

---

# EF Core для работы с MongoDB: Стоит ли использовать MongoDBProvider?

---

Дзицкий Виктор

# Немного о себе

---



**Виктор Дзицкий**

TeamLead .NET Developer, MCSD

Компания Solarlab

г. Севастополь

## Контакты

✉ [dzitskiy@gmail.com](mailto:dzitskiy@gmail.com)

✈ Dzitskiy, dzitskiy\_dev

# О чем поговорим?

- Введение в проблематику
- Обзор MongoDB Provider for EF Core
- Сравнение с MongoDB.Driver
- Библиотека MongoDB.Entities
- Рекомендации и выводы



---

# Введение в проблематику

---

# Введение в проблематику

---

- Кратко о EF Core и MongoDB (ORM vs документная БД)
- Зачем использовать EF Core с MongoDB?
- Какие есть альтернативы?



# Для кого этот доклад

---

- Для тех, кто уже использует ORM (EF Core)

# Для кого этот доклад

---

- Для тех, кто уже использует ORM (EF Core)
- Для тех, кто только планирует начать использовать MongoDB

# Для кого этот доклад

---

- Для тех, кто уже использует ORM (EF Core)
- Для тех, кто только планирует начать использовать MongoDB
- Для тех, кто хочет использовать MongoDB в типовых сценариях с помощью привычного интерфейса



# Для кого этот доклад

---

- Для тех, кто уже использует ORM (EF Core)
- Для тех, кто только планирует начать использовать MongoDB
- Для тех, кто хочет использовать MongoDB в типовых сценариях с помощью привычного интерфейса
- Для тех, кто хочет с наименьшими трудозатратами проверить правильность выбора и сравнить производительность RDBMS (например, PostgreSQL) и MongoDB

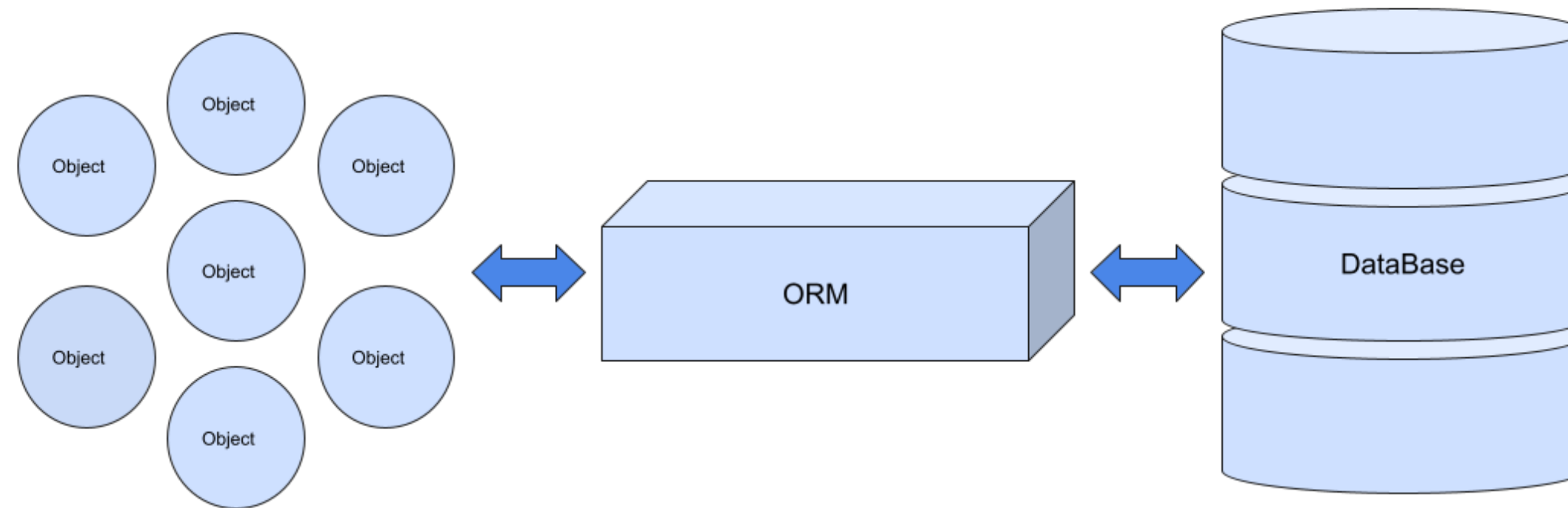
# Вопрос в зал

Вы применяете ORM  
в своих проектах?

# ORM

---

Object Relational Mapping (ORM) is a technique used in creating a "bridge" between object-oriented programs and, in most cases, relational databases.



# Основные задачи ORM

---

- **Маппинг** Сопоставление таблиц базы данных с классами, а столбцов — со свойствами классов.
- **Генерация SQL** Автоматическое создание SQL-запросов на основе LINQ или других конструкций.
- **Управление состоянием объектов** Отслеживание изменений в объектах и их синхронизация с базой данных.

# Особенности применения ORM

---

## Преимущества:

---

- Устраняет необходимость в повторяющемся коде SQL
- Сокращает время разработки
- Снижает затраты на разработку
- Преодолевает специфические для поставщика различия SQL

## Недостатки:

---

- Потеря производительности разработчика
- Разработчики теряют понимание того, что на самом деле делает код
- ORM имеет тенденцию быть медленным
- ORM не может конкурировать со сложными нативными запросами



# ignorantia legis non excusat

---

Незнание особенностей работы ORM не освобождает от ответственности за выполнение запросов, который ORM генерирует.

# ORMs for .NET

---

- LINQ to SQL
- LINQ to DB
- Dapper
- NHibernate
- Entity Framework Core

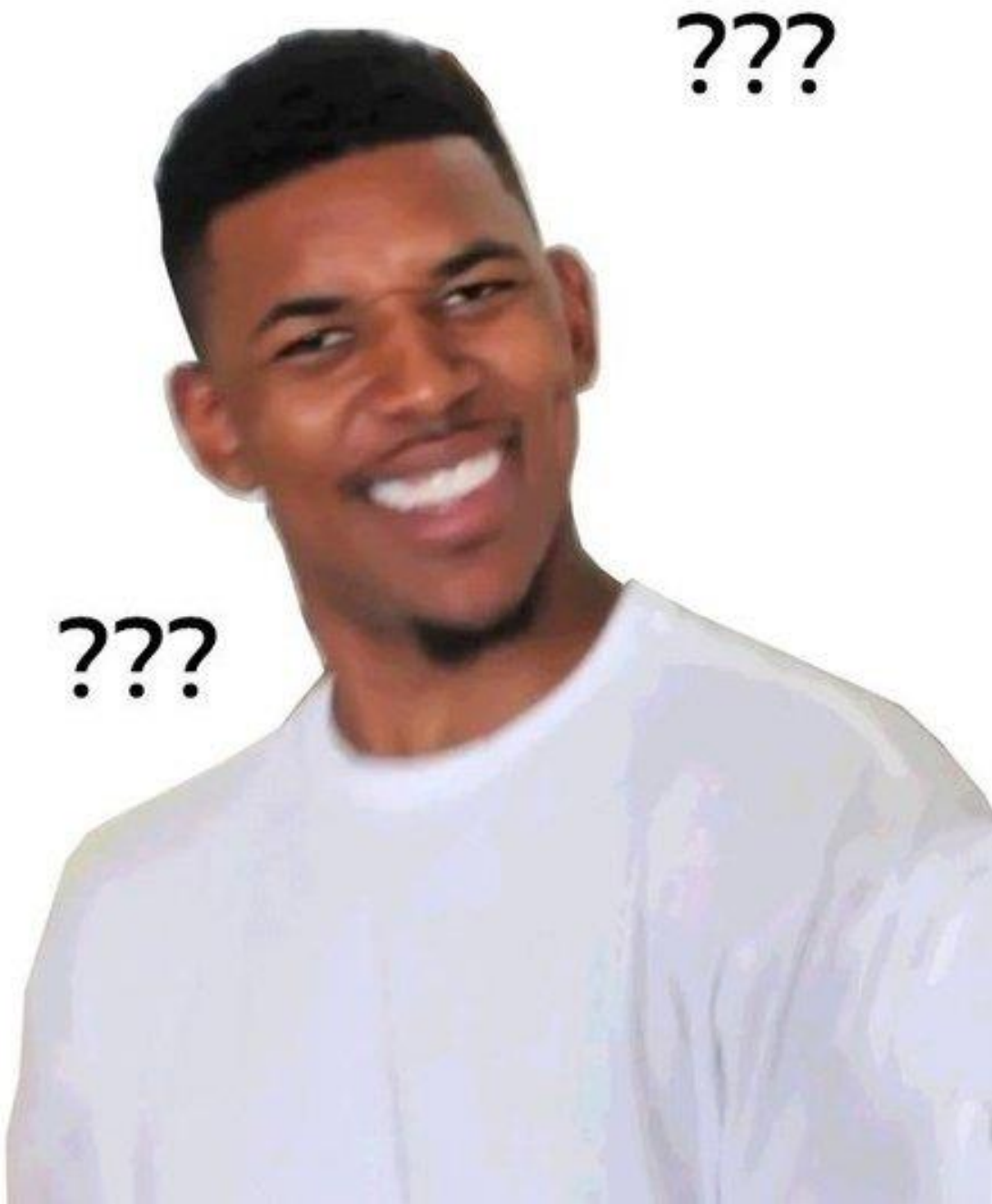
# ORMs for .NET

---


- LINQ to SQL
- LINQ to DB
- Dapper
- NHibernate
- **Entity Framework Core ([EF10](#))**

# MongoDB Entity Framework Core Provider

NuGet Package	Supported database engines	Maintainer / Vendor
<a href="#">Microsoft.EntityFrameworkCore.SqlServer</a>	Azure SQL, SQL Server 2012 onwards, Azure Synapse Analytics	<a href="#">EF Core Project</a> (Microsoft)
<a href="#">Microsoft.EntityFrameworkCore.Sqlite</a>	SQLite 3.46.1 onwards	<a href="#">EF Core Project</a> (Microsoft)
<a href="#">Microsoft.EntityFrameworkCore.InMemory</a>	EF Core in-memory database	<a href="#">EF Core Project</a> (Microsoft)
<a href="#">Microsoft.EntityFrameworkCore.Cosmos</a>	Azure Cosmos DB SQL API	<a href="#">EF Core Project</a> (Microsoft)
<a href="#">Npgsql.EntityFrameworkCore.PostgreSQL</a>	PostgreSQL	<a href="#">Npgsql Development Team</a>
<a href="#">Pomelo.EntityFrameworkCore.MySql</a>	MySQL, MariaDB	<a href="#">Pomelo Foundation Project</a>
<a href="#">MySql.EntityFrameworkCore</a>	MySQL	<a href="#">MySQL project</a> (Oracle)
<a href="#">Oracle.EntityFrameworkCore</a>	Oracle DB 11.2 onwards	<a href="#">Oracle</a>
<a href="#">MongoDB.EntityFrameworkCore</a>	MongoDB	<a href="#">MongoDB</a>




# MongoDB Entity Framework Core Provider




**James Kovacs** ✓ · 3-й  
Director of Engineering at MongoDB  
Канада · [Контактные сведения](#)  
500+ контакта


[Отправить сообщение](#) [+ Отслеживать](#) [Еще](#)



MongoDB




Harvard University




**Luce Carter** ✓ · 3-й  
Developer Advocate at MongoDB - No that is not a developer, no I do not want messages about developer jobs recruiters!  
Агломерация Манчестера, Великобритания · [Контактные сведения](#)  
298 контактов


[+ Отслеживать](#) [+ Установить контакт](#) [Еще](#)



MongoDB



The Manchester Metropolitan University



[Making MongoDB Easy for Entity Framework Users with MongoFramework](#)  
[Entity Framework Core & MongoDB: A New Era in Data Management](#)  
[Announcing MongoDB Provider for Entity Framework Core](#)

[Start building your next app now with MongoDB Provider for EF Core | StudioFP117](#)  
[EF Core 9: Evolving Data Access in .NET | OD535](#)




# Entity Framework Core для MongoDB

---

- MongoDB стала первой документоориентированной БД, для которой выпущен официальный провайдер Entity Framework Core
- Годовое сотрудничество MongoDB и Microsoft позволило создать провайдер, полностью совместимый с EF Core 8 и .NET 8
- Была реализована поддержка стандартных паттернов EF Core: DbContext, LINQ, ChangeTracking (Unit of Work)
- Провайдер построен поверх MongoDB C# Driver (применяется гибридный подход для переключения на низкоуровневый драйвер для специфичных функций MongoDB)

# MongoDB Entity Framework Core Provider

 **MongoDB.EntityFrameworkCore** 9.0.0

Prefix Reserved

.NET 8.0

.NET CLI

Package Manager

PackageReference

Central Package Management

Paket CLI

Script & Interactive

Cake

> dotnet add package MongoDB.EntityFrameworkCore --version 9.0.0

Copy

README

Frameworks

Dependencies

Used By

Versions

Release Notes

## MongoDB Entity Framework Core Provider

nuget v9.0.0

The MongoDB EF Core Provider requires Entity Framework Core 8 or 9 on .NET 8 or later and a MongoDB database server 5.0 or later, preferably in a transaction-enabled configuration.



# MongoDB

---

**MongoDB** - это документоориентированное NoSQL-хранилище, предназначенное для хранения, обработки и управления большими объемами неструктурированных или полуструктурированных данных.

В отличие от традиционных реляционных БД хранит данные в формате, близком к JSON (Binary JavaScript Object Notation), что делает её гибкой и удобной для работы с современными приложениями.

# Основные особенности MongoDB

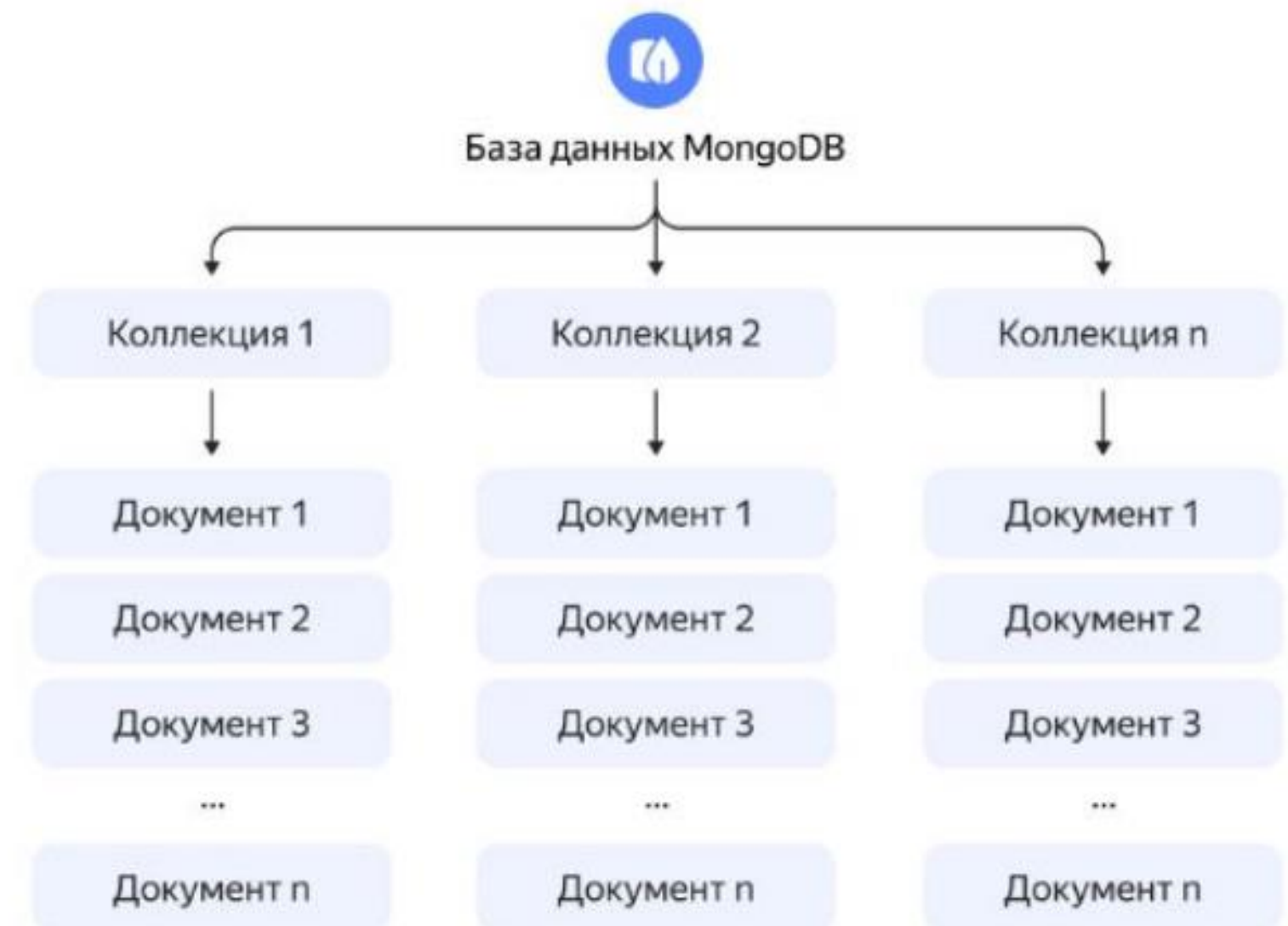
---

- Документная модель
- Гибкость схемы
- Масштабируемость
- Высокая производительность
- Поддержка сложных запросов
- Репликация и отказоустойчивость
- Поддержка различных типов данных
- Интеграция с современными технологиями

# Схема данных MongoDB

Основные компоненты MongoDB:

- **Кластер:** Группа серверов, которые хранят и обрабатывают данные
- **База данных:** Контейнер для коллекций
- **Коллекция:** Группа документов (аналог таблицы в SQL)
- **Документ:** Основная единица хранения данных (аналог строки в SQL)





# EF Core и MongoDB

---

- **EF Core:** ORM для реляционных БД. Основные задачи: маппинг объектов на таблицы, LINQ, миграции, транзакции.
- **MongoDB:** Документоориентированная NoSQL. Сильные стороны: гибкая схема, горизонтальное масштабирование, агрегации.

Парадигмы в конфликте:

- ORM vs документная модель (таблицы <> коллекции, JOIN <> вложенные документы).
- EF Core накладывает реляционное мышление на NoSQL

# Зачем использовать EF Core с MongoDB?

---

- Унификация кода в гибридных проектах (SQL + MongoDB).
- Снижение порога входа для команд, знающих EF Core.
- Удобство LINQ и автоматического маппинг

---

# Обзор MongoDB Provider for EF Core

---

# What's New

Learn what's new in:

- [Version 9.0](#)
- [Version 8.3](#)
- [Version 8.2.3](#)
- [Version 8.2.2](#)
- [Version 8.2.1](#)
- [Version 8.2](#)
- [Version 8.1](#)

## Октябрь 2023 г.

Первый анонс и начало интеграции: на встрече сообщества .NET Data анонсирована работа над провайдером MongoDB для EF Core.

## Ноябрь 2023 г.

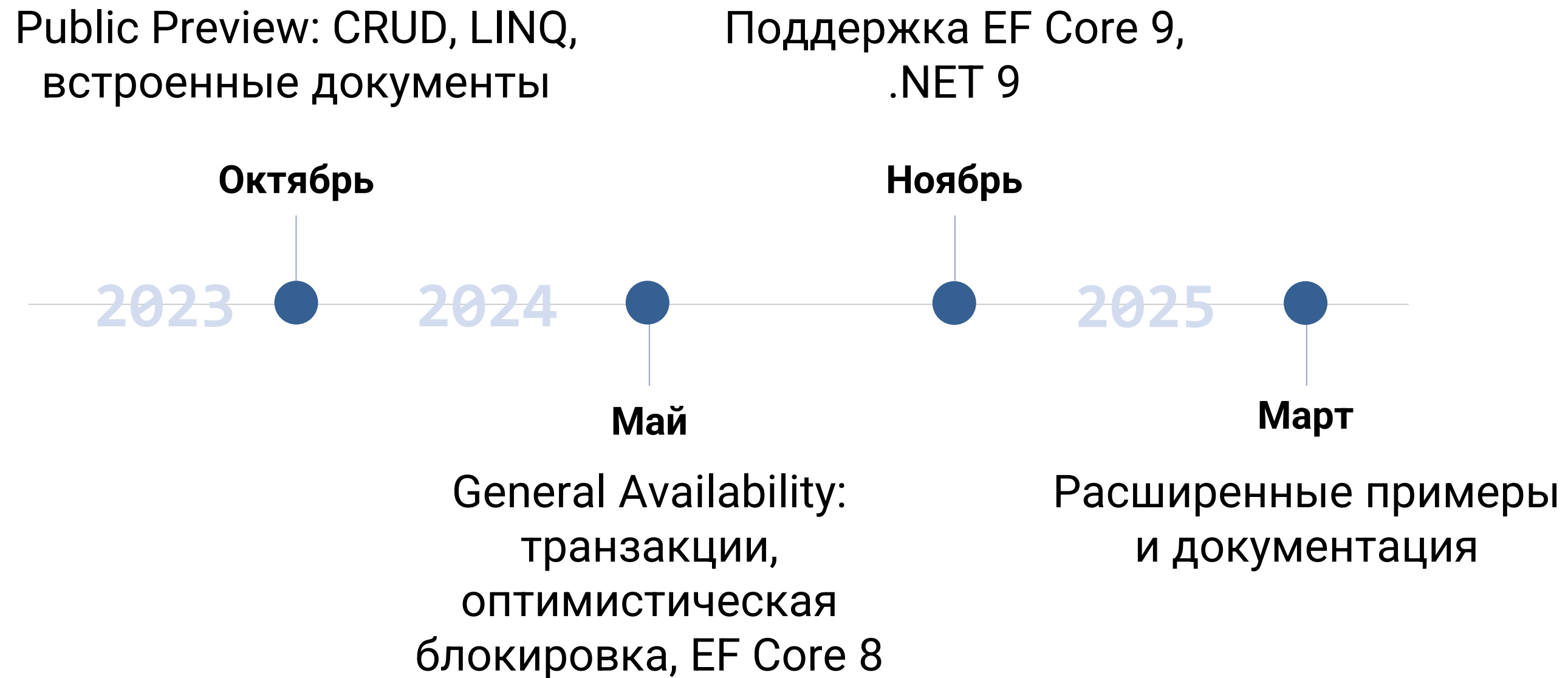
В EF Core 8.0 добавлена экспериментальная поддержка MongoDB (preview version).

## Май 2024 г.

Релиз GA (версия 8.1.0)  
Провайдер MongoDB для EF Core официально выпущен (General Availability).

# Основные этапы разработки

---





# MongoDB Provider for EF Core

---

## Ключевые возможности провайдера:

- *Совместимость*: Работает с EF Core 8 и .NET 8.
- *Расширенные запросы*: Поддержка сложных операций (Where, OrderBy, ThenBy) и агрегаций.
- *Гибкость маппинга*: Настройка типов данных, составных ключей, вложенных документов.
- *Работа с массивами/списками*: Упрощённое управление сложными структурами данных.
- *Логирование*: Улучшенная видимость операций.

# MongoDB Provider for EF Core

---

## Ограничения:

- Нет транзакций в шардированных кластерах.
- Частичная поддержка агрегаций (только через LINQ).
- Недоступны специфичные типы данных (например, GeoJSON).

# MongoDB Provider for EF Core

---

## Как работает:

- Наследование от DbContext, регистрация коллекций и индексов через Fluent API.
- Нет миграций — схема управляется кодом приложения.
- Сущности мапятся в BSON-документы (включая вложенные объекты и коллекции).
- Поддержка LINQ с ограничениями (например, не все методы IQueryable).

# Подключение провайдера

---

Установите NuGet-пакет:

```
dotnet add package MongoDB.EntityFrameworkCore
```

Подключите зависимости:

```
using Microsoft.EntityFrameworkCore;  
using MongoDB.EntityFrameworkCore.Extensions;
```

# Подключение провайдера

---

Создайте *DbContext*:

```
public class AppDbContext : DbContext
{
    public DbSet<User> Users { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseMongoDB("mongodb://localhost:27017", "YourDatabaseName");
    }
}
```

# Подключение провайдера

---

Настройка маппинга через Fluent API:

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<User>().ToCollection("users");
    modelBuilder.Entity<User>().Property(u => u.Id).HasElementName("_id");
}
```

# Логирование запросов

---

Включите логирование для отладки MQL-запросов:

```
optionsBuilder.UseMongoDB(connectionString, databaseName)
    .EnableSensitiveDataLogging()
    .LogTo(Console.WriteLine, LogLevel.Information);
```



# CRUD-операции

```
using var context = new AppDbContext();  
var user = new User { Name = "Alice",  
    Orders = new List<Order> { new Order  
    { Product = "Book", Price = 20 } } };  
context.Users.Add(user);  
await context.SaveChangesAsync();
```

```
var user = await context.Users  
    .FirstAsync(u => u.Name == "Alice");  
user.Name = "Bob";  
await context.SaveChangesAsync();
```

```
context.Users.Remove(user);  
await context.SaveChangesAsync();
```

```
var users = await context.Users  
    .Where(u => u.Orders.Any(o =>  
        o.Price > 10))  
    .OrderBy(u => u.Name)  
    .ToListAsync();
```

# Как работает `SaveChangesAsync` для MongoDB?

---

После вызова метода `SaveChangesAsync()` провайдер выполняет следующую последовательность действий:

- Собирает все изменения из `ChangeTracker`.
- Преобразует их в операции MongoDB.
- Открывает сессию и транзакцию (если возможно).
- Выполняет операции через MongoDB .NET Driver.
- Фиксирует или откатывает транзакцию.

# Как работает SaveChangesAsync для MongoDB?

---

- ChangeTracker
  - При работе с DbContext провайдер автоматически отслеживает изменения в сущностях (добавление, изменение, удаление).
  - Для каждой сущности определяется состояние (Added, Modified, Deleted), как в классическом EF Core.

# Как работает SaveChangesAsync для MongoDB?

---

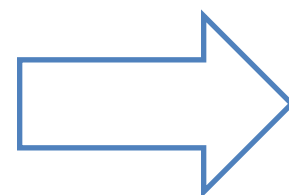
- Генерация операций MongoDB
- Провайдер преобразует изменения в операции MongoDB:
  - Добавление (**Added**) → *InsertOneAsync*.
  - Изменение (**Modified**) → *ReplaceOneAsync*  
(или *UpdateOneAsync* для частичного обновления).
  - Удаление (**Deleted**) → *DeleteOneAsync*.

# Как работает SaveChangesAsync для MongoDB?

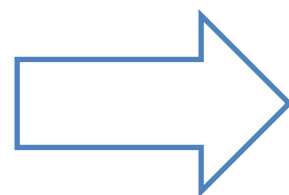
---

- Генерация операций MongoDB (Пример)
  - Изменение (**Modified**) → *ReplaceOneAsync*

```
var user = await context.Users
    .FirstAsync(u => u.Name == "Alice");
user.Name = "Bob";
await context.SaveChangesAsync();
```



```
await collection.ReplaceOneAsync(
    filter: Builders<User>.Filter.Eq(u => u.Id, user.Id),
    replacement: user
);
```



# Как работает SaveChangesAsync для MongoDB?

---

- Сессии и транзакции
  - По умолчанию SaveChangesAsync использует неявную транзакцию для всех операций в рамках одного вызова.
  - Транзакции работают через сессии MongoDB.
  - Если транзакция не поддерживается, провайдер выполнит операции без неё.

# Как работает SaveChangesAsync для MongoDB?

---

- Сессии и транзакции (Пример)

```
public async Task<int> SaveChangesAsync()
{
    using var session = await _client.StartSessionAsync();
    session.StartTransaction();
    try
    {
        foreach (var entry in _changeTracker.Entries())
        {
            // Генерация и выполнение операций MongoDB
        }
        await session.CommitTransactionAsync();
    }
    catch
    {
        await session.AbortTransactionAsync();
        throw;
    }
}
```



# Как работает SaveChangesAsync для MongoDB?

---

- Оптимистическая блокировка
- При обновлении провайдер автоматически добавляет фильтр по версии (если в модели есть свойство с атрибутом [Timestamp] ):

```
await collection.ReplaceOneAsync(  
    filter: Builders<User>.Filter.And(  
        Builders<User>.Filter.Eq(  
            u => u.Id, user.Id),  
        Builders<User>.Filter.Eq(  
            u => u.Version, user.Version)  
        ),  
    replacement: user );
```

```
public class User {  
    [Timestamp]  
    public long? Version  
        { get; set; }  
}
```

# Особенности работы и ограничения

---

- Пакетная обработка
  - Все операции в рамках `SaveChangesAsync` выполняются последовательно, а не батчем (в отличие от SQL-провайдеров).
- ! MongoDB не поддерживает батчинг команд в рамках одной транзакции.

# Особенности работы и ограничения

---

Change Tracking в MongoDB EF Core Provider работает через сравнение снимков состояний сущностей, что позволяет:

- Определять изменения на уровне полей
- Генерировать оптимальные MongoDB-запросы
- Поддерживать базовые сценарии CRUD

**!** Из-за особенностей документной модели (отсутствие JOIN, транзакций по умолчанию) некоторые возможности EF Core (например, каскадные обновления) могут быть недоступны или требовать ручной реализации.

# Когда использовать `SaveChangesAsync`?

---

- Для атомарного сохранения группы изменений в рамках одного документа.
- Если нужны ACID-гарантии для кластера MongoDB с репликацией.
- Для автоматического управления версиями документов.

# Когда следует избегать?

---

- Избегайте частых вызовов `SaveChangesAsync` для отдельных операций — это может снизить производительность.
- Для сложных сценариев (например, обновление вложенных массивов) используйте `MongoDB.Driver` напрямую.

# Особенности работы и ограничения

---

## Транзакции:

Поддерживаются только для одного документа (ACID-гарантии на уровне документа).

```
using var transaction = await context.Database.BeginTransactionAsync();
try
{
    // Операции...
    await transaction.CommitAsync();
}
catch
{
    await transaction.RollbackAsync();
}
```

# Особенности работы и ограничения

---

## Настройка поведения транзакций:

Транзакции можно отключить, если это требуется.

```
db.Database.AutoTransactionBehavior = AutoTransactionBehavior.Never;
```

Может быть полезно для сценариев, где транзакции не нужны или создают избыточную нагрузку.



# Особенности работы и ограничения

---

## Ограничения

- *В ранних версиях нет распределенных транзакций:*  
Транзакции работают только в пределах одного кластера MongoDB.
- *Замена документов:*  
По умолчанию используется ReplaceOneAsync, что может привести к перезаписи всего документа (даже если изменилось одно поле).

# Особенности работы и ограничения

---

## Ограничения

- *В ранних версиях нет распределенных транзакций:*  
Транзакции работают только в пределах одного кластера MongoDB.
- *Замена документов:*  
По умолчанию используется ReplaceOneAsync, что может привести к перезаписи всего документа (даже если изменилось одно поле).

***Решение:** Настройте частичное обновление через Fluent API*

# Особенности работы и ограничения

---

## Индексы:

Для базовых сценариев можно добавлять индексы через атрибут [Index] непосредственно в классе сущности.

```
using MongoDB.Entities;
using MongoDB.Driver;

[Index("Name", Name = "idx_name")]
public class Product
{
    public string Id { get; set; }
    public string Name { get; set; }
    public decimal Price { get; set; }
}
```

# Особенности работы и ограничения

---

## Индексы:

Составные или уникальные индексы можно настроить через Fluent API в методе OnModelCreating настроить.

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Product>().HasIndex(p => new { p.Name, p.Price })
        .HasName("idx_name_price")
        .IsUnique();
}
```

# Особенности работы и ограничения

---

## Индексы:

Если провайдер EF Core не поддерживает нужные типы индексов, используйте MongoDB.Driver для создания индексов напрямую

```
var collection = database.GetCollection<User>("users");  
var indexKeys = Builders<User>.IndexKeys.Ascending(u => u.Name);  
await collection.Indexes.CreateOneAsync(new CreateIndexModel<User>(indexKeys));
```

# Особенности работы и ограничения

---

## Индексы:

Для сложных типов индексов (например, текстовых или 2dsphere) используйте MongoDB.Driver:

```
// Текстовый индекс для поиска по тексту
var textKeys = Builders<Product>.IndexKeys.Text(p => p.Name);
await collection.Indexes.CreateOneAsync(new CreateIndexModel<Product>(textKeys));

// Геопространственный индекс
var geoKeys = Builders<Store>.IndexKeys.Geo2DSphere(s => s.Location);
await collection.Indexes.CreateOneAsync(new CreateIndexModel<Store>(geoKeys));
```

# Чего в целом удалось добиться?

---

Удалось реализовать:

- MongoDBContext и MongoDBSet для работы с коллекциями MongoDB, как DbContext и DbSet в EF Core
- Отслеживание изменений (Change Tracking)
- Поддержка LINQ-запросов, включая AsNoTracking()
- Возможность группировки операций (вставка, обновление, удаление) в единый BulkWrite-запрос к MongoDB
- Использование атрибутов для настройки маппинга
- Поддержка создания индексов
- Entity Buckets (Группировка документов)
- Профилирование запросов к MongoDB



# Дальнейшие планы

---

Планы развития:


- GridFS: Поддержка работы с большими файлами.
- Транзакции: Полная интеграция с транзакциями MongoDB.
- Fluent API: Альтернатива атрибутам для настройки маппинга.
- Анализатор C# для MongoDB: Интеграция для улучшения проверки запросов.
- Время выполнения запросов: Оптимизация производительности без существенных накладных расходов.

---

# Сравнение с MongoDB.Driver

---

# Сравнение с MongoDB.Driver

 **MongoDB.Driver** 3.3.0

Prefix Reserved

.NET 6.0

.NET Standard 2.1

.NET Framework 4.7.2

.NET CLI

Package Manager

PackageReference

Central Package Management

Paket CLI

Script & Interactive

Cake

> dotnet add package MongoDB.Driver --version 3.3.0

Copy

README

Frameworks

Dependencies

Used By

Versions

Release Notes

## MongoDB C# Driver

You can get the latest stable release from the [official Nuget.org feed](#) or from our [github releases page](#).

**Downloads** [Full stats →](#)

Total 272.8M

Current version 316.1K

Per day average 53.5K

**About**

Last updated a month ago

[Project website](#)

[Source repository](#)

[Apache-2.0 license](#)

[Download package](#) (2.83 MB)

# Сравнение с MongoDB.Driver

---

## Гибкость:

- Driver: Полный доступ к агрегациям, транзакциям, Change Streams.
- EF Core: Только простые сценарии (нет \$lookup, \$geoNear).

## Кодовая база:

- EF Core: Меньше boilerplate, но ограничения в запросах.
- Driver : Больше контроля, подходит для DDD-проектов с кастомными репозиториями.

# Сравнение с MongoDB.Driver

---

## **Производительность (BenchmarkDotNet):**

- Вставка: MongoDB.Driver быстрее на 15-20%.
- Чтение: EF Core проигрывает из-за накладных расходов на маппинг.
- Сложные запросы: Драйвер выигрывает за счет прямого использования агрегаций.

## MongoDB.Driver

```
await collection.InsertOneAsync(product);  
var result = await collection.Find(p => p.Id == id)  
    .FirstOrDefaultAsync();
```

## MongoDB EF Core Provider

```
await context.Products.AddAsync(product);  
var result = await context.Products  
    .FirstOrDefaultAsync(p => p.Id == id);
```

# Недостатки MongoDB EF Core Provider

---

- Ограниченная поддержка возможностей MongoDB
- Низкая производительность в сложных сценариях
- Отсутствие полной совместимости
- Дополнительный слой абстракции

# Когда использовать MongoDB Driver

---

Официальный драйвер MongoDB.Driver предпочтителен, если:

- Нужен полный контроль над запросами и операциями.
- Требуются специфичные для MongoDB функции:
  - Агрегации с сложными пайплайнами.
  - Текстовый поиск, гео-индексы.
  - Работа с GridFS (хранение файлов).
- Оптимизация производительности критична (например, для высоконагруженных систем).
- Используются транзакции (в MongoDB 4.0+).



# Гибридный подход

Оптимальным решением может быть комбинация обоих подходов:

```
private readonly AppDbContext _context;
private readonly IMongoCollection<Product> _collection;

public ProductService(AppDbContext context, IMongoDatabase database)
{
    _context = context;
    _collection = database.GetCollection<Product>("Products");
}

// Использование EF Core для простых операций
public async Task UpdatePriceAsync(string id, decimal price)
{
    var product = await _context.Products.FindAsync(id);
    product.Price = price;
    await _context.SaveChangesAsync();
}


// Использование MongoDB Driver для агрегаций
public async Task<List<Product>> GetTopExpensiveProductsAsync(int limit)
{
    return await _collection.Aggregate().SortByDescending(p => p.Price).Limit(limit).ToListAsync();
}
```

---

# MongoDB.Entities

---

# MongoDB.Entities

 **MongoDB.Entities** 24.1.1

[.NET Standard 2.1](#)

[.NET CLI](#) [Package Manager](#) [PackageReference](#) [Central Package Management](#) [Paket CLI](#) [Script & Interactive](#) [Cake](#)

```
> dotnet add package MongoDB.Entities --version 24.1.1
```

[Copy](#)

[README](#) [Frameworks](#) [Dependencies](#) [Used By](#) [Versions](#)

[license](#) MIT [version](#) v24.1.1 [downloads](#) 907k [tests](#) 791 passed [discord](#) 11 online

## MongoDB.Entities

A light-weight .net standard library with barely any overhead that aims to simplify access to mongodb by abstracting the official driver while adding useful features on top of it resulting in an elegant API surface which produces beautiful, human friendly data access code.

### Downloads

[Full stats →](#)

Total **906.7K**

Current version **12.1K**

Per day average **412**

### About

🕒 Last updated 2 months ago

🌐 [Project website](#)

💎 [Source repository](#)

📄 [MIT license](#)

📦 [Download package](#) (129.04 KB)

📦 [Download symbols](#) (32.25 KB)

🔗 [Open in NuGet Package Explorer](#)

# MongoDB.Entities

---

**MongoDB.Entities** - библиотека для упрощения работы с MongoDB в .NET-приложениях, предоставляя высокоуровневый API, похожий на ORM (Object-Relational Mapping), но адаптированный для документоориентированной СУБД.

Она позволяет работать с данными через объекты и LINQ-запросы, минимизируя ручной маппинг и низкоуровневые операции.



# MongoDB.Entities

---

## Для чего нужна:

- Упрощает взаимодействие с MongoDB через объектно-ориентированный подход.
- Позволяет использовать LINQ для запросов, автоматический маппинг сущностей, репозитории и дополнительные абстракции.
- Полезна для разработчиков, которые хотят избежать ручной работы с BSON-документами (как в MongoDB.Driver), но не хотят использовать Entity Framework.

# MongoDB.Entities

---

## Особенности:

- Настройка над MongoDB.Driver.
- Поддержка отношений между сущностями (в отличие от стандартного драйвера).
- Встроенные методы для CRUD, пагинации, каскадных операций.
- Отсутствие миграций (как в реляционных БД), так как MongoDB схемонезависима.

# MongoDB.Entities

---

**MongoDB.Entities** - библиотека для упрощения работы с MongoDB в .NET-приложениях, предоставляя высокоуровневый API, похожий на ORM (Object-Relational Mapping), но адаптированный для документоориентированной СУБД.

Она позволяет работать с данными через объекты и LINQ-запросы, минимизируя ручной маппинг и низкоуровневые операции.



# Настройка подключения и модели

```
using MongoDB.Entities;
using System;

// Определение сущности
public class Book : Entity
{
    public string Title { get; set; }
    public string Author { get; set; }
    public int Year { get; set; }
    // Связь "один ко многим"
    public Many<Review> Reviews { get; set; }

    public Book() => this.InitOneToMany(() => Reviews);
}

public class Review : Entity
{
    public string Text { get; set; }
    public int Rating { get; set; }
}

// Инициализация подключения
await DB.InitAsync("BookStore", "localhost", 27017);
```

## Настройка маппинга:

```
DB.Entity<Book>(entity =>
{
    entity.CollectionName("Books"); // Название коллекции
    entity.Property(b =>
b.Title).HasColumnName("book_title");
});
```

## Пагинация и сортировка:

```
var (results, totalCount) = await DB.Paged<Book>(
    page: 1,
    pageSize: 10,
    sort: b => b.Year,
    order: Order.Descending
);
```



# CRUD-операции

```
var book = new Book
{
    Title = "Война и мир",
    Author = "Лев Толстой",
    Year = 1869
};

await book.SaveAsync(); // Вставка документа
```

```
var review =
    new Review { Text = "Отличная книга!", Rating = 5 };
await review.SaveAsync();

book.Reviews.Add(review); // Связывание отзыва с книгой
```

```
await DB.Update<Book>()
    .Match(b => b.ID == book.ID)
    .Modify(b => b.Year, 1870)
    .ExecuteAsync();
```

```
await DB.DeleteAsync<Book>(book.ID);
```

```
var foundBook = await DB.Find<Book>().OneAsync(book.ID);
Console.WriteLine(foundBook.Title); // "Война и мир"
```

```
// Поиск книг, выпущенных после 1900 года
var books = await DB.Queryable<Book>()
    .Where(b => b.Year > 1900)
    .ToListAsync();
```

```
// Поиск с проекцией (только заголовки и автора)
var projected = await DB.Find<Book>()
    .Project(b => new { b.Title, b.Author })
    .ExecuteAsync();
```

# MongoDB.Entities

---

MongoDB.Entities позволяет работать с MongoDB через объекты и LINQ, избегая ручного маппинга BSON.

Библиотека позволяет реализовать:

- Создание связей между сущностями.
- Использование асинхронных методов (SaveAsync, FindAsync).
- Гибкие запросы с LINQ и Fluent API.
- Управление коллекциями и проекциями.

Для сложных сценариев можно использовать низкоуровневые методы через `DB.Database<T>`, которые предоставляют доступ к `MongoDB.Driver`.

# Когда что выбрать?

MongoDB.Driver	MongoDB.Entities	MongoDB.EntityFrameworkCore
<ul style="list-style-type: none"><li>• Полный контроль над запросами и оптимизациями</li><li>• Сложные агрегации, кастомные индексы, работа с GridFS</li><li>• Высоконагруженные приложения, где важна производительность</li></ul>	<ul style="list-style-type: none"><li>• Быстрое развитие проекта с минимумом кода</li><li>• Удобство работы с объектами и LINQ без привязки к EF</li><li>• Простые CRUD-операции и связи между сущностями</li></ul>	<ul style="list-style-type: none"><li>• Если проект уже использует EF Core</li><li>• Необходимость единого кода для работы с разными БД (например, MongoDB и SQL)</li><li>• Привычный синтаксис EF</li></ul>

# Сравнение библиотек

Критерий	MongoDB.Entities	MongoDB.Driver	MongoDB EFCore Provider
Уровень абстракции	Высокий (ORM-подобный слой)	Низкий (нативный доступ к MongoDB API)	Высокий (интеграция с EF Core)
Синтаксис запросов	LINQ и Fluent-API	BSON-документы, Builders<T>	LINQ (как в EF Core для реляционных БД)
Маппинг сущностей	Автоматический через атрибуты/Fluent-конфигурацию	Ручной (например, BsonClassMap)	Через DbContext и modelBuilder
Управление индексами	Через Fluent-API или низкоуровневые методы	Прямое через IndexKeysDefinition<T>	Не поддерживается через HasIndex; требуется MongoClient
Транзакции	Поддерживаются (через MongoDB.Driver)	Полная поддержка	Поддержка с версии 8.1.0 (автотранзакции)
Изменение схемы	Динамическая схема (без миграций)	Гибкость схемы MongoDB	Code First с частичной поддержкой миграций
Производительность	Накладные расходы из-за абстракций	Максимальная	Умеренные накладные расходы
Зависимости	Сторонняя библиотека (надстройка над Driver)	Официальный драйвер MongoDB	Требует EF Core и MongoDB.Driver
Подходящие сценарии	Быстрое прототипирование, простые CRUD	Сложные запросы, кастомные оптимизации	Проекты с EF Core, кроссплатформенные решения

---

# Рекомендации и выводы

---

# Кейсы использования EF Core Provider

---

Когда стоит использовать EF Core Provider:

---

- Проекты с гибридными БД (часть данных в SQL, часть в MongoDB)
- Команды, уже использующие EF Core для реляционных БД
- Простые CRUD-сценарии без сложных запросов

Когда лучше отказаться:

---

- Высоконагруженные системы с требованиями к низкой задержке
- Использование специфичных функций MongoDB (например, [Change Streams](#)).

# Рекомендации и выводы

---

- **Чек-лист для принятия решения:**

требования проекта, навыки команды.

- **Как минимизировать риски:**

кастомизация провайдера, гибридный подход (EF Core + MongoDB.Driver).

- **Ответ на главный вопрос:**

*«Стоит ли?»* — Да, но только в конкретных сценариях.

*Нет, если вам нужна вся мощь MongoDB.*

# Примеры кейсов

---

Два примера из жизни, в которых применим MongoDB Provider:

- Кейс 1 - Сервис справочников для сложного классификатора лекарственных средств
- Кейс 2 - Сервис машиночитаемых доверенностей со сложной доменной сущностью



# Кейс1 – Сервис справочников

---

Сервис справочников работает со сложным классификатором лекарственных средств.

[Федеральный закон от 12.04.2010 N 61-ФЗ \(ред. от 26.12.2024\) "Об обращении лекарственных средств" \(с изм. и доп., вступ. в силу с 01.03.2025\)](#)

Для хранения изначально используется MongoDB (по средствам MongoDB.Driver), для других справочников используется EF Core.

Для поиска используется стандартный набор CRUD-операций.

MongoDB Provider позволяет привести все справочники к общему интерфейсу взаимодействия по средствам EF Core.

# Кейс2 – Сервис МЧД

Версия схемы: 03

📄 — Описание XML-схемы

О Доверенность / Сведения о доверенности

ИнСвед / Иные сведения

О Документ / Состав и структура документа

УО Довер / Доверенность

О СвДов / Сведения о доверенности

Тип: СвДовТип

М КодНОДейст / Код налогового органа, в отношении которого действует доверенность

О СведСист / Сведения об информационной системе, которая предоставляет техническую возможность получения информации о доверенности, досрочном прекращении действия доверенности, в том числе в силу ее отмены доверителем

Базовый тип: строка

Максимальное количество символов: 1000

Минимальное количество символов: 1

Безотзыв / Сведения о безотзывной доверенности

АО ВидДовер / Вид доверенности

АО ПрПередов / Признак возможности оформления передоверия

АО ВнНомДовер / Внутренний номер доверенности

АО НомДовер / Единый регистрационный номер доверенности

А НомРНДДовер / Регистрационный номер доверенности в реестре нотариальных действий

А ДопИдДовер / Дополнительный идентификатор доверенности

А ДатаВнРегДовер / Дата внутренней регистрации доверенности

АО ДатаВидДовер / Дата совершения (выдачи) доверенности

АО СрокДейст / Срок действия доверенности

Сервис со сложной доменной сущностью, с глубокой вложенностью на PostgreSQL (работа по средствам EF Core).

## Требования информационных систем к форматам МЧД XML-схема для формирования электронного документа

Сервис требует постоянного анализа запросов и, из-за большого количества JOIN-ов, оптимизаций по средствам SplitQuery и т.п.

MongoDB Provider позволяет протестировать аналогичную логику работы на MongoDB.

# Полезные ссылки

---

- [What's New in EF Core 10](#)
- [MongoDB C# Driver \[GitHub\]](#)
- [Создание Web API приложения с использованием .NET Core + MongoDB .NET Driver](#)
- [MongoDB.Entities \[GitHub\]](#)
- [MongoDB Entity Framework Core Provider \[GitHub\]](#)
- [MongoDB EF Core Provider: What's New?](#)
- [MongoDB Provider for EF Core: The Latest Updates](#)
- [Migrating From PostgreSQL to MongoDB in a .NET EF Core Application \[GitHub\]](#)



# Мои контакты

➤ Telegram:

- Мой канал: [@dzitskiy\\_dev](https://t.me/dzitskiy_dev)
- Группа по .NET: [@dzitskiy\\_net](https://t.me/dzitskiy_net), [@sevdotnet](https://t.me/sevdotnet)



@DZITSKIY\_DEV



@DZITSKIY\_NET



@SEVDOTNET

**Спасибо!**