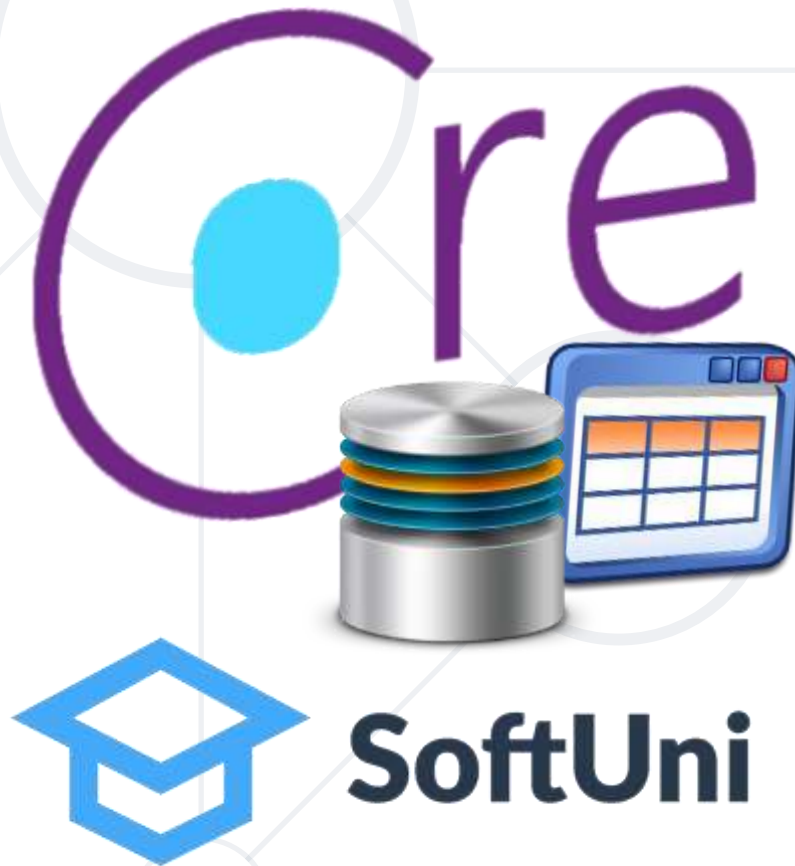


Best Practices and Architecture

Useful Patterns and Code Structure



SoftUni Team
Technical Trainers



Software University

<https://about.softuni.bg/>

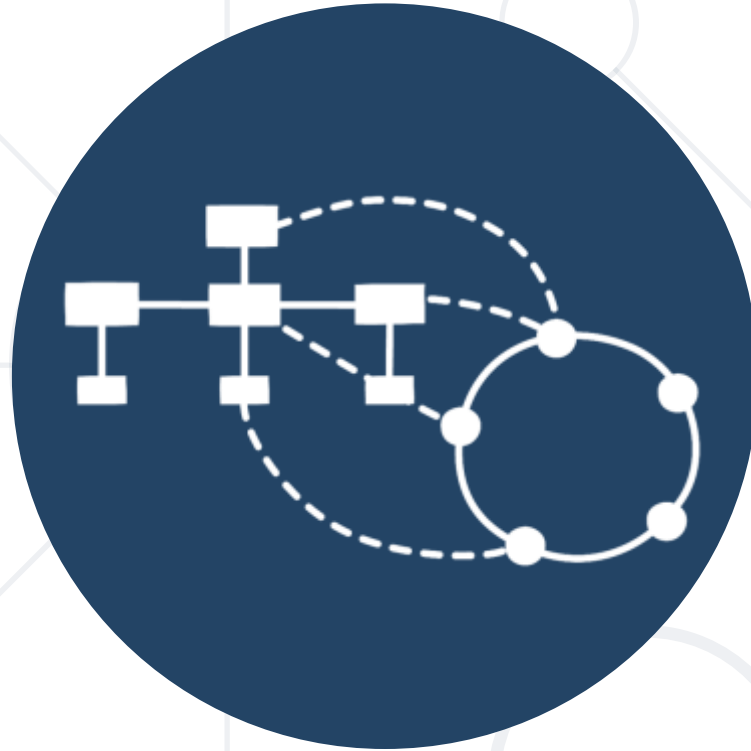
Table of Contents

- Project Structure
- EF Core Optimizations
- Useful Patterns



sli.do

#CSharpDB



Project Structure

Organizing Solutions

Importance of Organized Code

- Scalability
- Maintainability
- Manageability
- Testability

S

Single Responsibility

O

Open / Closed

L

Liskov Substitution

I

Interface Segregation

D

Dependency Inversion

- Application code can be split into sections
 - **Data Layer** – database connection (context)
 - **Domain Models** – entity classes
 - **Client** – user-interaction and app logic
 - **Business Logic** – data validation, transformations
- Reasons
 - Easier to locate files when maintaining
 - Don't have to rebuild entire codebase after changes (DLLs)



Usage Optimization

Entity Framework Core Performance

- Only fetch **required data** by filtering and projecting your queries

```
context.Employees
    .Where(e => e.Salary >= 15000)
    .Select(e => new {
        e.FirstName,
        e.LastName,
        e.Salary
    })
    .ToList();
```

```
SELECT
    1 AS [C1],
    [Extent1].[FirstName] AS [FirstName],
    [Extent1].[LastName] AS [LastName],
    [Extent1].[Salary] AS [Salary]
FROM [dbo].[Employees] AS [Extent1]
WHERE [Extent1].[Salary] >= cast(15000 as decimal(18))
```


- LINQ queries are **executed** each time the data is **accessed**
 - If materialized in a collection – **ToList()**
 - If the elements are aggregated – **Count()**, **Average()**, **First()**
 - When a property is accessed
- Try to delay execution (materialization) until you actually need the results
- You can monitor query execution using **Express Profiler**

- EF will cache entities and compare the cache for changes
 - Use **Find()** with change detection disabled

```
try
{
    context.ChangeTracker.AutoDetectChangesEnabled = false;
    var product = context.Products.Find(productId);
    ...
}
finally
{
    context.ChangeTracker.AutoDetectChangesEnabled = true;
}
```

- When adding or updating a record, Entity Framework makes a call to **DetectChanges()**
- Use **AddRange()** and **RemoveRange()** to reduce calls

```
List<Product> products = new  
List<Product>()  
    { product1, product2, product3 };  
  
context.Products.AddRange(products);
```

Works with
any collection

- Entity Framework builds **associations** and **tracks changes** for every loaded entity
- If we **only** want to **display** data, this process is redundant
- Disable tracking

```
context.Products  
    .AsNoTracking()  
    .Where(p => p.Price < 150)  
    .ToList();
```

- Note this also **disables caching!**

- Payload size and number of roundtrips to the database are inversely proportional
 - **Lazy** – less data, more queries
 - **Eager** – more data, less queries
- There is no best approach – performance depends on usage scenario

- Do you need to access many **navigation properties** from the fetched entities?
 - **No** – **Lazy** for large payloads, **Eager** for small
 - **Yes** – **Eager** loading for up three entities, **Lazy** for more
- Do you know exactly what data will be needed at run time?
 - **No** – **Lazy**
 - **Yes** – **Eager** at first unless, **Lazy** if loading lots of data

- Is your code executing far from your database? (increased network latency)
 - **No** – **Lazy** will simplify your code; don't take database proximity for granted
 - **Yes** – Depending on scenario **Eager** will require fewer round trips
- Always test application-wide performance, only optimize if results aren't satisfactory



Design Patterns

Solving Problems More Easily

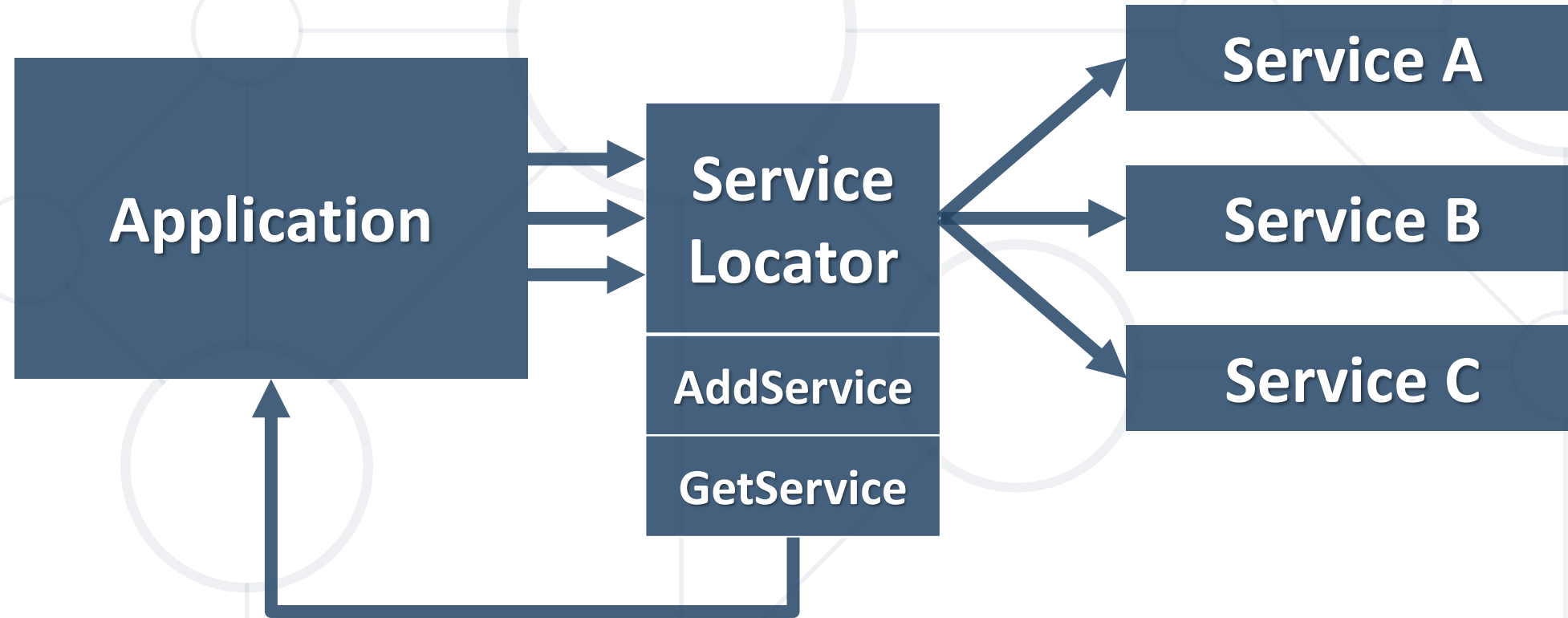
- **Singleton** – Ensure a class has only one instance and provide a global point of access to it
- **Service Locator** – Make a service available globally and decouple the calling class from the dependent object
- **Dependency Injection** - no client code has to be changed simply because an object it depends on needs to be changed to a different one
- **Command** – Encapsulate a request as an object, allowing delayed execution, undo and replay
- **Repository** – Separates the data access logic and maps it to the entities in the business logic
- **Unit of work** – Used to group one or more into a single transaction or "unit of work", so that all operations either pass or fail as one

Singleton Pattern

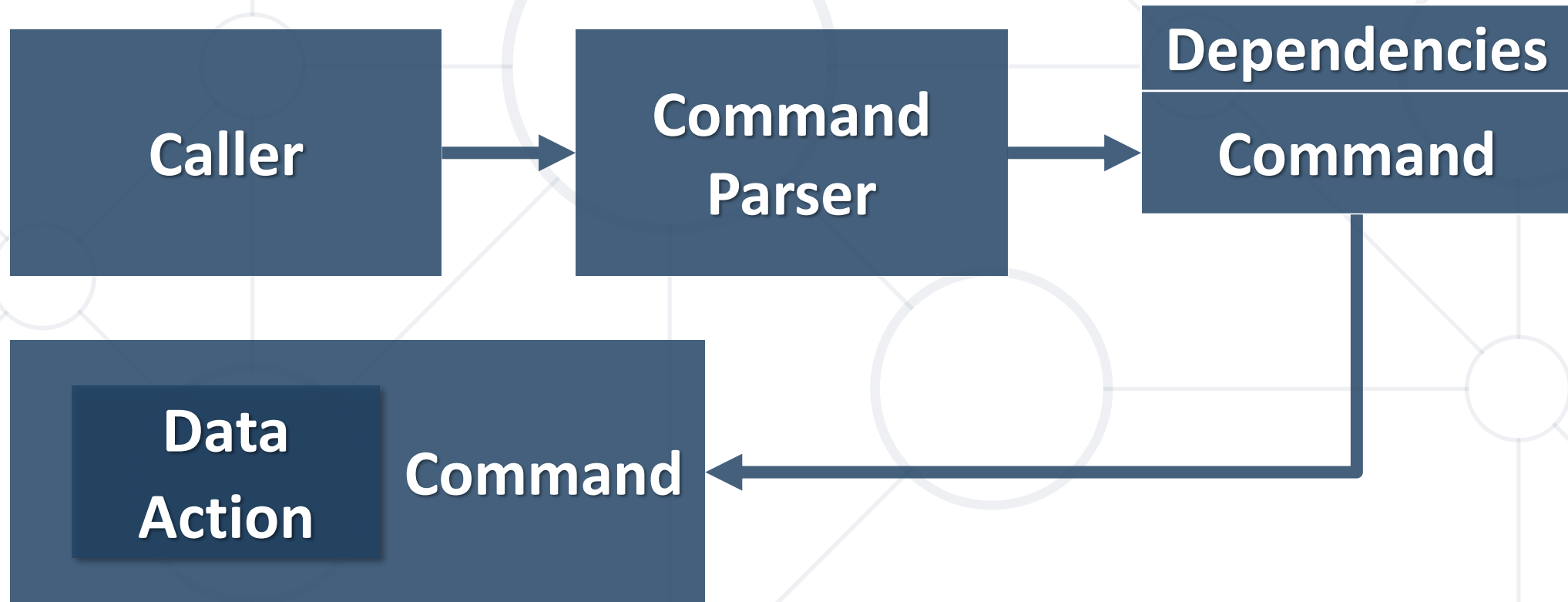
```
public class Authenticator
{
    private static Authenticator instance;
    private Authenticator() { ... }
    public static Authenticator Instance
    {
        get
        {
            if (instance == null)
                instance = new Authenticator();
            return instance;
        }
    }
}
```

Private Constructor

Instantiate when first
accessed



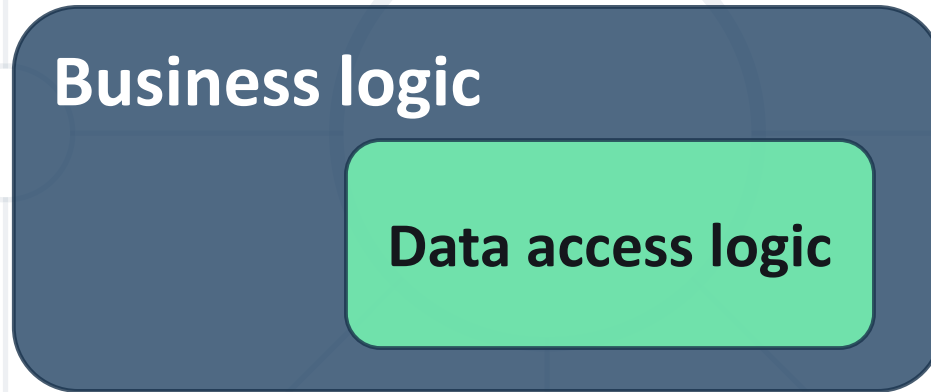
Command Pattern



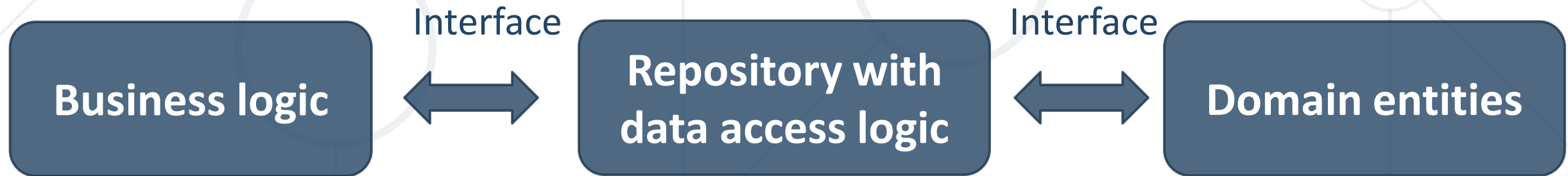
- It works with the **domain entities** and performs **data access logic**
- The domain entities, the data access logic and the business logic talk to each other **using interfaces**
- It **hides the details** of data access from the business logic
- Business logic **can access** the data object without having knowledge of the underlying data access architecture

Repository Pattern

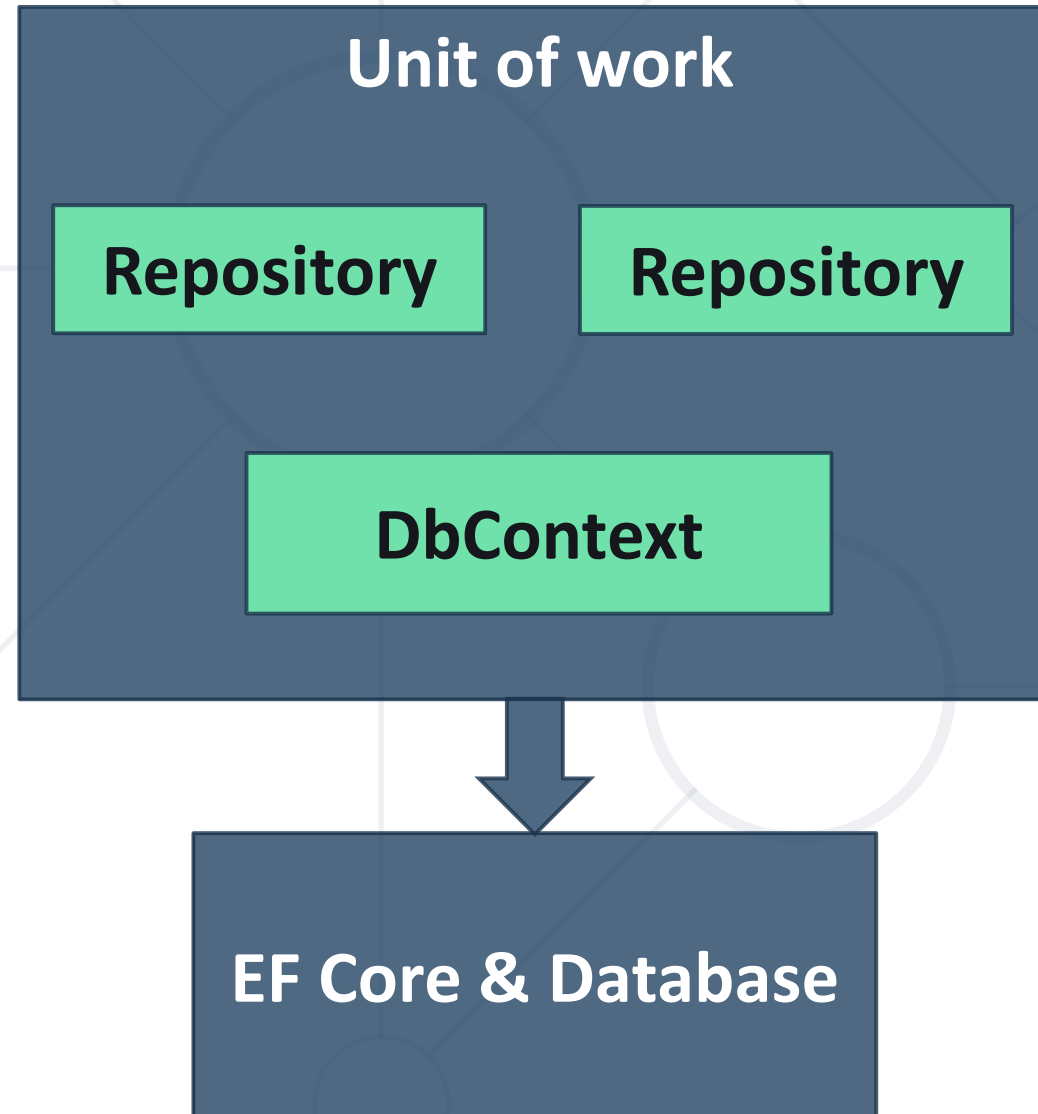
- Without repository



- With repository



- **Serves one purpose** – to make sure that when you use multiple repositories, they share a **single** database context
- With a Unit of Work, you might also choose to implement **Undo / Rollback** functionality
 - When using Entity Framework Core, the recommended approach to undo is to **discard your context** with the changes you are interested in undoing





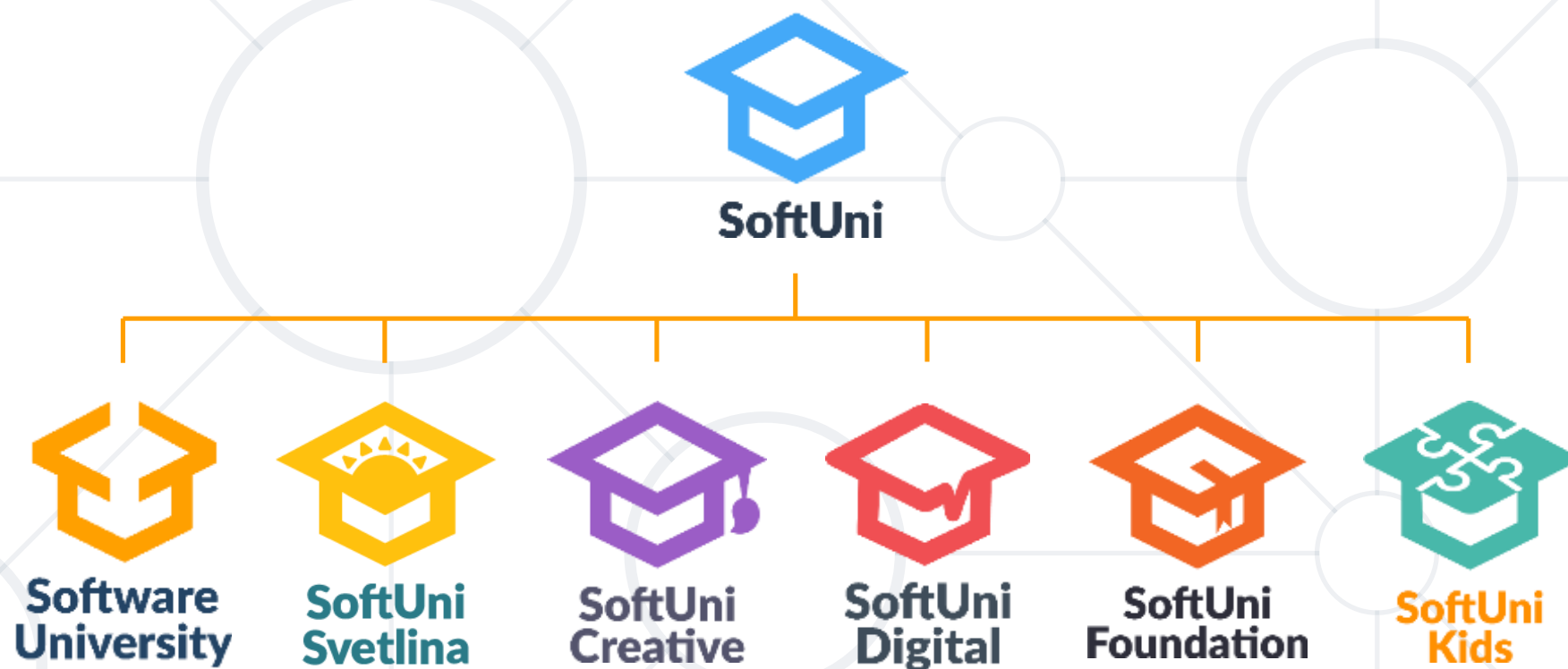
Best Practices and Architecture

Live Demo

- **Project structure** is important as an application is scaled
- Entity Framework Core **performance** can be improved by following certain guidelines
- **Design Patterns** define a common approach to solving certain development problems



Questions?



SoftUni Diamond Partners

SCHWARZ



Coca-Cola HBC
Bulgaria



Postbank

Решения за твоето утре



POKERSTARS



CAREERS



AMBITIONED

DXC
TECHNOLOGY



**SOFTWARE
GROUP**

Bosch.IO

INDEAVR
Serving the high achievers

**DRAFT
KINGS**



SmartIT

createX

**SUPER
HOSTING
.BG**



- Software University – High-Quality Education, Profession and Job for Software Developers

- softuni.bg, about.softuni.bg

- Software University Foundation

- softuni.foundation

- Software University @ Facebook

- facebook.com/SoftwareUniversity

- Software University Forums

- forum.softuni.bg



Software University



- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – <https://about.softuni.bg>
- © Software University – <https://softuni.bg>

