Cuestionario para responder:

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?

R// se utiliza para obtener el último valor insertado en un columna de identidad dentro del mismo ámbito de ejecución.

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?

R// se verifica el inventario para evitar eliminar jugadores que aún tiene ítems, prevenir errores y manteniendo la integridad de los datos

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 usará esta estructura.

R// Que el método using asegura el cierre automático al finalizar el bloque , si ocurre un error esto puede evitar fugas de conexión.

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?

R// está marcada como readonly para evitar que su valor pueda ser modificado en cualquier parte del programa después de su inicialización y evitar accesos indebidos o cambios accidentales.

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?

R// Creo dos clases: una para definir los logros y otra para asignarlos a jugadores. Agrego una lista de logros en la clase Jugador, creo el LogroService con métodos para crear, asignar y obtener logros, y agrego dos tablas nuevas en la base de datos

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?

R// La conexión se cierra automáticamente al salir del bloque, incluso si ocurre una excepción.

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

R// si la consulta de Sql no devuelve a ningún jugador, el método devuelve una lista vacía, no nula

8) Si necesitas 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?

R// En la clase `Jugador` agregaría una nueva propiedad para guardar el tiempo jugado. También agregaría una columna en la tabla `Jugadores` para registrar ese dato en sql server. Luego, modificaría los métodos del jugadorservice para que reciban y actualicen el tiempo cuando sea necesario

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?

R// Uso el try- catch evita que el programa se detenga si ocurre un error al conectar. Devolver un valor booleano permite saber si la conexión fue exitosa o falló, sin que el programa se cierre por la excepción.

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?

R// Ayuda a organizar mejor el proyecto y facilita mantenimiento, pruebas y crecimiento.

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?

R// La transacción SQL en agregaritem asegura que todas las acciones necesarias para añadir un ítem se realicen juntas o no se hagan en absoluto. Así evitar errores y se asegura de que el inventario siempre quede correcto y sin datos incompletos.

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?

R// se recibe el DataBaseManager en el constructor para aplicar el patrón de inversión de Dependencia. Estos permite que el servidor no dependa de cómo se crea la conexión a la base de datos,

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?

R// devuelve un Null si no encuentra un id que no se encuentre en la base de datos

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?

R// 1. Crear la tabla Amigos en la base de datos.
2.Agregar claves foráneas JugadorId y AmigoId que apunten a la tabla Jugadores.
Nuevos métodos en JugadorService:

- EnviarSolicitudAmistad
- AceptarSolicitudAmistad
- RechazarSolicitudAmistad
- ObtenerAmigos
- EliminarAmigo

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?

R// Se delega a la base de datos, lo que asegura precisión con la hora del servidor, simplifica el código y registra la fecha al momento exacto de la inserción

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?

R// Se crea una nueva conexión para evitar errores por uso compartido y asegurar estabilidad. Utilizarla puede causar conflictos.

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?

R// Le agregaría un nuevo campo llamado versión y modificó la consulta que está en la clase actualizar inventario, le agrego que version = version + 1 para que cuando alguien actualiza un recurso la versión se aumente a 1 y si alguien mas actualiza ese mismo recurso entonces version seria = 2 y por lo tanto no se podría actualizar hasta recargar base de dados

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?

R// El rowsAffected me ayuda a verificar si el jugador fue encontrado mostrándome un mensaje de éxito si este es >0, sino significa que el jugador con el Id proporcionado no fue encontrado, mostrando un mensaje de error.

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?

R//Creo una clase aparte para el logging y solo la llamo desde los métodos que usan la base de datos. Así registro las operaciones sin cambiar mucho el resto del proyecto

20) Observa cómo se maneja la relación entre jugadores e inventario en el proyecto. Si necesitas 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?

R// Para manejar jugadores en varios mundos con inventarios separados, primero creó una tabla Mundos y otra llamada JugadorMundos que conecta jugadores con mundos. Luego modificó la tabla Inventarios para agregar el id_mundo. En el código, creó el modelo Mundo y actualizo los modelos Jugador e Inventario para incluir el id_mundo.

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

R// clase que establece la conexión entre el programa y la base de datos sql server
using (SqlConnection connection = new SqlConnectio(connetionString))

22) ¿Para qué sirven los SqlParameter?

R// Para evitar inyecciones SqL y mejorar la seguridad.