Programación Web 3

UNLaM - Tecnicatura en Desarrollo Web

Trabajo Práctico de Investigación

Título: *Entity Framework en Base de datos NoSQL*

Integrantes

1. Gastón Santos.
2. Erica Herrera.
3. Leandro Martínez.
4. Nicolas Kuhn.
5. Tabatha Peralta.

Objetivo

El objetivo de este trabajo practico consiste en investigar y comprender como se puede consumir una base de datos NoSQL utilizando Entity Framework, como conectarse a la API de Azure CosmosDB para cuenta de MongoDB haciendo uso de la librería MongoFramework.

Se buscará comprender como se puede lograr la integración de Entity Framework con bases de datos NoSQL y las diferencias que existen con respecto a una Base de tipo relacional, cómo funciona la librería elegida y su uso en un caso de ejemplo.

Situación Actual

Se ha desarrollado una aplicación Web utilizando .NET 6, la API Azure CosmosDB para MongoDB y la librería MongoFramework para el mapeo de las colecciones de datos a código.

La aplicación ofrece una variedad de características, incluyendo una vista que muestra las propiedades disponibles para la venta, una sección de favoritos donde los usuarios pueden guardar sus propiedades preferidas, una sección de recomendados que sugiere propiedades basadas en las preferencias del usuario, una

vista detallada de cada propiedad la cual nos da información adicional y el contacto del Agente correspondiente a esa propiedad y una opción para agregar nuevas propiedades. También existe la opción de búsqueda con ciertas características que hacen que puedas encontrar lo que buscabas de forma rápida y precisa. Se puede coordinar una fecha de visita a la casa seleccionada.

Desarrollo de la Investigación

MongoDB

MongoDB es una base de datos orientada a documentos de código abierto y sin esquemas. En lugar de almacenar los datos en tablas, como lo hacen las bases de datos relacionales tradicionales, MongoDB almacena los datos en documentos BSON (Binary JSON), que es una representación binaria de documentos en formato JSON (JavaScript Object Notation).
JSON es un formato de intercambio de datos ligero y legible por humanos. Se basa en la sintaxis de objetos de JavaScript y se utiliza ampliamente en aplicaciones web para transmitir datos entre el servidor y el cliente. MongoDB utiliza el formato JSON como su formato de almacenamiento principal debido a varias razones:

Flexibilidad

JSON es un formato muy flexible y permite representar datos estructurados y anidados. No tiene un esquema estricto, lo que significa que los documentos en una colección de MongoDB pueden tener estructuras diferentes y agregar nuevos campos sin requerir cambios en la estructura existente. Esto facilita la adaptación de los datos a medida que evoluciona la aplicación.

Compatibilidad con lenguajes de programación

JSON es compatible con una amplia variedad de lenguajes de programación. La mayoría de los lenguajes modernos tienen soporte nativo para la serialización y deserialización de datos en formato JSON. Al utilizar JSON como formato de almacenamiento, MongoDB puede integrarse fácilmente con diferentes lenguajes y plataformas.

Integración con JavaScript

MongoDB está escrito en C++ pero tiene una interfaz de cliente en JavaScript que permite interactuar con la base de datos utilizando código JavaScript. El uso de JSON como formato de almacenamiento se alinea con la naturaleza de JavaScript y facilita el manejo de los datos en aplicaciones que utilizan JavaScript tanto en el servidor como en el cliente.

Eficiencia en la transmisión de información

JSON es un formato de texto simple y compacto, lo que facilita la transferencia de datos a través de la red. Además, al ser un formato basado en texto, es legible por humanos y facilita el proceso de depuración y el trabajo con los datos almacenados en MongoDB.

En resumen, MongoDB utiliza el formato JSON como su formato de almacenamiento debido a su flexibilidad, compatibilidad con lenguajes de programación, integración con JavaScript y eficiencia en la transferencia de datos. Esto permite a los desarrolladores trabajar con datos estructurados y no estructurados de manera flexible y eficiente.

Sintaxis básica de JSON

```
{
  "llave1": "valor1",
  "llave2": "valor2",
  "llave3": "valor3",
  "llave4": 7,
  "llave5": null,
  "favAmigos": ["Kolade", "Nithya", "Dammy", "Jack"],
  "favJugadores": {"uno": "Kante", "dos": "Hazard", "tres": "Didier"}
}
```

MongoFramework

Entity Framework de manera nativa no tiene soporte oficial para MongoDB, ya que se enfoca principalmente para Base de datos relacional como PostgrestSQL o SQL Server. Debido a esto se decidió el uso de la Liberia MongoFramework la cual era la elección más madura dentro del ecosistema y cuenta con mayor uso y mantenimiento.

Existe soporte no oficial de algunos proyectos para que se pueda utilizar Entity Framework junto a Bases no relacionales, pero son de tipo experimental y no soportan las versiones más recientes de .NET. Un ejemplo de esto es la librería BlueShift Entity Core MongoDB.

MongoFramework es una librería de mapeo de objetos ODM (Object-document Mapper) siendo un concepto similar al de los ORMs junto a las Bases de tipo relacional.

La librería ofrece una capa de abstracción sobre la librería oficial de MongoDB Driver facilitando la interacción y el mapeo de las colecciones de objetos desde la Base MongoDB hacia la aplicación .NET. Esto con el objetivo de simplificar la comunicación con la Base de datos.

La idea detrás de MongoFramework es traer y emulas muchas de las características de Entity Framework. Las más destacables son las siguientes:

Seguimiento de cambios de las entidades

De manera similar a Entity Framework, el ODM realiza un seguimiento automático de los cambios realizados a las entidades durante su ciclo de vida dentro de la aplicación. Esto permite que el ODM decida por sí mismo que actualizaciones debe realizar sobre el documento almacenado en la base.

Ejemplo

```
using (var context = new MyDbContext())
{
   var customer = context.Customers.Find(1);
   customer.Name = "John Doe";
   context.SaveChanges();
```

}

En este caso el método SaveChanges() le permite al ODM detectar y generar automáticamente el código necesario para actualizar el documento en base a los cambios sufridos por la Entidad.

Mapeo fluido

La librería permite definir y configurar el mapeo entre las entidades del modelo y las colecciones de la base de datos mediante código. El mapeo permite establecer las relaciones entre los documentos, configurar restricciones, los índices, entre otros.
La creación de las colecciones se realiza por medio de atributos que le permitirán a la librería crear las colecciones según como la quiera el desarrollador.

Operaciones LINQ

Se permite el uso del lenguaje de consulta LINQ para realizar consultas contra la base MongoDB, lo que facilita la escritura de consultas complejas y la manipulación de los objetos.
El método AsQueryable() se utiliza para convertir una colección de MongoDB en un objeto IQueryable<T>, lo cual permite utilizar la sintaxis de consultas LINQ sobre esa colección. La cual se le pueden agregar Filtros y proyecciones adicionales antes de ejecutarla.

¿Qué es MongoDbContext y MongoDbSet?
Al igual que en Entity Framework, MongoFramework basa su uso en los contextos, específicamente MongoDbContext.

La siguiente imagen nos sirve para representar como se puede crear un contexto específico para la aplicación extendiendo del MongoDbContext

```
namespace PruebaMongo.Repository;

11 referencias
public class AppContext : MongoDbContext
{

    5 referencias
    public AppContext(IMongoDbConnection connection) : base(connection)  {}

    6 referencias
    public MongoDbSet<Property> Propiedades { get; set; }
    2 referencias
    public MongoDbSet<Agente> Agentes { get; set; }
    3 referencias
    public MongoDbSet<User> Users { get; set; }
    1 referencia
    public MongoDbSet<Contacto> Messages { get; set; }

    1 referencia
    public MongoDbSet<Visit> Visita { get; set; }
}
```

MongoDbSet son una colección de objetos que le permite al desarrollador interactuar con las colecciones y los documentos que están almacenan. Se puede encapsular su acceso a través de un repositorio.

```
4     namespace PruebaMongo.Repository.Users;
5
6     public class UserRepository : IUserRepository
7     {
8         private readonly AppContext _context;
9
10        public UserRepository()...
14
15        public User? GetUserById(string id)
16        {
17            return _context.Users.FirstOrDefault(user => user.Id == ObjectId.Parse(id));
18        }
19
20        public void UpdateUser(User user)
21        {
22            this._context.Users.Update(user);
23            _context.SaveChanges();
24        }
25
26        public void Insert(User user)
27        {
28            _context.Users.Add(user);
29            _context.SaveChanges();
30        }
31    }
32
```

Azure CosmosDB

Azure CosmosDB es un servicio de Base de datos en la nube, que le permite a las empresas almacenar y gestionar grandes volúmenes de datos estructurados, semiestructurados y no estructuras. Se debe recalcar este punto ya que CosmosDB permite la integración con Bases de datos de diversos tipos ya sea SQL o NoSQL.

El uso de CosmosDB permite la capacidad de proporcionar una baja latencia y una alta disponibilidad a nivel mundial. Esto se logra mediante la replicación automática

de los datos a través de las distintas regiones sobre las que se encuentran los usuarios.

CosmosDB le provee a los desarrolladores distintas Apis por las que pueden acceder a la información según qué tipo de modelado de datos hayan elegido. Para el caso de este TP investigativo se usó la API desarrollada específicamente para MongoDB.

¿Como funciona a nivel técnico?

A nivel técnico, la API de Azure Cosmos DB para cuentas de MongoDB implementa un traductor y un mapeo entre las operaciones de MongoDB y las operaciones subyacentes en Azure Cosmos DB. Esto permite que las aplicaciones que utilizan la API de MongoDB se comuniquen con CosmosDB sin requerir modificaciones significativas en el código existente.

Cuando se realiza una operación utilizando la API de Azure Cosmos DB para cuentas de MongoDB, la API traduce esa operación en una forma compatible con Azure Cosmos DB. A continuación, se realiza la operación en el backend de Azure Cosmos DB utilizando sus capacidades de almacenamiento y administración de datos distribuidos. El resultado de la operación se devuelve a la aplicación como si se hubiera realizado directamente en una base de datos de MongoDB.

Cuando una aplicación realiza una operación utilizando la API de Azure Cosmos DB para cuentas de MongoDB, el siguiente proceso ocurre a nivel técnico:

Conexión a la cuenta de Azure Cosmos DB: La aplicación establece una conexión a la cuenta de Azure Cosmos DB utilizando las credenciales y la cadena de conexión proporcionadas.
Traducción de operaciones de MongoDB: La API de Azure Cosmos DB traduce las operaciones de MongoDB realizadas por la aplicación en un formato compatible con Azure Cosmos DB.

Comunicación con Azure Cosmos DB: Las operaciones traducidas se envían a Azure Cosmos DB a través de la red. Esto implica la comunicación con el backend de Azure Cosmos DB, que es una infraestructura distribuida que maneja el almacenamiento y la administración de datos.

Ejecución de operaciones en Azure Cosmos DB: Azure Cosmos DB ejecuta las operaciones en su backend distribuido. Utiliza su motor de almacenamiento y su modelo de datos para procesar las operaciones y acceder a los datos almacenados.

Retorno de resultados: Una vez que se completa la operación en Azure Cosmos DB, los resultados se devuelven a la aplicación que realizó la operación. Esto puede incluir documentos, respuestas a consultas, códigos de estado y cualquier otra información relevante.

Conexión a la API de AZURE.

Bases relacionales vs No Relacionales

Las bases de datos relacionales se basan en un modelo de datos relaciones que incluye el uso de tablas para organizar y almacenar los datos. Usa SQL como lenguaje para poder operar sobre la información almacenada. Es el paradigma mas antiguo y el mas utilizado con respecto al almacenamiento de información.

Las características de las bases relacionales son:

- Estructura y esquema fijos: Los datos siguen un esquema predefinido y deben cumplir con la estructura del esquema.
- Consultas via SQL: Se usa SQL para realizar consultas y manimulaciones de datos.
- Escalables verticalmente: La escalabilidad de este tipo de bases se basa primariamente en el aumento de los recursos del hardware dentro un solo servidor.

Las Bases de datos no relacionales, por otro lado, se destacan por usar estructuras de datos de una manera mas flexible y no relacionada entre si mediante el uso de documentos y llaves valor.

Las características principales de las bases no relacionales son:

- Estructura flexible: Este tipo de bases permite el almacenamiento de estructuras flexibles, lo que permite agregar o modificar campos sin tener que cambiar el esquema.
- Escalables horizontalmente: Este tipo de bases, como MongoDB, están diseñadas especialmente para manejar grandes volúmenes de datos distribuyendo la carga entre diferentes servidores.
- Alto rendimiento y baja latencia: Estas bases están optimizadas para lograr un buen rendimiento y bajo tiempo de acceso a datos.

Conclusiones

En conclusión, esta investigación demostró que, aunque el Entity Framework no tiene un soporte directo para bases de datos NoSQL, existen alternativas como MongoFramework que nos permiten aprovechar los beneficios de los ORM y las operaciones de bases de datos NoSQL. Esto amplía nuestras opciones para desarrollar aplicaciones escalables y flexibles, adaptadas a diferentes requisitos y necesidades de almacenamiento de datos.

Referencias/Bibliografía

Como decidir que tipo de BB.DD usar

Diferencias entre SQL y NoSQL

Diferencias entre SQL y NoSQL 2

Ventajas de MongoDB sobre BB.DD tradicionales

Mongo Framework

Introducción a MongoDB

JSON introducción