada esta información?

Si tuviera que implementar una función que registra el tiempo jugado, simplemente modificaría la tabla de jugador, y haría que cada vez que se salga de un mundo o revise la configuración de tiempo guardo se actualice. Creo que lo guardaría en ms para guardar simplicidad, pero el usuario miraría una conversión dependiendo del tiempo jugado.

9. En el método TestConnection() de la clase DatabaseManager, ¿qué propósito cumple el bloque try-catch y por qué es importante devolver un valor booleano en lugar de simplemente lanzar la excepción?

Cumple con el propósito de probar la conexión y sí todo ocurre de manera correcta devuelve un true, y si falla un false. Es importante debido que al devolver un valor booleano, nos da una especie de personalización, podríamos probar otra dirección para la conexión o probar realizar otra cosa.

10. Si observas el patrón de diseño utilizado en este proyecto, ¿por qué crees que se separaron las clases en carpetas como Models, Services y Utils? ¿Qué ventajas ofrece esta estructura para el mantenimiento y evolución del proyecto?

Al principio sentí que era algo innecesario, pero después de tener la experiencia de trabajar en un proyecto como este, diría que se separaron para tener un mejor control y orden. Las ventajas que me parece que dan es el orden, si falla cierta parte del código ya se donde tengo que ir, si necesito crear una clase o modificarla ya se dónde están.

11. En la clase InventarioService, cuando se llama el método AgregarItem, ¿por qué es necesario usar una transacción SQL? ¿Qué problemas podría causar si no se implementara una transacción en este caso?

No se usan explícitamente, pero las transacciones SQL por lo que investigue es el tratar un conjunto de operaciones secuencialmente que se toman como una unidad, y si una de estas fallas, todas fallan. Considero que son importantes porque puede que se realice bien una operación, pero luego en otra ocurra un error y eso evitaría el que se comentan errores en la base de datos.

12. Observa el constructor de JugadorService: ¿Por qué recibe un DatabaseManager como parámetro en lugar de crearlo internamente? ¿Qué patrón de diseño se está aplicando y qué ventajas proporciona?

Por que permite practicar la reutilización del código, con solo una permite el transportar los datos necesarios a otro lado sin necesidad de volver a llamarlo. Por lo que investigue este tipo de patrón se llama Inyección de dependencias.

13. En el método ObtenerPorId de JugadorService, ¿qué ocurre cuando se busca un ID que no existe en la base de datos? ¿Cuál podría ser una forma alternativa de manejar esta situación?

Retorna un null, a mi parecer se debería crear una excepción que indique que pasa.

14. Si necesitas implementar un sistema de "amigos" donde los jugadores puedan conectarse entre sí, ¿cómo modificarías el modelo de datos y qué nuevos métodos agregarías a los servicios existentes?

A mi parecer debería crear otra tabla que pueda unir a la personas, y crear una función para esa tabla que realizaría la función de relacionar dos jugadores, y también permitiría que se conecten entre sí.

15. En la implementación actual del proyecto, ¿cómo se maneja la fecha de creación de un jugador? ¿Se establece desde el código o se delega esta responsabilidad a la base de datos? ¿Cuáles son las ventajas del enfoque utilizado?

Se le delega a la base de datos, me parece que es un buen enfoque, ya que permite que el registro del tiempo se genere automáticamente al crear un jugador, y esto lo realiza la base datos evitando errores, de lógica

16. ¿Por qué en el método GetConnection() de DatabaseManager se crea una nueva instancia de SqlConnection cada vez en lugar de reutilizar una conexión existente? ¿Qué implicaciones tendría para el rendimiento y la concurrencia?

Creo que se realiza de esta manera, ya que sería una mejor gestión de los recursos, ya que la conexión solo estaría presente en los momentos necesarios, en vez de estar todo el tiempo, aunque no se le necesite.

17. Cuando se actualiza un recurso en el inventario, ¿qué ocurriría si dos usuarios intentan modificar el mismo recurso simultáneamente? ¿Cómo podrías mejorar el código para manejar este escenario?

Dependería de lo que tarden de llegar las acciones al servidor, pero si fueran simultáneamente, dependiendo de la acción, digamos que sería tomar un objeto de un cofre, diría que pueden ocurrir dos situaciones la primera sería que ambos jugadores tendrían el objeto, o la segunda que al momento de tomar el objeto el sistema elija aleatoriamente quien se queda el objeto. A mi parece para evitar este tipo de errores, sería el que solo una persona puede interactuar con el objeto al mismo tiempo.

18. En el método Actualizar de JugadorService, ¿por qué es importante verificar el valor de rowsAffected después de ejecutar la consulta? ¿Qué información adicional proporciona al usuario?

Porque eso nos permitiría conocer si funciono la operación, si tira por lo menos que una línea fue afectada, ya sabemos que funciono correctamente, en cambio si tira que 0 o menos, nos da a conocer que no funciono

19. Si quisieras implementar un sistema de registro (logging) para seguir todas las operaciones realizadas en la base de datos, ¿dónde colocarías este código y cómo lo implementarías para afectar mínimamente la estructura actual?

Crearía otra utilidad que guarde cada acción realizada, simplemente colocando en el bloque de código que guarde que se cambio y cuando, esto lo haría al final del código evitando cambiar mucho el código prestablecido con anterioridad.

20. Observa cómo se maneja la relación entre jugadores e inventario en el proyecto. Si necesitaras agregar una nueva entidad "Mundo" donde cada jugador puede existir en múltiples mundos, ¿cómo modificarías el esquema de la base de datos y la estructura del código para implementar esta funcionalidad?

Se tendría que crear otra tabla y agregar un allave foránea a la tabla de inventario, indicado de que mundo es, y respecto al esquema de la base datos y la estructura, diría que cambiara casi todo, pero indicando que un parámetro más se tiene que guardar o modificar.

21. ¿Qué es un SqlConnection y cómo se usa?

Es un tipo de objeto, que permite usar diferentes funcionalidades para la conexión con la base datos, principalmente se utiliza para realizar la conexión y realizar ciertas comprobaciones.

22. ¿Para qué sirven los SqlParameter?

Los SqlParameter son los parámetros a pasar a la base datos, nos permiten interactuar con esta.