RAVENDB

NoSQL Document Database

Getting Started | Creating CRUD

Ricardo de Luna Galdino
Software Engineer

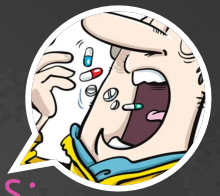
github.com/ricardogaldino/microlearning-ravendb

Agile learning culture

Microlearning:

- It is a teaching methodology that subdivides a subject into **smaller doses of content**, with quick activities, helping to understand and retain this content;

Knowledge Pills:

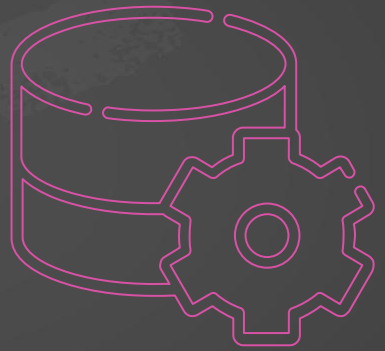


- These are **small contents** presented to the professional so that he **can assimilate them in a more focused and objective** way, improving efficiency and enhancing the results obtained;
- **Microlearning is composed** of several **Knowledge Pills**;

github.com/ricardogaldino/microlearning-ravendb

01

Introduction



Introduction to the Database
Non-Relational and **NoSQL**

github.com/ricardogaldino/microlearning-ravendb

A brief history...

- The term NoSQL **appeared** in mid-**1998** by Software Engineer "**Carlo Strozzi**" and initially would be "**NoREL**", because he was developing a database that would not be relational;

- "**No SQL**", for some it became "**Not Only SQL**" because it does not replace, in all aspects, relational databases, in fact, currently (2023) they complement each other;

- It was driven by **Google** (BigTable, 2004) and **Amazon** (Dynamo, 2007) when they **sought a new form of data storage** that **could scale horizontally,** with the creation of clusters. It was a **good idea**, both from a **technical** point of view and from an **economic** point of view;

Carlo Strozzi

Concepts about NoSQL

Non-relational

- The term "**Non-Relational**" refers to DBs that do not follow the "Relational" architecture, that is, they do **not use** the **Tables**, **Rows** and **Columns** schema when storing data;
- Data can be stored, for example: as **Key/Value** pairs, as **Documents** (XML, JSON, ..) or as **Graphs** (nodes and edges);

NoSQL

- Originally the term "NoSQL" referred to databases that do **not use SQL** (Structured Query Language) **for queries**;

References...

- Microsoft, 2023. "Non-relational data and NoSQL".
- Microsoft, 2023. "What are NoSQL databases?".
- Amazon, 2023. "What is NoSQL?".

Advantages of NoSQL

- **Flexibility**: they can store and combine any type of data, whether structured or unstructured, unlike relational databases, which only store in a structured way;
- **Scalability**: they support growth in volume of data and simultaneous access well because they are built to scale horizontally (distributed environments (clusters)), unlike relational databases, which generally scale vertically;
- **Availability**: they are efficient in data replication, that is, if one or more servers go down, another is able to continue the work. With this, they avoid the loss of data and, consequently, of dissatisfied customers;
- **High performance**: they are built to have great performance, measured by the transfer rate (amount of data (bits) transferred per time (seconds)) and latency (time required to make a request and receive the response);
- **Dynamic schema**: accepts changes in the data structure (schema) without major problems;
- **Open source**: Because they are open source, they require no licensing fees and can be deployed cost-effectively;

Disadvantages of NoSQL

- **Data consistency** (integrity constraints to ensure data accuracy and reliability) is not one of your priorities. This responsibility will be up to the application that will use NoSQL;

  For example, it natively allows data duplication;

- **Data normalization** is also not one of your priorities. It uses denormalized data models (it contains all the necessary information even if it is repeated);

- They were not made for **complex queries** with reference between several "**joins**" entities;

- Doesn't contain as many advanced features and tools as relational databases;

- There is a lack of **standardization** and incompatibility with each other. Each uses its query language and implements concepts in its own way;

NOTE!

Relational Databases prioritize these items Natively and that's why they are still necessary and work together with NoSQL databases!

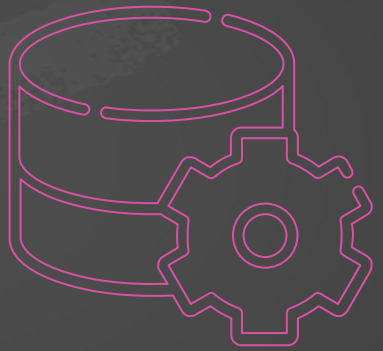
NewSQL movement! Will it be the future?

- **NewSQL** databases seek to promote the same improvement in performance and scalability as NoSQL systems, without giving up the benefits of relational databases and the SQL language;
- The term was first used by Data Engineer "**Matthew Aslett**" in a **2011** research paper discussing the emergence of a new generation of database management systems;
- The "BOOM" has not yet occurred, but what we see is, little by little, the **inclusion** of **Relational** Database (SQL) **resources in NoSQL** Databases and the same in reverse;

Matthew Aslett

02

RavenDB



Concepts

github.com/ricardogaldino/microlearning-ravendb

RavenDB - Concepts

- It is a **document**-oriented **NoSQL** database (default **JSON**), open-source, written in C# and with support for ACID transactions;
- Developed in **2010**, under the leadership of **Oren Eini**, by **Hibernating Rhinos Ltd**;
- **Multi-platform**, running on Windows, Linux, MacOS, Docker and Raspberry Pi;
- In **2019** it began to be offered, as a **Cloud Service**, on AWS, Azure and GCP;

Oren Eini

- Uses the **RQL** (Raven Query Language) **for queries**;
- **LINQ** (Language Integrated Query) support in **C#**;
- Has **clients** in C#, C++, Java, Node.js, Python, Ruby, PHP, Elixir (Erlang) and Go;
- Do **dynamic indexing**. All queries use indexes. If it does not exist, an index is **automatically created**;

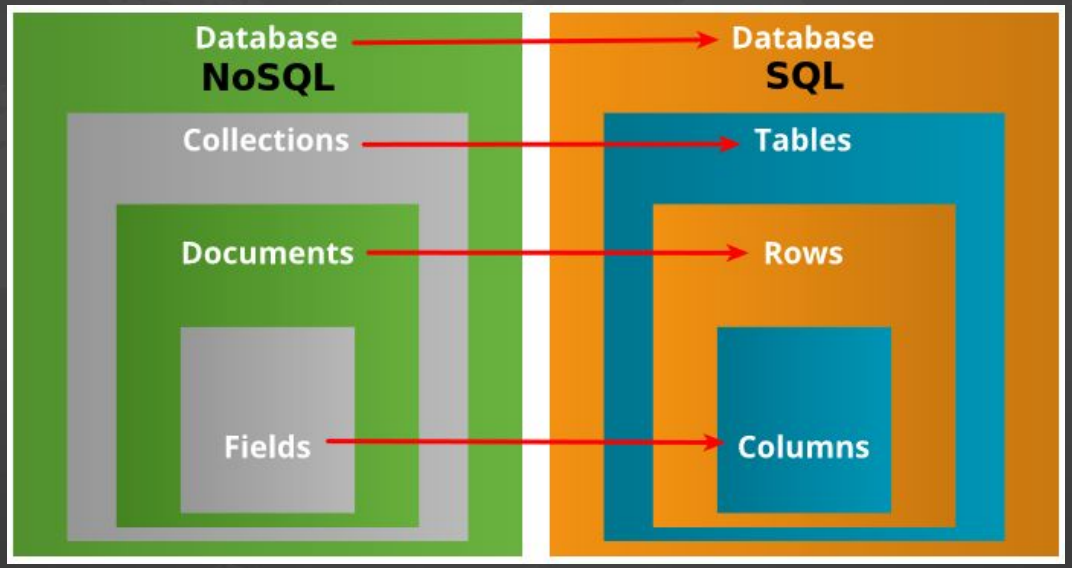
References...

- RavenDB 2023. “Documentation”.
- RavenDB 2023. “RDL (Raven Query Language)”.

github.com/ricardogaldino/microlearning-ravendb

RavenDB - Architecture

We can **compare** the architecture of a "**NoSQL**" **Document** DB with a "**SQL**" Relational DB:

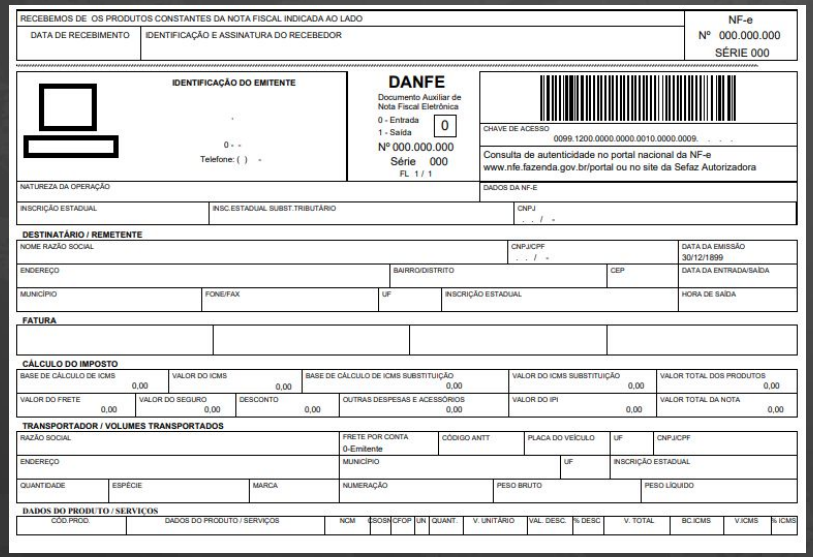


github.com/ricardogaldino/microlearning-ravendb

RavenDB - Modeling

A DB of the "**Document**" type is oriented to **aggregates** (a term that comes from **Domain-Driven Design**), which means, in a simplified way, that each document is a collection of related objects, treated as a single unit over a context.

Ex: Invoice, its items and other data would be a **single denormalized "JSON" document**.



RECEBEMOS DE OS PRODUTOS CONSTANTES DA NOTA FISCAL INDICADA AO LADO

DATA DE RECEBIMENTO | IDENTIFICAÇÃO E ASSINATURA DO RECEBEDOR

NF-e
Nº 000.000.000
SÉRIE 000

IDENTIFICAÇÃO DO EMITENTE

Telefone: ( )

DANFE
Documento Auxiliar de Nota Fiscal Eletrônica
0 - Entrada
1 - Saída [0]
Nº 000.000.000
Série 000
FL 1 / 1

CHAVE DE ACESSO
0099 1200 0000 0000 0010 0000 0009

Consulta de autenticidade no portal nacional da NF-e www.nfe.fazenda.gov.br/portal ou no site da Sefaz Autorizadora

NATUREZA DA OPERAÇÃO | DADOS DA NF-E

INSCRIÇÃO ESTADUAL | INSC.ESTADUAL SUBST.TRIBUTÁRIO | CNPJ

DESTINATÁRIO / REMETENTE

NOME RAZÃO SOCIAL | CNPJ/CPF | DATA DA EMISSÃO 30/12/1899

ENDEREÇO | BAIRRO/DISTRITO | CEP | DATA DA ENTRADA/SAÍDA

MUNICÍPIO | FONE/FAX | UF | INSCRIÇÃO ESTADUAL | HORA DE SAÍDA

FATURA

CÁLCULO DO IMPOSTO

BASE DE CÁLCULO DE ICMS	VALOR DO ICMS	BASE DE CÁLCULO DE ICMS SUBSTITUIÇÃO	VALOR DO ICMS SUBSTITUIÇÃO	VALOR TOTAL DOS PRODUTOS
0,00	0,00	0,00	0,00	0,00

VALOR DO FRETE	VALOR DO SEGURO	DESCONTO	OUTRAS DESPESAS E ACESSÓRIOS	VALOR DO IPI	VALOR TOTAL DA NOTA
0,00	0,00	0,00	0,00	0,00	0,00

TRANSPORTADOR / VOLUMES TRANSPORTADOS

RAZÃO SOCIAL | FRETE POR CONTA 0-Emitente | CÓDIGO ANTT | PLACA DO VEÍCULO | UF | CNPJ/CPF

ENDEREÇO | MUNICÍPIO | UF | INSCRIÇÃO ESTADUAL

QUANTIDADE | ESPÉCIE | MARCA | NUMERAÇÃO | PESO BRUTO | PESO LÍQUIDO

DADOS DO PRODUTO / SERVIÇOS

CÓD.PROD.	DADOS DO PRODUTO / SERVIÇOS	NCM	CSOSN	CFOP	UN	QUANT.	V. UNITÁRIO	VAL. DESC.	% DESC	V. TOTAL	BC.ICMS	V.ICMS	% ICMS

```
"buyer": {
    "name": "teste",
    "address": {
        "city": {
            "code": "3550308",
            "name": "jundiai"
        },
        "state": "SP",
        "district": "centro",
        "street": "rua petronilha antunes",
        "postalCode": "13207760",
        "number": "204",
        "country": "BRA"
    },
    "federalTaxNumber": 8662968678
},
"items": [{
    "code": "2617",
    "unitAmount": 9.98,
    "quantity": 5,
    "cfop": 5102,
    "ncm": "47079000",
    "codeGTIN": "SEM GTIN",
    "codeTaxGTIN": "SEM GTIN",
    "tax": {
```

github.com/ricardogaldino/microlearning-ravendb

RavenDB - Identifiers (IDs)

- Every document in a RavenDB database has a **unique string** associated with it, called an **identifier**;
- It uses HiLo conventions and algorithms to produce identifiers;
- By **convention**, RavenDB looks for the property or field named "**Id**" (case sensitive);
- The identifier **can be generated automatically** according to the rules:
    - If you pass "**NULL**" in the "Id" field, an identifier will be generated in the "**collection/number-tag**" format
    - If you pass "**EMPTY**" in the "Id" field, an identifier will be generated in the "**GUID**" format;

Url: https://ravendb.net/docs/article-page/5.4/csharp/client-api/document-identifiers/working-with-document-identifiers

RavenDB - Setup (Docker)

To **run** RavenDB locally we can use a **docker-compose**:

Url: **https://github.com/ricardogaldino/docker-ravendb/blob/main/docker-compose.yml**

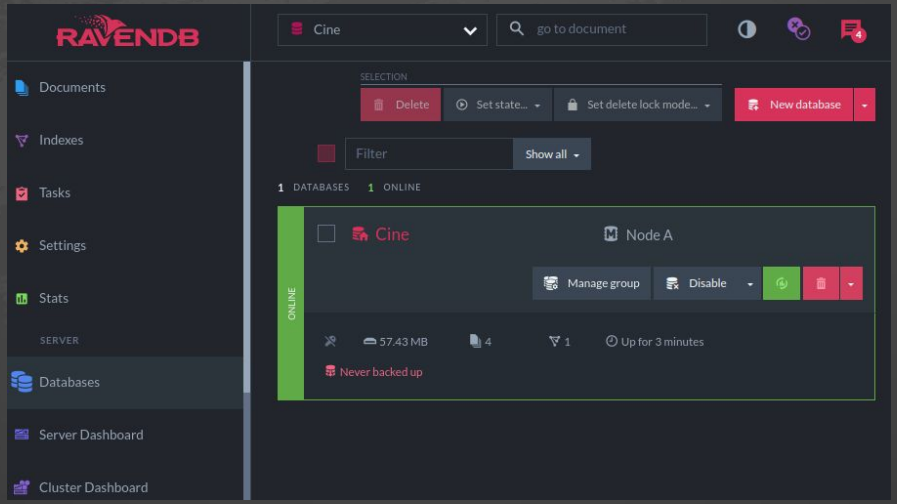
```yaml
version: '3.7'
services:

  ravendb:
    image: ravendb/ravendb:ubuntu-latest
    container_name: raven-db
    hostname: raven-db
    ports:
      - 8088:8080
      - 38888:38888
    environment:
      - RAVEN_Security_UnsecuredAccessAllowed=PublicNetwork
      - RAVEN_Setup_Mode=None
      - RAVEN_License_Eula_Accepted=true
    volumes:
      - .ravendb/volumes:/opt/RavenDB/Server/RavenData
    networks:
      - raven-network

networks:
  raven-network:
    driver: bridge
```

github.com/ricardogaldino/microlearning-ravendb

RavenDB - Management Studio

When installing RavenDB, it comes with the **Management Studio (Web) tool**, where it is possible to **administer the database**.

Url: **http://localhost:8080/studio/index.html**

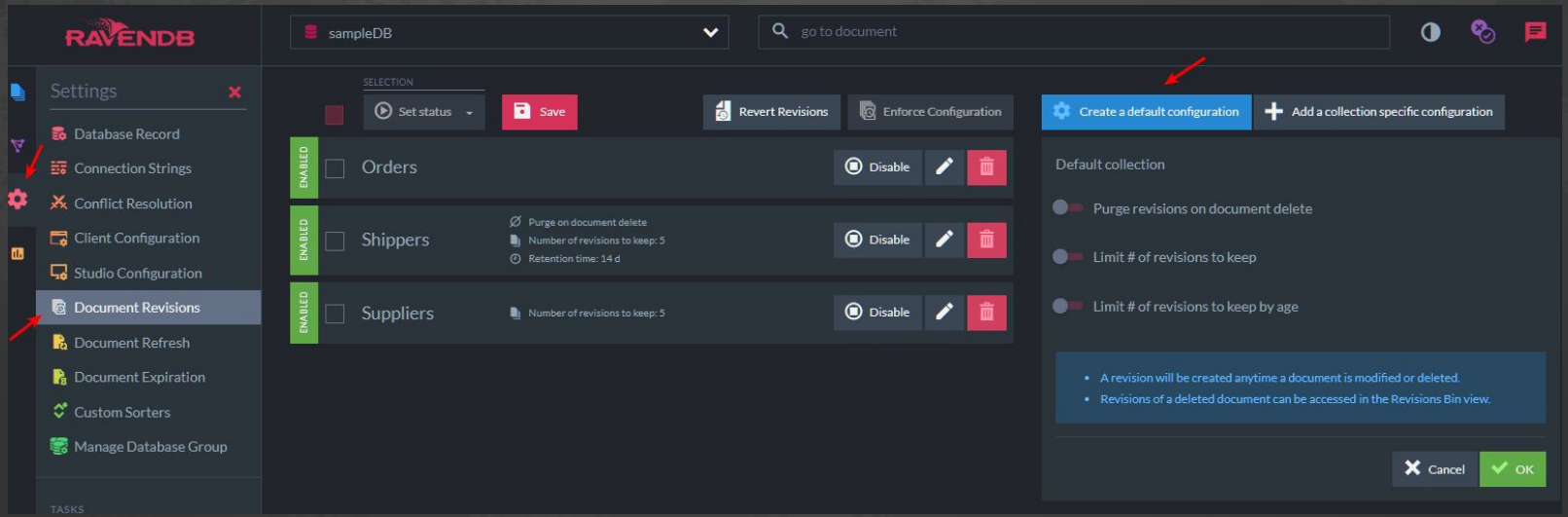


github.com/ricardogaldino/microlearning-ravendb

RavenDB - Document History

The “**Revisions**” feature will **create a snapshot** for a document every time the **document** is updated and after deletion.

Url: **https://ravendb.net/docs/article-page/4.2/csharp/server/extensions/**



github.com/ricardogaldino/microlearning-ravendb

03

RavenDB

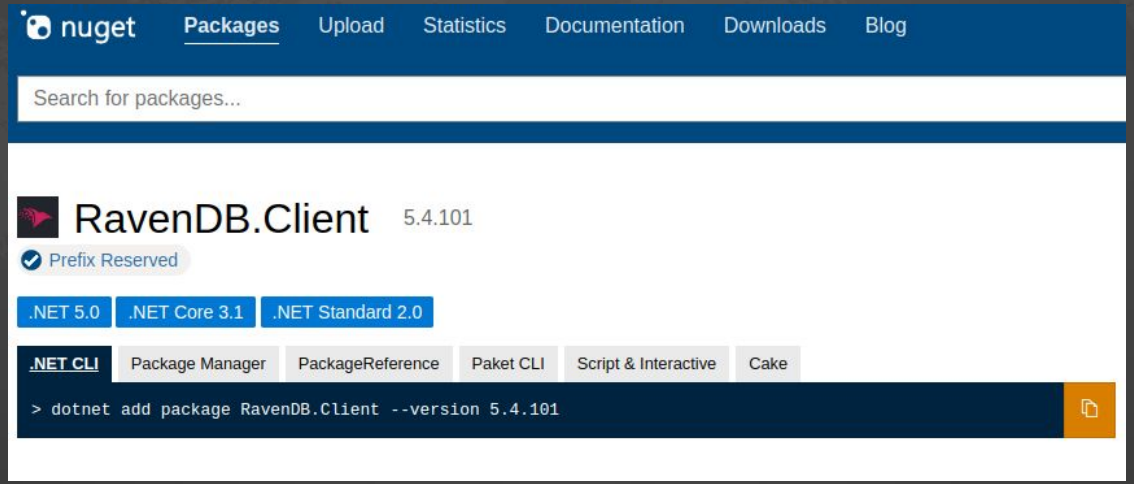
Setup for .NET

github.com/ricardogaldino/microlearning-ravendb

RavenDB - Client .NET

To use RavenDB in .NET applications, it is necessary to **install** the **RavenDB Client** package in the project by **NuGet**.

Url: **https://www.nuget.org/packages/RavenDB.Client**



RavenDB - Sample (.NET)

To use RavenDB in .NET applications, the minimum code required is as follows:

```
class Program
{
    static void Main(string[] args)
    {
        var documentStore = new DocumentStore();
        documentStore.Url = "http://localhost:8081";
        documentStore.Initialize();

        using (var session = documentStore.OpenSession())
        {
            var cliente = new Cliente
            {
                Nome = "Robson",
                DataDeNascimento = Convert.ToDateTime("05/04/1978")
            };
            session.Store(cliente);
            session.SaveChanges();
        }
        documentStore.Dispose();

        Console.WriteLine("Operação concluída");
        Console.ReadKey();
    }
}
```

- The "**Document Store**" is the main Client API object that establishes and manages the (HTTP) **connection** between your client application and the RavenDB server. A single Document Store instance (**Singleton Pattern**) must be created during the lifetime of your application;

- The "**Document Session**" represents a **transaction** in RavenDB and the changes are persisted on the server when the "SaveChanges()" command is called (**Unit of Work Pattern**);

github.com/ricardogaldino/microlearning-ravendb

04 RavenDB

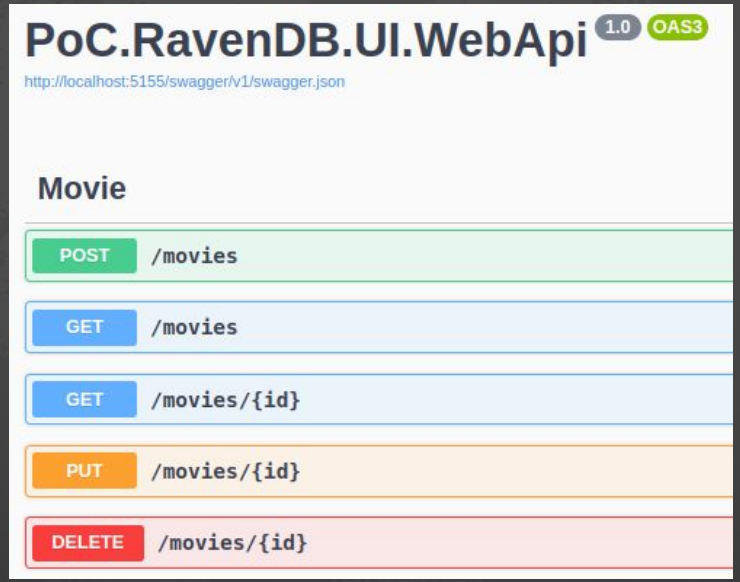
CRUD in .NET

github.com/ricardogaldino/microlearning-ravendb

RavenDB - CRUD (API)

We are going to use as a base for our CRUD (Create, Read, Update, Delete) the .NET application that registers movies. It will store, in addition to other data, the ratings given by film critics.

Url: **https://github.com/ricardogaldino/poc-ravendb-dotnet**



github.com/ricardogaldino/microlearning-ravendb

RavenDB - Documents (JSON)

To test our CRUD I separated some examples of documents.

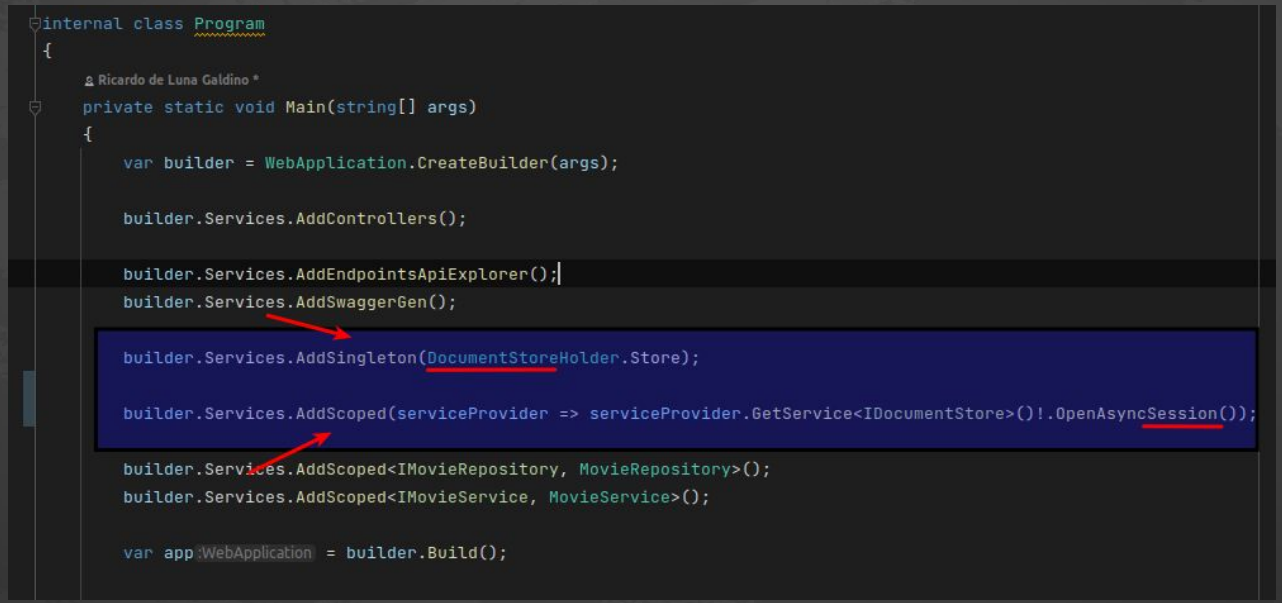
Url: **https://github.com/ricardogaldino/poc-ravendb-dotnet/blob/main/sample/movies.json**


```json
{
    "id": "60e5b801-f5c6-4b7d-a507-47b705a23a82",
    "name": "Interestelar",
    "year": 2014,
    "rating": 9,
    "genre": "Ficção cientifica",
    "director": "Christopher Nolan",
    "country": "USA",
    "actors": [
      "Matthew McConaughey",
      "Anne Hathaway",
      "Jessica Chastain",
      "Mackenzie Foy",
      "Matt Damon"
    ],
    "languages": [
      "English",
      "Portuguese",
      "Spanish"
    ],
    "subtitles": [
      "English",
      "Portuguese",
      "Spanish"
    ]
  }
```

RavenDB - Dependency Injection

Following the creator “Oren Eini” we would have something like this for the Dependency Injection configuration:

Url: **https://ayende.com/blog/187906-B/using-ravendb-unit-of-work-and-net-core-mvc**



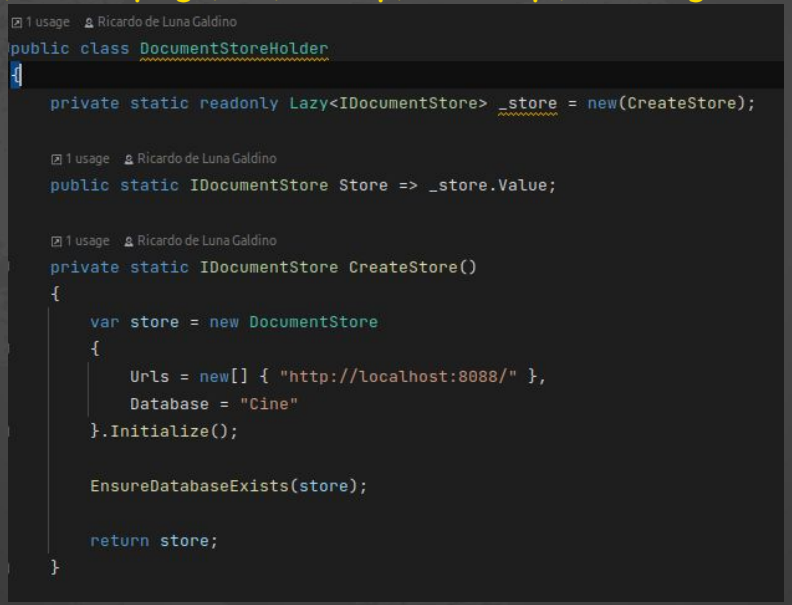
```
internal class Program
{
    Ricardo de Luna Galdino *
    private static void Main(string[] args)
    {
        var builder = WebApplication.CreateBuilder(args);

        builder.Services.AddControllers();

        builder.Services.AddEndpointsApiExplorer();
        builder.Services.AddSwaggerGen();

        builder.Services.AddSingleton(DocumentStoreHolder.Store);

        builder.Services.AddScoped(serviceProvider => serviceProvider.GetService<IDocumentStore>()!.OpenAsyncSession());

        builder.Services.AddScoped<IMovieRepository, MovieRepository>();
        builder.Services.AddScoped<IMovieService, MovieService>();

        var app:WebApplication = builder.Build();
```

github.com/ricardogaldino/microlearning-ravendb

RavenDB - Connection configuration

Following the official documentation we would have something like this for configuring the connection with RavenDB:

Url: **https://ravendb.net/docs/article-page/5.4/csharp/client-api/creating-document-store**



```
1 usage  Ricardo de Luna Galdino
public class DocumentStoreHolder
{
    private static readonly Lazy<IDocumentStore> _store = new(CreateStore);

    1 usage  Ricardo de Luna Galdino
    public static IDocumentStore Store => _store.Value;

    1 usage  Ricardo de Luna Galdino
    private static IDocumentStore CreateStore()
    {
        var store = new DocumentStore
        {
            Urls = new[] { "http://localhost:8088/" },
            Database = "Cine"
        }.Initialize();

        EnsureDatabaseExists(store);

        return store;
    }
```

github.com/ricardogaldino/microlearning-ravendb

RavenDB - DB configuration

Following the official documentation we would have something like this for configuring the RavenDB database:

Url: **https://ravendb.net/docs/article-page/5.3/csharp/client-api/operations/server-wide/create-database**

```
1 usage  Ricardo de Luna Galdino
private static void EnsureDatabaseExists(IDocumentStore store,
    string database = null,
    bool createDatabaseIfNotExists = true)
{
    database = database ?? store.Database;

    if (string.IsNullOrWhiteSpace(database))
    {
        throw new ArgumentException( message: "Value cannot be null or whitespace.", nameof(database));
    }

    try
    {
        store.Maintenance.ForDatabase(database).Send(new GetStatisticsOperation());
    }
    catch (DatabaseDoesNotExistException)
    {
        if (createDatabaseIfNotExists == false)
        {
            throw;
        }

        try
        {
            store.Maintenance.Server.Send(new CreateDatabaseOperation(new DatabaseRecord(database)));
        }
        catch (ConcurrencyException)
        {
            // The database was already created before calling CreateDatabaseOperation
        }
    }
}
```

github.com/ricardogaldino/microlearning-ravendb

RavenDB - Repository (CRUD)

Sample code with CRUD operations on RavenDB database:

```
private readonly IAsyncDocumentSession _session;

Ricardo de Luna Galdino
public MovieRepository(IAsyncDocumentSession session)
{
    _session = session;
}
```

```
public async Task Create(Movie movie)
{
    await _session.StoreAsync(movie);
    await _session.SaveChangesAsync();
}
```

```
public async Task<Movie> ReadById(string id)
{
    return await _session.LoadAsync<Movie>(id);
}
```

```
public async Task<IEnumerable<Movie>> ReadByName(string name)
{
    return await _session.Query<Movie>(). // IRavenQueryable<Movie>
        Where(m :Movie => m.Name == name). // IQueryable<Movie>
        ToListAsync(); // Task<List<...>>
}
```

```
public async Task<IEnumerable<Movie>> ReadAll()
{
    return await _session.Query<Movie>().ToListAsync();
}
```

```
public async Task Update(Movie movie)
{
    movie.UpdatedDate = DateTime.Now;
    await _session.SaveChangesAsync();
}
```

```
0+1 usages  Ricardo de Luna Galdino *
public async Task Delete(string id)
{
    _session.Delete(id);
    await _session.SaveChangesAsync();
}
```

github.com/ricardogaldino/microlearning-ravendb

RavenDB - Service (Validations)

Sample code with validations to ensure database integrity:

```
0+1 usages  Ricardo de Luna Galdino
public async Task Create(Movie movie)
{
    var alreadyExists :Movie? = await ReadById(movie.Id!);
    if (!string.IsNullOrWhiteSpace(alreadyExists?.Id))
    {
        throw new ArgumentException( message: AlreadyExistsError);
    }

    alreadyExists = (await ReadByName(movie.Name)).FirstOrDefault();
    if (!string.IsNullOrWhiteSpace(alreadyExists?.Name))
    {
        throw new ArgumentException( message: AlreadyExistsError);
    }

    await _movieRepository.Create(movie);
}
```

```
0+1 usages  Ricardo de Luna Galdino *
public async Task Update(string id, UpdatedMovieDto updatedMovie)
{
    var movie = await ReadById(id);
    if (string.IsNullOrWhiteSpace(movie?.Id))
    {
        throw new ArgumentException( message: DoesNotExistError);
    }

    var duplicateName :IEnumerable<Movie> = await ReadByName(updatedMovie.Name);
    if (duplicateName.Any(m :Movie => (m.Id != movie.Id) & (m.Name == updatedMovie.Name)))
    {
        throw new ArgumentException( message: DuplicateNameError);
    }

    MovieMapper.Map(updatedMovie, movie);

    await _movieRepository.Update(movie);
}
```

```
0+1 usages  Ricardo de Luna Galdino *
public async Task Delete(string id)
{
    var movie = await ReadById(id);
    if (string.IsNullOrWhiteSpace(movie?.Id))
    {
        throw new ArgumentException( message: DoesNotExistError);
    }

    await _movieRepository.Delete(id);
}
```

github.com/ricardogaldino/microlearning-ravendb

05

RavenDB

Hands-on

github.com/ricardogaldino/microlearning-ravendb

RavenDB - Hands-on

- Imagine that you are a famous programmer, known worldwide as “Hackerman”. Born in Maranguape on the border with Chechnya, he started programming early and at the age of 7 he was already nicknamed by his friends: “nervous fingers”, “bit brush”, “the code grinder”, among others...
- Winner of the Oscar for best programmer, he participated in action films such as “Kung Fury”;
- You will take Man to Mars with your software!
- Finally... “you are the chosen one!”

- You were hired by “Bollywood” from India to **develop an endpoint in the “RavenDB CRUD API”, which brings the best rated movie by the critics according to its cinematographic genre**;

github.com/ricardogaldino/microlearning-ravendb

Thanks!

- **LinkedIn:**
  linkedin.com/in/ricardo-galdino
- **GitHub:**
  github.com/ricardogaldino
- **WhatsApp Group:**
  devnews.com.br

Ricardo Galdino

github.com/ricardogaldino/microlearning-ravendb