C# Fortgeschrittenenschulung

BONUS - Einführung Entity Framework

Einführung in Entity Framework Core

Überblick über ORMs

- **ORMs (Object-Relational Mappers):**
  - Werkzeuge, die programmiersprachenspezifische Objekte auf Relationen in Datenbanken abbilden.
  - Sie erleichtern die Datenbank-Interaktion durch das Einfügen, Lesen, Aktualisieren und Löschen (CRUD) von Objekten.

EF Core im Überblick

- **Entity Framework Core (EF Core):**
  - Eine leichte, erweiterbare und plattformübergreifende Version von Entity Framework.
  - Unterstützt LINQ-Abfragen, Änderungsverfolgung, Updates, Schema-Migrationen und mehr.

Alternative ORM-Mapper für .NET

Beliebte ORM-Alternativen

- **Dapper:**
  - Ein leichtgewichtiger Micro-ORM.
  - Schnell und effizient für einfache SQL-Abfragen.
  - Minimaler Abstraktionsaufwand mit direktem SQL-Zugriff.
- **NHibernate:**
  - Leistungsstarker und ausgereifter ORM.
  - Stark konfigurierbar mit einem Fokus auf komplexe Datenbankmappings und Transaktionen.
  - Unterstützung für fortgeschrittene Szenarien mit Caching, Batch-Processing und mehr.

Auswahlfaktoren

- **Anforderungen analysieren:**
  - Komplexität der Anwendungsdatenstrukturen.
  - Leistungsanforderungen und Abstraktionsgrad.
  - Unterstützung von verschiedenen Datenbanken und Cross-Plattform-Bedarf.
- **Projektumfang und Ressourcen:**
  - Evaluieren Sie das Projektvolumen und vorhandene Entwickler-Fachkenntnisse.

Vorteile der Verwendung von EF Core

- **Entwicklerproduktivität:**
  - Abstraktion der Datenbankinteraktion, was die Menge an Boilerplate-Code reduziert.
  - LINQ unterstützt intuitive und typsichere Abfragen.
- **Cross-Plattform:**
  - Lauffähig auf Windows, Linux und macOS in .NET Core-Umgebungen.
- **Flexibel:**
  - Unterstützung für beide Ansätze:
    - **Code-First:** Modellcode als primäre Quelle, Datenbank wird basierend darauf erstellt.
    - **Database-First:** Nutzen Sie eine bestehende Datenbank, um das Datenmodell zu generieren.

Vorteile der Verwendung von EF Core

- **Leistungsfähigkeit:**
  - Optimierte Abfragen und Cache-Strategien verbessern die Anwendungseffizienz.
- **Erweiterbar:**
  - Anpassbar mit benutzerdefinierten Abbildungen und Verhalten.

Grundlegende Konzepte von EF Core

Der `DbContext`

- **Was ist `DbContext`?**
  - Die zentrale Klasse zur Vermittlung zwischen einer Datenbank und Ihrer C#-Anwendung.
  - Verwaltet die Entitäten im Speicher während der Erstellung, Abfrage, Aktualisierungen oder Löschung.
- **Hauptfunktionen:**
  - Verwaltung von Datenbankverbindungen und Konfigurationen.
  - Verfolgung von Änderungen und Persistenz von Daten.
  - Angebot von CRUD-Operationen auf Daten.

Der `DbSet`

- **Was ist `DbSet`?**
  - Eine Sammlung von Entitäten eines bestimmten Typs, die in der Datenbank abgefragt oder gespeichert werden können.
  - Repräsentiert eine Tabelle oder Sicht im Kontext von EF Core.
- **Verwendung:**
  - Jede Entität im Datenmodell wird als `DbSet<TEntity>` innerhalb des `DbContext` definiert.

```
public class AppDbContext : DbContext
{
    public DbSet<Product> Products { get; set; }
}
```

Code First vs. Database First

Code First

- **Vorgehensweise:**
  - Modellklassen definieren und EF Core generiert die Datenbank.
- **Vorteile:**
  - Direkte Kontrolle über das Datenmodell im Code.
  - Migrationen ermöglichen einfache Modifizierungen.

Database First

- **Vorgehensweise:**
  - Mit einer bestehenden Datenbank verbinden und das Modell generieren lassen.
- **Vorteile:**
  - Ideal, wenn bereits Datenbankschemata vorhanden sind.
  - Nutzt die bestehenden Strukturen und Regeln direkt.

Auswahlkriterien

- **Code First:**
  - Geeignet für neue Projekte ohne bestehende Datenbank.
  - Entwickler mit Fokus auf Domänenmodellierung bevorzugt.
- **Database First:**
  - Nutzbar bei vorhandener Datenbankstruktur und -richtlinie.
  - Bevorzugt in Datenbank-zentrierten Entwicklungsumgebungen.

Einrichtung eines EF Core Projekts

Installation und Konfiguration von EF Core

Schritt 1: Neues Projekt erstellen

- **Visual Studio:**
  - Erstellen Sie ein neues .NET Core Console-Projekt.
- **.NET CLI:**

```
dotnet new console -n EFCoreExample
cd EFCoreExample
```

Installation und Konfiguration von EF Core

Schritt 2: EF Core Pakete installieren

- **NuGet Package Manager Konsole:**
  - `Install-Package Microsoft.EntityFrameworkCore`
  - `Install-Package Microsoft.EntityFrameworkCore.SqlServer`
  - `Install-Package Microsoft.EntityFrameworkCore.Tools`
- **.NET CLI:**

```
dotnet add package Microsoft.EntityFrameworkCore
dotnet add package Microsoft.EntityFrameworkCore.SqlServer
dotnet add package Microsoft.EntityFrameworkCore.Tools
```

Verbindung zu einem SQL Server einrichten

Schritt 3: Erstellen und Konfigurieren von `DbContext`

- **Erstellen Sie die `LibraryDbContext`-Klasse:**

```
public class LibraryDbContext : DbContext
{
    public DbSet<Book> Books { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        // Ersetzen Sie durch Ihre Datenbankverbindungszeichenfolge
        optionsBuilder.UseSqlServer("Server=.;Database=LibraryDB;Trusted_Connection=True;");
    }
}
```

- **Anpassen der Verbindungszeichenfolge:**
  - Passen Sie die Parameter an Ihre SQL Server-Umgebung und -datenbank an.

Definition eines einfachen Modells

Schritt 4: Modellierung der Entitäten

- **Erstellen der `Book` -Klasse:**

```
public class Book
{
    public int Id { get; set; }
    public string Title { get; set; }
    public string Author { get; set; }
    public int PublicationYear { get; set; }
}
```

- **Sicherstellen, dass der `DbSet<Book>` existiert:**
    - Verwenden, um CRUD-Operationen auf der `Books` -Tabelle auszuführen.

Zusammenfassung der Einrichtung

1. **Projekt erstellen und konfigurieren:**
    - Mit .NET CLI oder Visual Studio.
2. **Pakete hinzufügen:**
    - Installation der erforderlichen EF Core-Pakete.
3. **Verbindungsaufbau:**
    - Konfigurieren Sie den `DbContext`, um SQL Server zu nutzen.
4. **Modell definieren:**
    - Erstellen Sie Klassen, um Tabellen im Datenbankmodell widerzuspiegeln.

Erste Schritte mit CRUD-Operationen

Erstellen von Datenbanktabellen aus dem Modell (Migrationen)

- **Was sind Migrationen?**
  - Werkzeuge zur Verwaltung von Datenbank-Schemaänderungen über Versionen hinweg.
- **Befehle zum Erstellen und Aktualisieren:**
  - **Migration erstellen:**

```
dotnet ef migrations add InitialCreate
```

  - **Datenbank aktualisieren:**

```
dotnet ef database update
```

- **Vorteil:**
  - Kein manuelles SQL erforderlich, um Änderungen zu verwalten und anzuwenden.

Grundlagen von CRUD

Create (Erstellen)

```
using (var context = new LibraryDbContext())
{
    var book = new Book { Title = "C# Basics", Author = "John Doe", PublicationYear = 2021 };
    context.Books.Add(book);
    context.SaveChanges();
}
```

- **Funktion:** Neues Datensatzobjekt hinzufügen und Änderungen speichern.

Read (Lesen)

```
using (var context = new LibraryDbContext())
{
    var books = context.Books.ToList();
    books.ForEach(b => Console.WriteLine(b.Title));
}
```

- **Funktion:** Abrufen von Daten aus der Datenbank.

Update (Aktualisieren)

```
using (var context = new LibraryDbContext())
{
    var book = context.Books.First(b => b.Id == 1);
    book.Title = "Advanced C#";
    context.SaveChanges();
}
```

- **Funktion:** Vorhandene Daten ändern und speichern.

Delete (Löschen)

```
using (var context = new LibraryDbContext())
{
    var book = context.Books.First(b => b.Id == 1);
    context.Books.Remove(book);
    context.SaveChanges();
}
```

- **Funktion:** Entfernung eines Datensatzes aus der Datenbank.

Zusammenfassung der CRUD-Operationen

- **Create:** Fügen Sie dem DbSet ein neues Objekt hinzu und speichern Sie die Änderungen.
- **Read:** Verwenden von LINQ-Abfragen, um die Daten abzurufen und zu filtern.
- **Update:** Änderungen an bestehenden Objekten vornehmen und speichern.
- **Delete:** Entnahme von Objekten und Persistenz dieser Änderungen.

Übungen

Übung 1: Modell- und Datenbankerstellung mit EF Core

Ziel:

- Erstellen und konfigurieren Sie ein einfaches Datenmodell mit EF Core.
- Richten Sie den `DbContext` ein, um die Verbindung zu einem SQL Server herzustellen.
- Verwenden Sie Migrationen, um das Datenmodell in eine Datenbank umzuwandeln.

Übung 2: Einfaches CRUD mit EF Core

Ziel:

- Lernen Sie, CRUD-Operationen (Create, Read, Update, Delete) in EF Core zu verwenden.
- Implementieren Sie Funktionen, um Daten in der Datenbank hinzuzufügen, abzurufen, zu aktualisieren und zu löschen.

Fluent API vs. Data Annotations

Data Annotations

- **Definition:**
  - Attribute zur Konfiguration von Entitätsklassen direkt im Code.

```
public class Book
{
    public int Id { get; set; }

    [Required]
    [MaxLength(100)]
    public string Title { get; set; }
}
```

- **Vorteile:**
  - Einfache und direkte Konfiguration.
  - Code-fokussiert, leicht zu verstehen.
- **Nachteile:**

Fluent API

- **Definition:**
  - Ermöglicht die Konfiguration des EF Core-Verhaltens über die `OnModelCreating`-Methode der DbContext-Klasse.
- **Beispiele:**

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Book>()
                .Property(b => b.Title)
                .IsRequired()
                .HasMaxLength(100);
}
```

Fluent API

- **Vorteile:**
  - Mehr Kontrolle und Flexibilität über die Konfiguration.
  - Größere Unterstützung für komplizierte Beziehungen und Vererbungen.
- **Nachteile:**
  - Kann komplexer und weniger intuitiv sein.

Eager vs. Lazy Loading

Eager Loading

- **Definition:**
  - Daten werden beim ersten Abfragen vorgeladen, einschließlich der Navigationseigenschaften.
- **Verwendung:**

```
var books = context.Books.Include(b => b.Author).ToList();
```

Eager Loading: Vor- und Nachteile

- **Vorteile:**
  - Vermeidet verzögerte (n+1)-Abfrageprobleme.
  - Nützlich, wenn die Daten sofort benötigt werden.
- **Nachteile:**
  - Keine selektive Daten-Änderung möglich.
  - Kann mehr Daten laden, als eigentlich gebraucht.

Lazy Loading

- **Definition:**
  - Navigationseigenschaften werden nur geladen, wenn sie explizit angefordert werden.
- **Implementierung:**

```
public class Book
{
    public int Id { get; set; }
    public string Title { get; set; }
    public virtual Author Author { get; set; }
}
```

Lazy Loading: Vor- und Nachteile

- **Vorteile:**
  - Effiziente Nutzung der Datenbankressourcen.
  - Lädt nur benötigte Daten.
- **Nachteile:**
  - Potenzial für (n+1)-Abfrageprobleme.
  - Zusätzliche Verzögerungen bei weiteren Datenbankaufrufen.

Zusammenfassung der fortgeschrittenen Konzepte

- **Fluent API und Data Annotations:** Zwei Ansätze zur Modellkonfiguration mit unterschiedlichen Vorzügen und Anwendungsfällen.
- **Eager und Lazy Loading:** Steuerung der Datenabrufstrategie für Navigationseigenschaften, um Effizienz und Leistung zu optimieren.

Abfragen mit LINQ in EF Core

EF Core und LINQ: Datenbankspezifische Abfragen

LINQ in EF Core

- **Definition:**
  - Language Integrated Query (LINQ) bietet eine einheitliche Methode zur Arbeit mit Daten direkt in Ihrer C#-Anwendung.
- **Verwendung in EF Core:**
  - Ermöglicht die Erstellung komplexer Abfragen, die von EF Core in SQL übersetzt und zur Datenbank gesendet werden.

EF Core und LINQ: Datenbankspezifische Abfragen

Vorteile

- **Typsicherheit:**
  - Compiler-geprüfte Abfragen vermeiden Laufzeitfehler.
- **Lesbarkeit:**
  - LINQ-Abfragen sind oft intuitiv und lesbar.

Projektionsoperationen mit LINQ

Projektion in LINQ

- **Definition:**
  - Auswahl spezifischer Felder oder Transformation der Ergebnisse einer Abfrage.
- **Beispiel:**

```
var bookTitles = context.Books
                        .Select(b => new { b.Title, b.Author })
                        .ToList();
```

- **Ergebnis:**
  - Eine Liste anonymisierter Objekte nur mit Titel und Autor.

Filteroperationen mit LINQ

Filtern von Daten

- **Definition:**
  - Auswählen von Elementen, die bestimmten Kriterien entsprechen.
- **Beispiel:**

```
var recentBooks = context.Books
                         .Where(b => b.PublicationYear > 2015)
                         .ToList();
```

- **Ergebnis:**
  - Eine Liste von Büchern, die nach 2015 veröffentlicht wurden.

Aggregationsoperationen mit LINQ

Aggregation in LINQ

- **Definition:**
  - Zusammenfassen von Daten durch Funktionen wie `Count`, `Sum`, `Average`, `Min`, und `Max`.
- **Beispiel:**

```
var totalBooks = context.Books.Count();
var averagePrice = context.Books.Average(b => b.Price);
```

- **Ergebnis:**
  - Gesamtanzahl der Bücher und der durchschnittliche Preis aller Bücher.

Zusammenfassung der LINQ-Abfragen

- **Projektionsoperationen:** Wandeln Sie Datenformen um und extrahieren Sie spezifische Felder.
- **Filteroperationen:** Isolieren Sie relevante Daten basierend auf Bedingungen.
- **Aggregationsoperationen:** Fassen Sie Daten für Berichte zusammen und analysieren Sie sie.

Übungen

Übung 3: Beziehungen und Navigationseigenschaften

Ziel:

- Definieren und verwenden Sie Beziehungen zwischen Entitäten mittels EF Core.
- Nutzen Sie Navigationseigenschaften, um auf verwandte Daten zuzugreifen und diese zu manipuliere

Übung 4: LINQ-Abfragen mit EF Core

Ziel:

- Erstellen und ausführen Sie komplexe LINQ-Abfragen in EF Core.
- Arbeiten Sie mit Projektions-, Filter- und Aggregationsoperationen, um Daten präzise abzurufen und zu analysieren.