Universidad Mariano Gálvez Guatemala, Campus Jutiapa

Facultad de Ingeniería en Sistemas

Curso: Programación 1

Catedrático: Ing. Ruldin Ayala

Nombre del estudiante: Lester David Payes Méndez

Carné: 0905-24.22750

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 consulta SQL. Esto asegura que el ID devuelto sea el correspondiente al registro recién creado, incluso en entornos con múltiples usuarios o transacciones concurrentes. El beneficio es que permite asignar correctamente el ID generado al objeto Jugador en el código.

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?**
   Se verifica la existencia de elementos en el inventario para evitar inconsistencias en la base de datos. Si se eliminara un jugador con elementos en el inventario, se generarían referencias huérfanas en la tabla Inventario, lo que violaría la integridad referencial.

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 es que using garantiza que la conexión se cierre automáticamente al salir del bloque, incluso si ocurre una excepción. Si no se usara esta estructura, podrías olvidar cerrar la conexión manualmente, lo que podría agotar el límite de conexiones disponibles en el servidor y afectar el rendimiento.

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?**
   _connectionString está marcada como readonly para evitar que sea modificada después de su inicialización, lo que protege la configuración de la base de datos contra cambios accidentales o malintencionados. Si no tuviera este modificador, un error o un ataque podría alterar la cadena de conexión, comprometiendo la seguridad.

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?**
   Agregaría una tabla Logros con columnas como Id, Nombre y Descripcion, y una tabla intermedia JugadorLogros para relacionar jugadores con logros. En los servicios, implementaría métodos como AsignarLogro(int jugadorId, int logroId) para asignar logros y ObtenerLogros(int jugadorId) para obtener los logros de un jugador.

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?**
   La conexión se cierra automáticamente al salir del bloque using, incluso si ocurre una excepción. Esto es posible porque SqlConnection implementa la interfaz IDisposable, que asegura la liberación de recursos.

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?**
   Devuelve una lista vacía porque se inicializa una lista antes de ejecutar la consulta (var jugadores = new List<Jugador>();). Esto se diseñó así para evitar errores al iterar sobre el resultado y permitir que el código consumidor trabaje sin necesidad de verificar si el valor es null.

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?**
   Agregaría una propiedad TiempoJugado en la clase Jugador y modificaría el método Actualizar para incluir la actualización de este campo. También implementaría un método IncrementarTiempoJugado(int jugadorId, TimeSpan tiempo) para registrar el 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?**
   El bloque try-catch captura errores de conexión y evita que la aplicación se detenga abruptamente. Devolver un valor booleano permite al código consumidor manejar el resultado de la prueba de conexión de manera más flexible, como mostrar un mensaje al usuario o intentar reconectar.

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?**
    Las clases se separaron en carpetas para aplicar el principio de responsabilidad única, donde cada carpeta agrupa clases con responsabilidades similares. Esto mejora la organización, facilita la navegación y permite que el proyecto sea más mantenible y escalable.

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 (como insertar un ítem y actualizar el stock) se completen o se deshagan juntas. Sin una transacción, un error en una operación podría dejar la base de datos en un estado inconsistente.

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?**
Se aplica el patrón de inyección de dependencias, lo que desacopla la lógica del servicio de la gestión de la base de datos. Esto facilita las pruebas unitarias al permitir el uso de un DatabaseManager simulado y mejora la flexibilidad del código.

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?**
Devuelve null si no se encuentra el jugador. Una alternativa sería lanzar una excepción personalizada como JugadorNoEncontradoException para informar explícitamente que el jugador no existe.

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?**
Agregaría una tabla Amigos con columnas JugadorId y AmigoId para representar relaciones bidireccionales. En los servicios, implementaría métodos como AgregarAmigo(int jugadorId, int amigoId) y ObtenerAmigos(int jugadorId).

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?**
La fecha de creación se obtiene desde la base de datos (FechaCreacion en la consulta SQL). Esto asegura consistencia y precisión al usar la hora del servidor, evitando discrepancias entre clientes.

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?**
Se crea una nueva instancia para evitar conflictos en entornos concurrentes. Reutilizar conexiones podría causar bloqueos o errores si múltiples operaciones intentan usar la misma conexión simultáneamente.

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?**
Podría ocurrir una condición de carrera, donde los cambios de un usuario sobrescriban los del otro. Para manejar esto, se podría implementar control de concurrencia optimista (usando un campo Version) o bloqueos en la base de datos.

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 confirma que la operación afectó algún registro. Si es 0, significa que el ID no existe, lo que permite informar al usuario que no se encontró el jugador.

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 el DatabaseManager para capturar todas las consultas ejecutadas. Usaría una biblioteca como Serilog para registrar las operaciones en un archivo o base de datos.

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 y una tabla intermedia JugadorMundos para manejar la relación muchos a muchos. En el código, implementaría métodos como AsignarMundo(int jugadorId, int mundoId) y ObtenerMundos(int jugadorId).

21. **¿Qué es un SqlConnection y cómo se usa?**
Es una clase que representa una conexión a una base de datos SQL Server. Se usa para abrir, ejecutar comandos y cerrar conexiones con la base de datos.

22. **¿Para qué sirven los SqlParameter?**
Sirven para pasar valores a las consultas SQL de forma segura, evitando inyecciones SQL y asegurando que los datos se procesen correctamente.