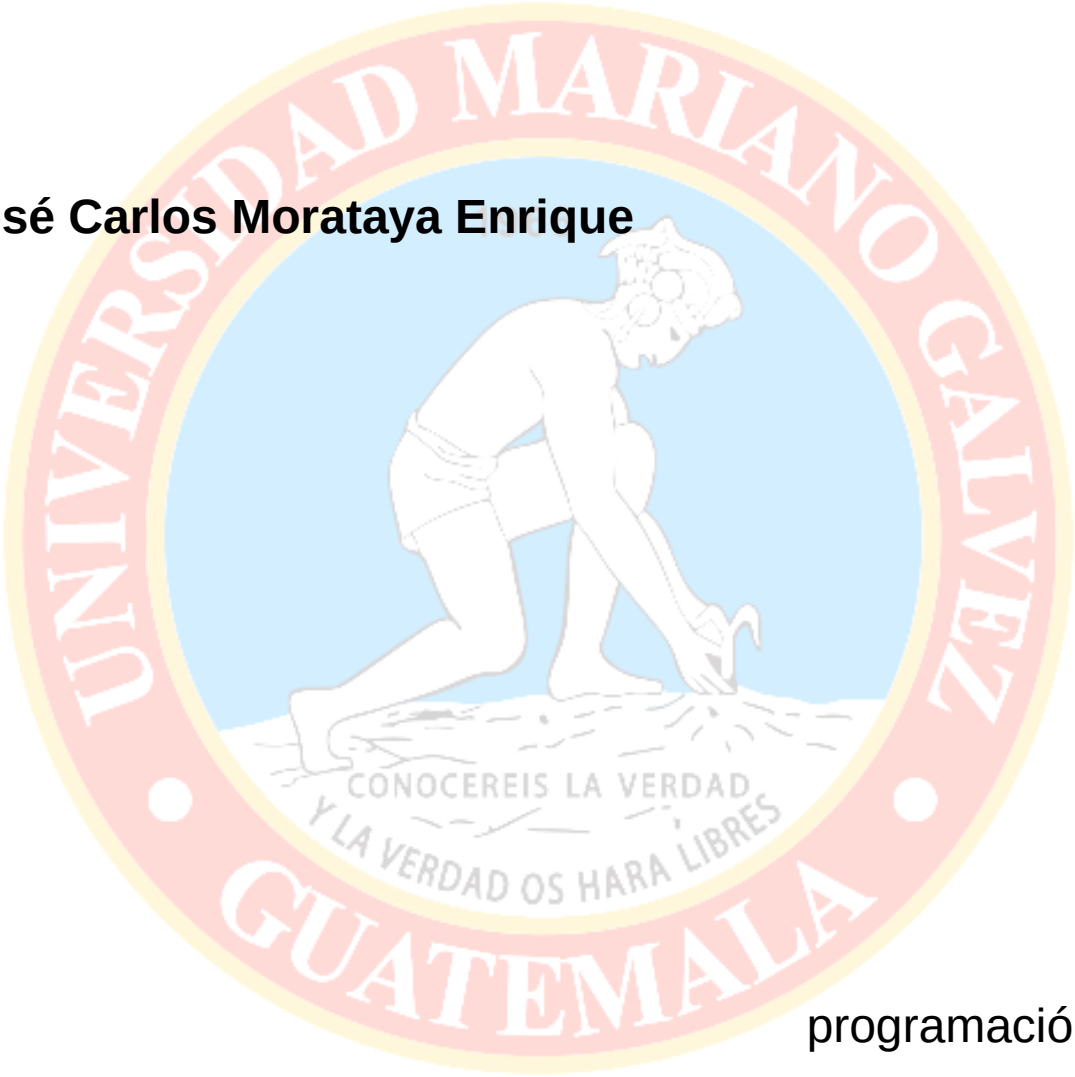
Ingeniería en Sistemas y Ciencias de la Computación

Tarea

Estudiante:

José Carlos Morataya Enrique



programación I

Jutiapa Abril 2025

Cuestionario a 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?

Genera la ID automáticamente evitando que el usuario la escriba

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?

Como estábamos trabajando con bases de datos para conectar tablas lo hacemos a través de llaves foráneas y primarios y al eliminar un jugador que esta conectado a la tabla de inventario podría ocasionar un error, entonces por eso verifica que el jugador no tenga cosas en el inventario para eliminarlo

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.

Al utilizar using estamos previniendo incluso si ocurre una excepción, donde al momento de usar connecion.open también se estará cerrando

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?

Al aplicar el readonly si esta haciendo en el constructor y no lo estamos instanciando como lo vimos en las anteriores clases esto ayuda a que no se puede modificar

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?

Primero si queremos almacenar los logros por jugador debemos de implementar una tabla de logros en la base da datos para almacenar y conectarlos con los jugadores, luego crearía una void que inserte en la base de datos el logro del jugador que cumplió los requisitos

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 la conexión con la 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?

Se devuelve una lista vacia, me imagino por la simplicidad y que no se muestre ninguna excepción en la interfaz inesperada

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?

Podría implementar una lógica con una void para que el jugador ingrese el tiempo jugado en Jugadorservice y tendría que crear una nueva columna en en tb jugadores para almacenar ese tiempo jugado por jugador

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?

Para mas simplicidad y no lanzar la extensión el objetivo es saber si la conexión esta funcionando

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?

Ordenar las clases y saber donde esta cada cosa para mayor accesibilidad a si como yo al momento de agregar los Resource lo hice a través de carpetas para llevar un orden

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?

Patrón utilizado: Singleton.

Por qué es adecuado:Consistencia: Una sola instancia asegura que todas las conexiones usen el mismo string de conexión.

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?

Usar SqlTransaction con connection.BeginTransaction().

Ejecutar DELETE FROM Inventario y DELETE FROM Jugadores dentro de la transacción.

Llamar Commit() si ambas operaciones son exitosas, o Rollback() si hay un error.

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?

Garantiza que solo se inserten ítems con cantidades válidas (> 0), respetando la lógica

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?

Nueva tabla Tb_amigos con columnas IdJugador1, IdJugador2, FechaAmistad, y claves foráneas a Tb_jugadores.

AgregarAmigo: Inserta una nueva relación de amistad.

ObtenerAmigos: Lista los amigos de un jugador usando un JOIN con UNION para relaciones bidireccionales.

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?

En los models del jugador aparece ToShortDateString que es para que aparezca la fecha al momento de ingresarla

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?

El connection pool de ADO.NET reutiliza conexiones físicas, haciendo eficiente la creación de nuevas instancias.

Garantiza un estado limpio y evita conflictos en operaciones concurrentes, ya que SqlConnection no es thread-safe

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?

Sin manejo de concurrencia, las actualizaciones simultáneas pueden causar condiciones de carrera, sobrescribiendo cambios y dejando datos inconsistentes (por ejemplo, cantidades incorrectas).

Usaria un bloqueo optimista con una columna Version para detectar conflictos.

Aplicar transacciones para asegurar atomicidad.

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 asegura que la actualización afectó al menos una fila, confirmando que el jugador existía.Evita errores lógicos donde el método parece exitoso pero no hizo nada.

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?

Crear una clase Logger en Utils. Inyectar Logger en JugadorService y InventarioService vía constructor.

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?

Nueva tabla Tb_mundos para almacenar mundos.

Nueva tabla Tb_jugador_mundo para relacionar jugadores y mundos.

Agregar MundoId a Inventario para asociar ítems a un mundo específico.

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

Es un extensión de C# para conectar la base de datos de SQL, al usar el SqlConnection estamos entrando a la base de datos

22. ¿Para qué sirven los SqlParameter?

Prevenir inyección SQL: Asegura que los valores se traten como datos