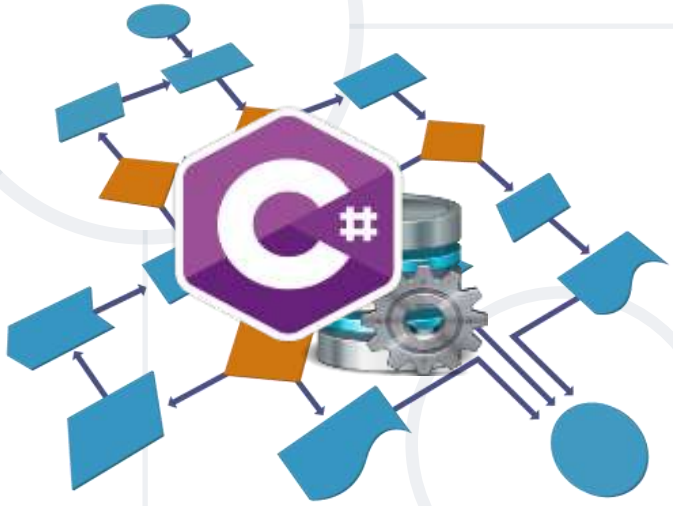
C# Auto Mapping Objects

Manual Mapping

and AutoMapper Library



SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://about.softuni.bg/

Table of Contents

Have a Question?

Software University

sli.do

#csharp-db

Data Transfer Objects

Definition and Usage

What is a Data Transfer Object?

- A **DTO** is an object that **carries data** between processes
  - Used to **aggregate** only the **needed information** in a single call
  - Example: In web applications, between the **server** and **client**
- Doesn't contain any logic – only **stores values**

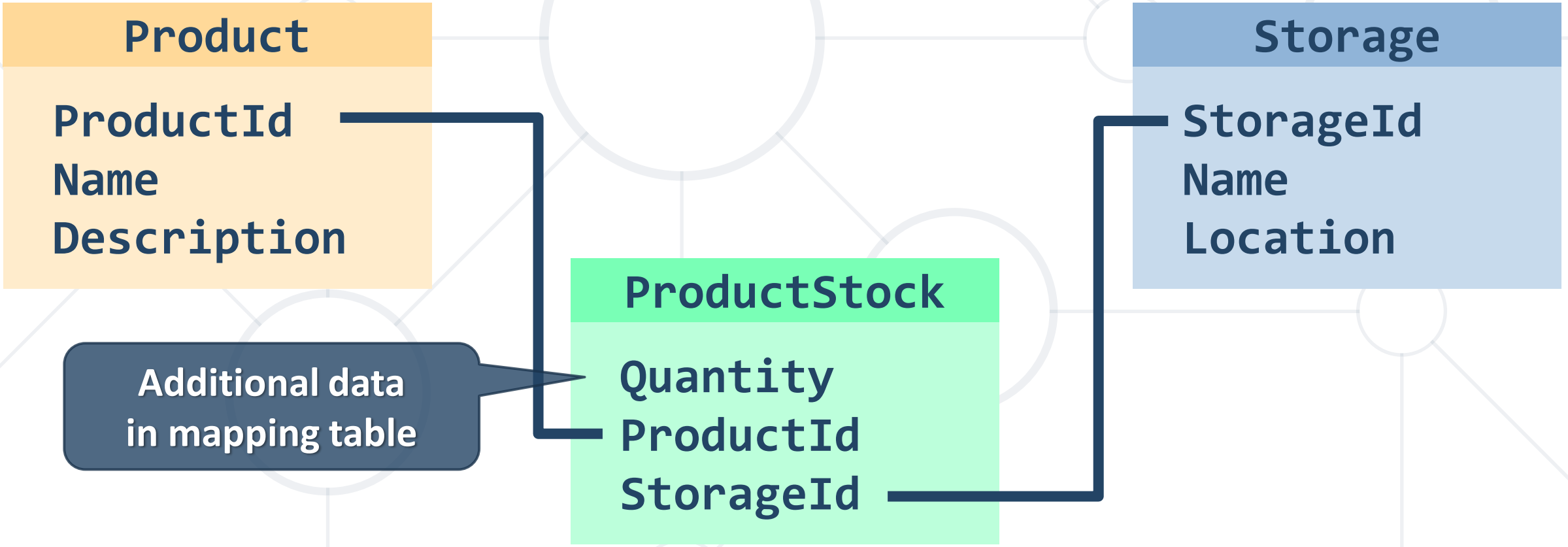
```
public class ProductDTO
{
   public string Name { get; set; }

   public int StockQty { get; set; }
}
```

DTO Usage Scenarios

- **Hide** particular properties that **clients** are not supposed to view
- **Remove** circular references
- **Omit** some properties in order to **reduce** payload **size**
- **Flatten** object graphs that contain nested objects to make them more convenient for clients (denormalization)
- **Decouple** your service layer from your database layer

Manual Mapping (1)

- Relationship Diagram



Manual Mapping (2)

- Get product name and stock quantity in a new DTO object

```
var product =
  context.Products.FirstOrDefault();
var productDto = new ProductDTO
{
  Name = product.Name,
  StockQty = product.ProductStocks
    .Sum(ps => ps.Quantity)
};
```

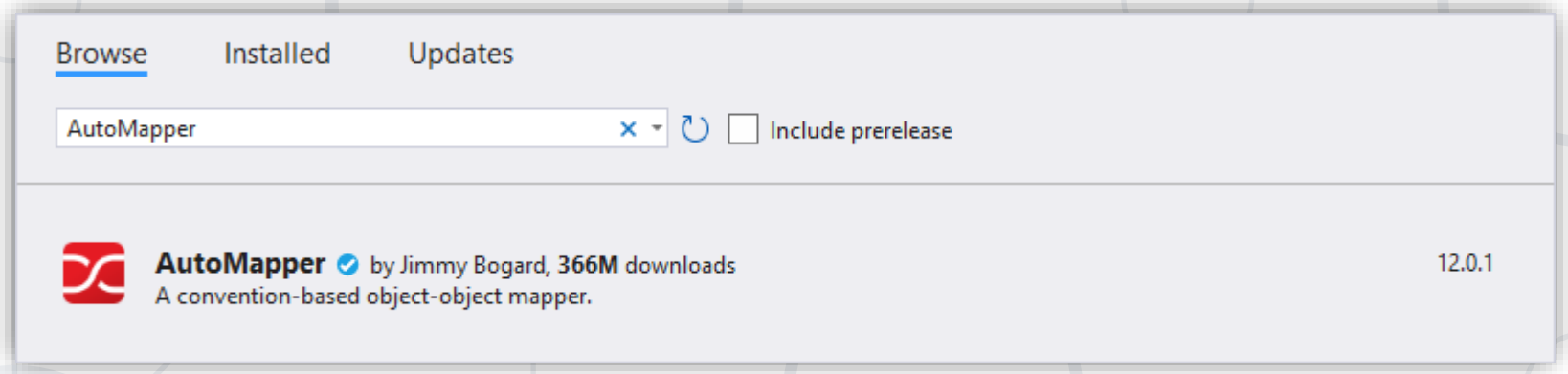
Aggregate information from mapping table

AutoMapper Library

Automatic Translation of Domain Objects

What is AutoMapper?

- Library to eliminate **manual mapping** code
- Available as a **NuGet** Package




```
Install-Package AutoMapper
```

- Official **Website** and **GitHub**

Initialization and Configuration

- AutoMapper offers an **instance service** for use and configuration
  - Add mappings between objects and DTOs

```
var config = new MapperConfiguration(cfg =>
    cfg.CreateMap<Product, ProductDTO>());
var mapper = config.CreateMapper();
```

Source

Target

- Properties will be mapped **by name**

```
var product = context.Products.FirstOrDefault();
ProductDTO dto = mapper.Map<ProductDTO>(product);
```

Multiple Mappings

- You can configure all mapping configurations at once

```
var config = new MapperConfiguration(cfg =>
{
    cfg.CreateMap<Product, ProductDTO>();
    cfg.CreateMap<Order, OrderDTO>();
    cfg.CreateMap<Client, ClientDTO>();
    cfg.CreateMap<SupportTicket, TicketDTO>();
});
var mapper = config.CreateMapper();
```

Mapping ICollection and IQueryable

- EF Core uses `IQueryable<T>` for all DB operations
  - AutoMapper can work with `IQueryable<T>` to map classes
- Using **`AutoMapper`** to map an entire DB collection

```
var posts = context.Posts
  .Where(p => p.Author.Username == "Nikolay.IT")
  .ProjectTo<PostDto>(config)
  .ToList();
```

IQueryable<PostDto>

IQueryable<Post>

- Works like an automatic `.Select()`
  - AutoMapper helps EF to generate **optimized SELECT SQL query** (like projection with an **anonymous object**)

Custom Member Mapping

- Map properties that don't match naming convention

```
var config = new MapperConfiguration(cfg =>
{
  cfg.CreateMap<Product, ProductDTO>()
     .ForMember(dto => dto.StockQty,
         opt => opt.MapFrom(src =>
             src.ProductStocks.Sum(p => p.Quantity))));
});
```

Destination property

Source

Flattening Complex Objects

- **Flattening** of related objects is automatically supported

```
public class OrderDTO
{
  public string ClientName { get; set; }
  public decimal Total { get; set; }
}
```

- **AutoMapper** understands **ClientName** is the **Name** of a **Client**

```
var config = new MapperConfiguration(cfg =>
   cfg.CreateMap<Order, OrderDTO>());
var mapper = config.CreateMapper();
OrderDTO dto = mapper.Map<Order, OrderDTO>(order);
```

Unflattening Complex Objects

- **Unflattening** of related objects is automatically supported

```
public class OrderDTO
{
  public string ClientName { get; set; }
  public decimal Total { get; set; }
}
```

- **AutoMapper** understands **ClientName** is the **Name** of a **Client**, but to unflatten it, it needs **ReverseMap()**

```
var config = new MapperConfiguration(cfg =>
   cfg.CreateMap<Order, OrderDTO>().ReverseMap());
var mapper = config.CreateMapper();
Order order = mapper.Map<OrderDTO, Order>(dto);
```

Mapping Profiles

- We can extract our configuration to a class (called a **profile**)

```
public class ForumProfile : Profile
{
  public ForumProfile()
  {
    CreateMap<Post, PostDto>();
    CreateMap<Category, CategoryDto>();
  }
}
```

```
using AutoMapper;
```

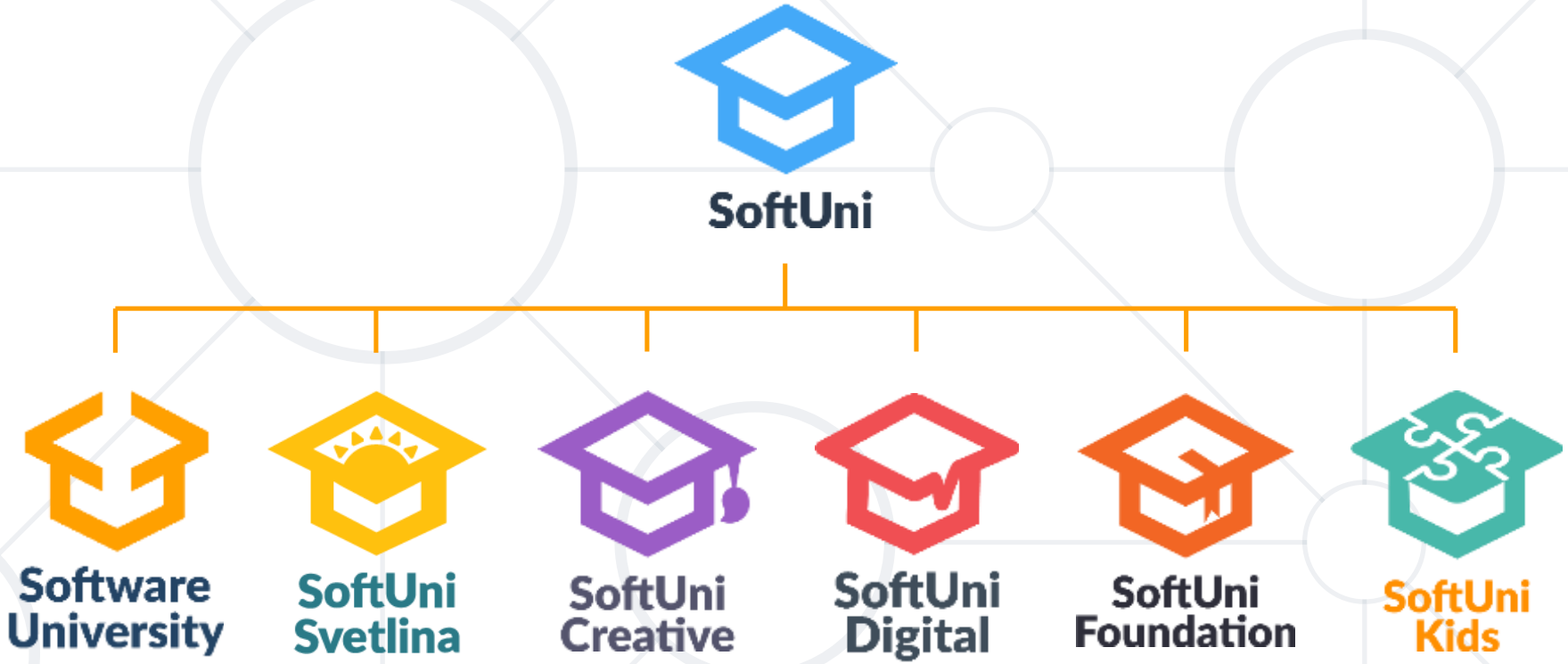
- Using our configuration class

```
var config = new MapperConfiguration(cfg =>
   cfg.AddProfile<ForumProfile>());
```

Summary

- To reduce round-trip latency and payload size, data is transformed into a **DTO**
- **AutoMapper** is a library that automates this process and reduces boilerplate code
- Complex objects can be **flattened** to fractions of their sizes

Questions?



SoftUni Diamond Partners

Software University

SCHWARZ

LIDL

Kaufland

SCHWARZ Produktion

prezero

Coca-Cola HBC Bulgaria

POKERSTARS | CAREERS

Postbank

Решения за твоето утре

AMBITIONED

DXC TECHNOLOGY

SOFTWARE GROUP

Bosch.IO

INDEAVR

Serving the high achievers

DRAFT KINGS

PHAR VISION

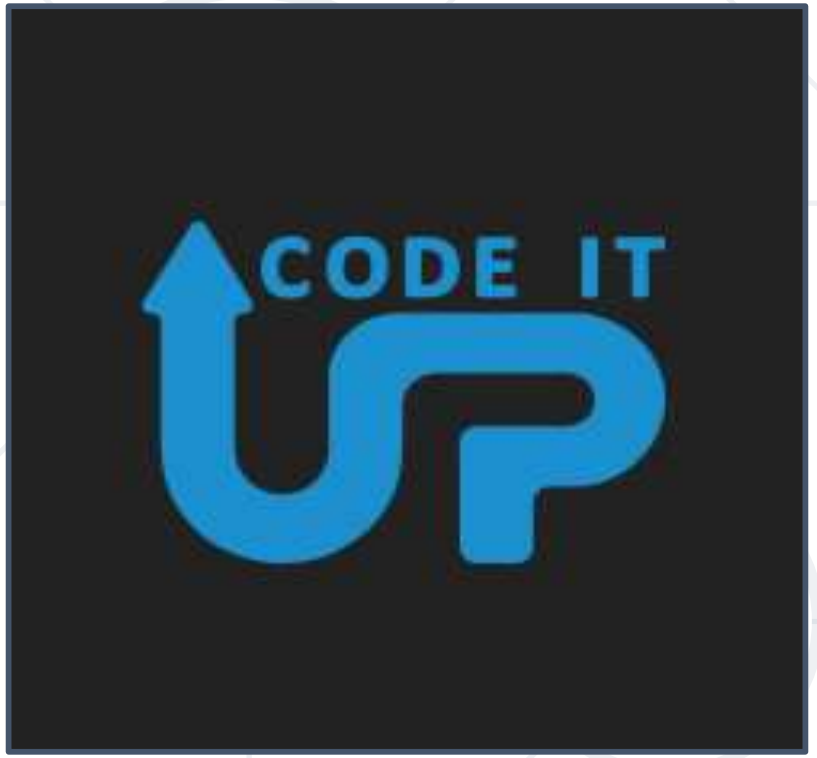
SmartIT

createX

SUPER HOSTING .BG

Educational Partners

Software University



License

- 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