

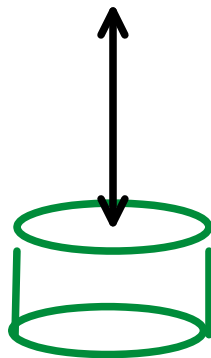
Multi-Tier-Architektur

Mittwoch, 8. Februar 2023 08:45

UI

BL

DAL

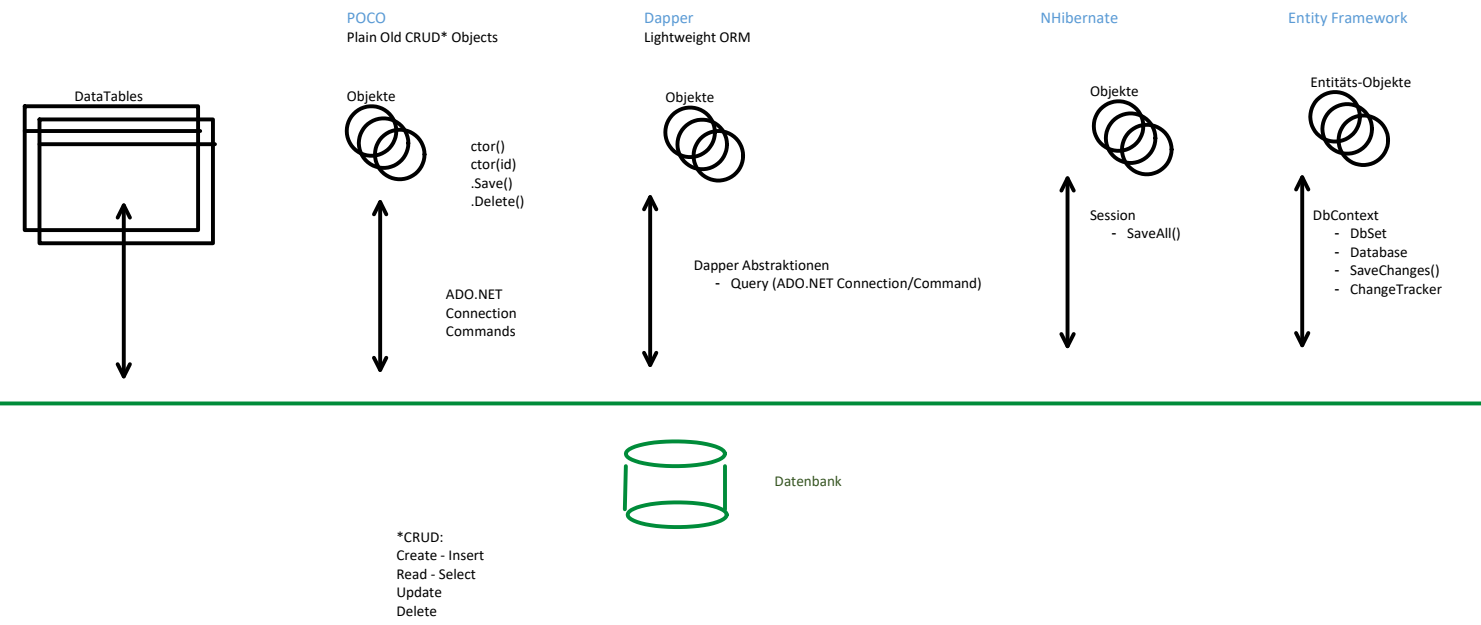


Datenbank

ADO.NET - DataTables
ADO.NET - Connections, Commands
Entity Framework

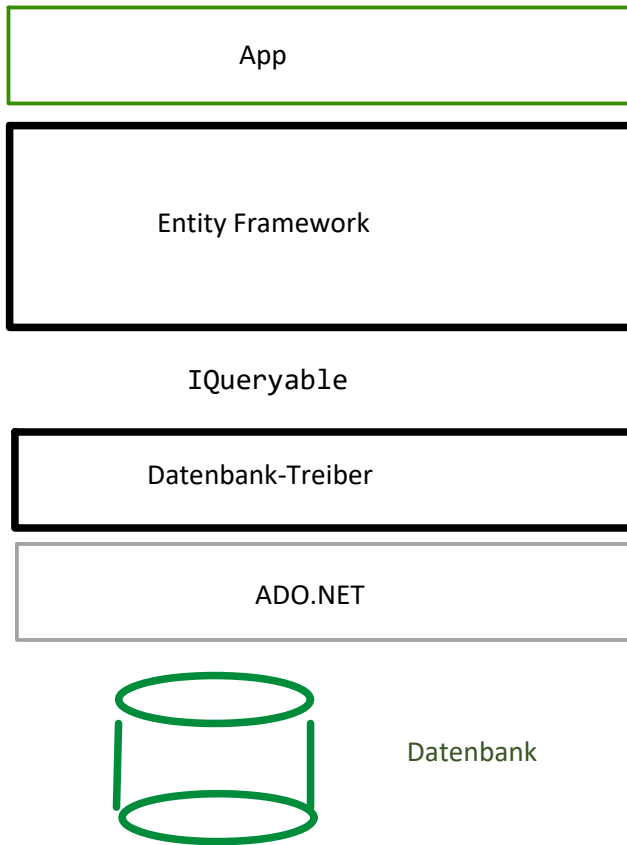
Data-Access-Layer

Mittwoch, 8. Februar 2023 09:35



Entity Framework Architektur

Mittwoch, 8. Februar 2023 09:59



Nuget

Microsoft.EntityFrameworkCore

Microsoft.EntityFrameworkCore.Tools

Microsoft.EntityFrameworkCore.Sqlite

Microsoft.EntityFrameworkCore.SqlServer

Npgsql.EntityFrameworkCore.PostgreSQL

Ausflug: .NET Framework vs. .NET (Core)

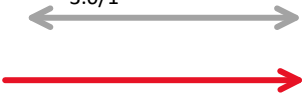
Mittwoch, 8. Februar 2023 10:36

.NET Framework (".NET Classic")

Jahr	Version	Bemerkungen
2000	1.0	
2001	1.1	Visual Studio .NET, ASP.NET (IIS)
2003	2.0	~650 Namespaces
2007	3.0	WPF, LINQ, Entity Framework
2008	3.5	WPF, LINQ, Entity Framework
2010	4.0	TPL (async/await), HTTP
2013	4.5	Roslyn-Compiler, ~14.000 Namespaces
2018	4.8.1	"Final Version"

.NET Standard

1.0
2.0
2.1
3.0/1



.NET Core

- plattformunabhängig
- leichtgewichtig, modular, RESTfull
- Open Source

Jahr	Version	Bemerkungen
2016	1.0	~850 Namespaces, Nuget, keine UI
	2.0	reines Backend
2019	3.0	WPF, Windows Forms - Open Source
2020	5.0	.NET Core
2021	6.0	Long Term Support (LTS)
2022	7.0	

Entity Framework Core

Mittwoch, 8. Februar 2023 13:59

Reverse Engineering (formerly known as Database First)

Datenbank existiert bereits, es soll ein Objektmodell für den Datenzugriff erstellt werden.

```
Scaffold-DbContext
-Connection "datasource=C:\ProgramData\SQLite\data\northwind.db"
-Provider Microsoft.EntityFrameworkCore.Sqlite
-OutputDir Model
-Context NorthwindContext
-Tables Customers, Orders, "Order Details", Products
```

Code First

Es wird eine Datenbank (automatisch?) generiert, die dem Objektmodell entspricht

Nullables in C#

Mittwoch, 8. Februar 2023 14:18

```
1  int a = 0;  
2  int? a2 = null;  
3  System.Nullable<int> a3 = null;  
4  
5  int b = a + (a2.HasValue ? a2.Value : 0);  
6  int c = a + a2 ?? 0;
```

Unit Tests in Visual Studio

Mittwoch, 8. Februar 2023 15:10

Funktionen einer Klassenbibliothek gezielt aufrufen und ausführen

Vorgehen

- Projekt hinzufügen, Test-Vorlage (MSTest, Nunit, xUnit)
- Als [TestMethod] gekennzeichnete Methoden tauchen im Test-Explorer auf (Test/TestExplorer)
- Testexplorer: Tests einzeln, alle, oder Auswahl ausführen
- Statische Assert stellt Methoden bereit, das Testergebnis zu definieren

Unit Tests debuggen

Test als Debug starten

Context benutzen

Mittwoch, 8. Februar 2023 15:18

Zugriff auf die Datenbank:

- Instanz von DbContext
- DbSet des Contextes stellen die Daten bereit

Language INtegrated Query

Mittwoch, 8. Februar 2023 15:25

SQL

```
1 SELECT * FROM Customers WHERE City LIKE 'Berlin%'
```

C# ohne LINQ

```
1 List<Customer> allCustomers = GetAllCustomers();
2
3 List<Customer> berliners = GetCustomersByCity(allCustomers, "Berlin");
4
5
6 private List<Customer> GetCustomersByCity(List<Customer> customers, string city)
7 {
8     foreach (Customer cu in customers)
9     {
10         //...
11     }
12 }
```

C# mit LINQ

```
1 List<Customer> allCustomers = GetAllCustomers();
2
3 // Deklarative Syntax
4 var berliners1 = from cu in allCustomers
5                  where cu.City.StartsWith("Berlin")
6                  select new { cu.Vorname, cu.Nachname };
7
8 // Lambda-Syntax
9 var berliners2 = allCustomers.Where(cu => cu.City.StartsWith("Berlin"));
10
11 List<Customer> berls = berliners1.ToList();
```

Deferred Execution

Zugriff auf die Ergebnismenge führt Query (GetEnumerator()) aus
IEnumerable/IQueryable aus

Wünsche

Mittwoch, 8. Februar 2023 16:08

- RawSql

DbContext - Datenoperationen

Donnerstag, 9. Februar 2023 16:30

Lesen

- > LINQ auf DbSet
- LINQ wird analysiert, daraus SQL gebaut

```
context.Customers.Where/Select/FirstOrDefault
```

Lazy Loading

- > DbContext lädt nur die allernotwendigsten Daten

```
context.Customers.Include(...)  
    .Include(...) // Customer  
    .ThenInclude(...) // Cascadierend
```

Zu ladende Daten im LINQ-Statement verwenden

Performance/Tweaking

- `.AsNoTracking` - laden ohne Tracking (ChangeTracker)
- `.AsEnumerable` - Query ausführen, nachfolgende Operationen im RAM

Context-Handling

Context sollte so sparsam wie möglich einsetzen!

- Context bevorzugt lokal instanziiieren
- Tipp: using
- Tipp: Tracking nur dann, wenn notwendig (Speichern geplant)

Schreiben

Nur eine Methode, um in die Datenbank zu schreiben:

```
context.SaveChanges();
```

Änderungen werden erkannt mit Hilfe des ChangeTrackers

- `context.Entry(...).State` - `EntityState`-Enum
lässt sich auch per Code setzen, empfehlenswert ist jedoch passende Methode verwenden:
`context.Customers.Attach(...)` statt `context.Entry(...).State = Modified`
- `SaveChanges` ändert den State zurück auf `Unchanged`
- `context.Entry(...).Reload()` - Daten neu laden, State = `Unchanged`