UNIVERDAD MARIANO GALVEZ DE GUATEMALA
FACULTAD DE INGENIERIA EN SISTEMAS DE
INFORMACION Y CIENCIAS DE LA COMPUTACION
CENTRO UNIVERSITARIO CAMPUS JUTIAPA

FACULTAD:

INGENIERIA EN SISTEMAS

CURSO:

PROCESO ADMINISTRATIVO

CATEDRATICO:

LIC. JUAN PABLO GONZALEZ LEON

ALUMNO:

LEBINSON DAVID GARCIA CASTILLO

CARNE:

0905-24-13926

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 para obtener el último ID generado en el mismo ámbito de la transacción SQL. Esto es útil cuando acabas de insertar un registro y necesitas trabajar con su ID, como para asociarlo con otros datos o devolverlo al usuario. Sin esta función, podrías terminar obteniendo un ID incorrecto si hay múltiples operaciones concurrentes en la base de datos. Es como pedirle al mesero el número de tu mesa después de hacer un pedido, para que no haya confusiones y te sirvan en la mesa correcta.

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?

Antes de eliminar un jugador, se verifica si tiene ítems en el inventario para evitar dejar datos "huérfanos" en la base de datos. Esto asegura que no queden registros en el inventario que apunten a un jugador que ya no existe, lo que podría causar errores o inconsistencias en el sistema. Es como asegurarte de que alguien no deje su mochila llena de cosas en tu casa antes de mudarse. ¡No queremos líos después!

3. ¿Qué ventaja ofrece la línea using var connection = _dbManager.GetConnection(); frente a crear y cerrar la conexión manualmente?

Usar using garantiza que la conexión se cierre automáticamente al salir del bloque, incluso si ocurre una excepción. Esto evita fugas de recursos y asegura que las conexiones no queden abiertas innecesariamente, lo que podría saturar el servidor de base de datos. Es como tener un robot que apaga las luces cuando sales de casa. Si no lo haces, podrías quedarte sin energía (o recursos del sistema), y nadie quiere eso.

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?

El modificador readonly asegura que el valor de _connectionString no pueda ser modificado después de ser inicializado, lo que mejora la seguridad y evita cambios accidentales. Si no tuviera este modificador, alguien podría cambiar la cadena de conexión en tiempo de ejecución, lo que podría comprometer la seguridad de la base de datos o causar errores inesperados. Es como grabar algo en piedra: nadie puede cambiarlo, ni siquiera tú, y eso da tranquilidad.

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 implementar un sistema de logros, añadiría una nueva tabla llamada Logros que contenga información como el nombre del logro, la descripción y su relación con el JugadorId. También crearía una tabla intermedia si los logros pueden ser compartidos entre jugadores. En los servicios, implementaría métodos para asignar logros a jugadores, consultar los logros obtenidos y verificar si un jugador cumple con los requisitos para desbloquear un logro. Es como crear una lista de medallas para los jugadores: cada vez que hacen algo épico, les damos una para motivarlos.

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?

Cuando ocurre una excepción dentro de un bloque using, la conexión se cierra automáticamente gracias a la implementación de IDisposable. Esto asegura que los recursos se liberen correctamente, incluso si algo sale mal. Es como si tuvieras un seguro que limpia el desastre aunque algo salga mal. ¡Tranquilidad total para ti y para el servidor de base de datos!

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?

Si la consulta SQL no devuelve ningún jugador, el método retorna una lista vacía. Esto evita errores al iterar sobre el resultado, ya que trabajar con null podría causar excepciones. Se diseñó de esta manera para simplificar el manejo de datos en el código y evitar la necesidad de comprobaciones adicionales. Es como abrir la nevera y no encontrar comida: no es lo ideal, pero al menos no te llevas un susto porque la nevera esté rota.

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?

Añadiría un campo TiempoJugado en la tabla Jugadores para almacenar el tiempo total jugado. En los métodos del servicio, implementaría lógica para actualizar este campo cada vez que un jugador inicie o termine una sesión. También podría agregar un método para consultar el tiempo jugado por jugador. Es como llevar un cronómetro para cada jugador: así sabemos quién es el más dedicado (o el más viciado).

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?

El bloque try-catch maneja errores de conexión sin interrumpir el flujo del programa. Devolver un valor booleano permite al código saber si la conexión fue exitosa o no, sin

necesidad de manejar excepciones en cada llamada. Es como preguntar "¿Todo bien?" antes de entrar a una fiesta. Si no, mejor ni entras y evitas el drama.

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?

Separar las clases en carpetas como Models, Services y Utils mejora la organización del proyecto, facilita el mantenimiento y sigue el principio de responsabilidad única. Esto hace que sea más fácil encontrar y modificar el código relacionado con una funcionalidad específica. Es como tener cajones separados para calcetines, camisetas y pantalones: todo está en su lugar y es más fácil encontrar lo que necesitas.

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?

Una transacción SQL asegura que todas las operaciones relacionadas se completen o se deshagan juntas. Sin una transacción, podrías terminar con datos inconsistentes si ocurre un error en medio del proceso. Es como pedir un combo en un restaurante: si no te dan todo, no pagas. ¡Todo o nada!

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?

Recibir un DatabaseManager como parámetro sigue el patrón de inyección de dependencias, lo que facilita pruebas unitarias y mejora la flexibilidad del código. Esto permite cambiar la implementación del DatabaseManager sin modificar el servicio. Es como pedirle a alguien que te pase la sal en lugar de ir tú mismo a buscarla. ¡Eficiencia pura y sin distracciones!

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?

Cuando se busca un ID que no existe, el método devuelve null. Esto indica que no se encontró el registro. Una forma alternativa de manejar esta situación sería lanzar una excepción personalizada o devolver un objeto con un estado de "no encontrado". Es como buscar un libro en la biblioteca y que te digan: "Ese no lo tenemos, amigo".

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?

Para implementar un sistema de "amigos", crearía una tabla Amigos con relaciones JugadorId1 y JugadorId2. También añadiría métodos en JugadorService para gestionar estas relaciones, como agregar, eliminar y listar amigos. Es como una red social para jugadores. ¡Conexión total!

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?

Generalmente, se delega a la base de datos con un valor predeterminado en la columna. Esto asegura consistencia y precisión, ya que la base de datos siempre tiene la hora exacta. Es como dejar que el reloj de la base de datos marque la hora. ¡Siempre puntual y sin margen de error!

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?

Crear una nueva instancia evita problemas de concurrencia y asegura que cada operación tenga su propia conexión. Reutilizar conexiones podría causar bloqueos o errores inesperados si varias operaciones intentan usar la misma conexión al mismo tiempo. Es como usar un vaso limpio cada vez que tomas agua. ¡Nada de mezclar sabores!

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?

Si dos usuarios intentan modificar el mismo recurso simultáneamente, podrían ocurrir conflictos de concurrencia, como sobrescribir los cambios del otro. Para manejar esto, implementaría bloqueos en la base de datos o un control de versiones en los registros. Es como dos personas intentando usar el mismo carrito en el supermercado. ¡Caos total si no se organiza bien!

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?

Verificar rowsAffected permite saber si la operación tuvo éxito o si no se encontró el registro a actualizar. Esto proporciona retroalimentación al usuario y evita que piense que la operación fue exitosa cuando no lo fue. Es como preguntar: "¿Seguro que entregaste el paquete?" antes de irte.

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?

Colocaría el código de logging en los métodos de los servicios, justo antes y después de las operaciones críticas. Usaría una biblioteca como Serilog para registrar las operaciones en un archivo o base de datos. Esto centraliza el registro sin afectar la lógica principal. Es como llevar un diario de todo lo que pasa. ¡Nada se escapa!

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?

Crearía una tabla Mundos para almacenar información sobre cada mundo y una tabla intermedia JugadorMundo para gestionar la relación muchos a muchos entre jugadores y mundos. También añadiría métodos en los servicios para gestionar esta nueva entidad. Es como darle a cada jugador su propio universo. ¡Multiverso desbloqueado!

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

Un SqlConnection es una clase que representa una conexión a una base de datos SQL Server. Se usa para ejecutar consultas y comandos. Es como el puente que conecta tu código con la base de datos. ¡Sin él, no hay tráfico de datos!

22. ¿Para qué sirven los SqlParameter?

Los SqlParameter se usan para evitar inyecciones SQL al pasar parámetros de forma segura a las consultas. También mejoran la legibilidad y el mantenimiento del código. Es como usar un candado en tu mochila. ¡Nada de que alguien meta cosas raras!