Multi-User-Applikationen objektorientiert realisieren

25.11.24

DTOs und Services

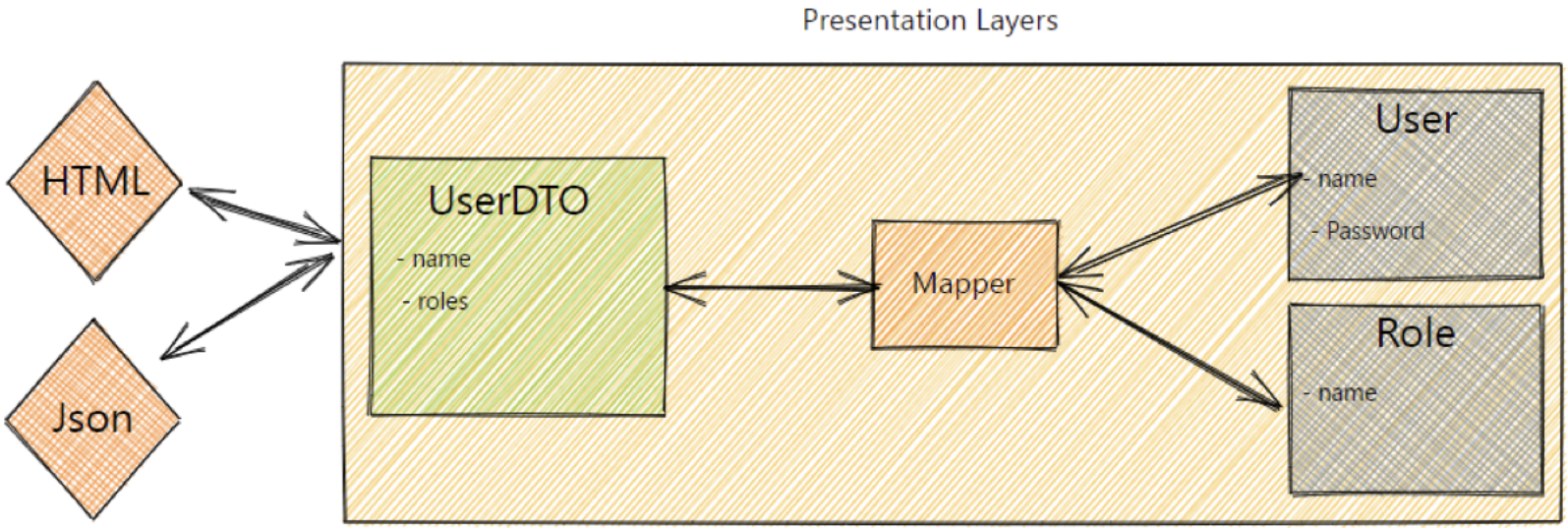
Ziele

- Du weisst was DTOs sind und kennst deren Verwendungszweck
- Du weisst wie du Entities in DTOs umwandeln kannst und umgekehrt
- Du weisst wo das Service-Layer eingesetzt wird

DTO
Data Transfer Object

- Repräsentation von Entities
- Zur Übertragung von Daten
- Senden ans Frontend
- Minimiert auf das Nötigste
- Kombinieren von Enitites

DTO
Data Transfer Object



Entity zu DTO

```
public class Person(int id, string firstName, string lastName, string password)
{
    public int Id { get; set; } = id;  2 usages
    public string FirstName { get; set; } = firstName;  2 usages
    public string LastName { get; set; } = lastName;  2 usages
    public string Password { get; set; } = password;
}
```

```
public record PersonResponseDto(int Id, string FirstName, string LastName)  3 usages
{
    public static PersonResponseDto MapToDto(Person person)  1 usage
    {
        return new PersonResponseDto(person.Id, person.FirstName, person.LastName);
    }
}
```

```
PersonResponseDto SendPerson()  1 usage
{
    var person = new Person( id: 1,  firstName: "John",  lastName: "Doe",  password: "password");
    var personDto = PersonResponseDto.MapToDto(person);
    return personDto;
}
```

DTO
Request vs Reponse

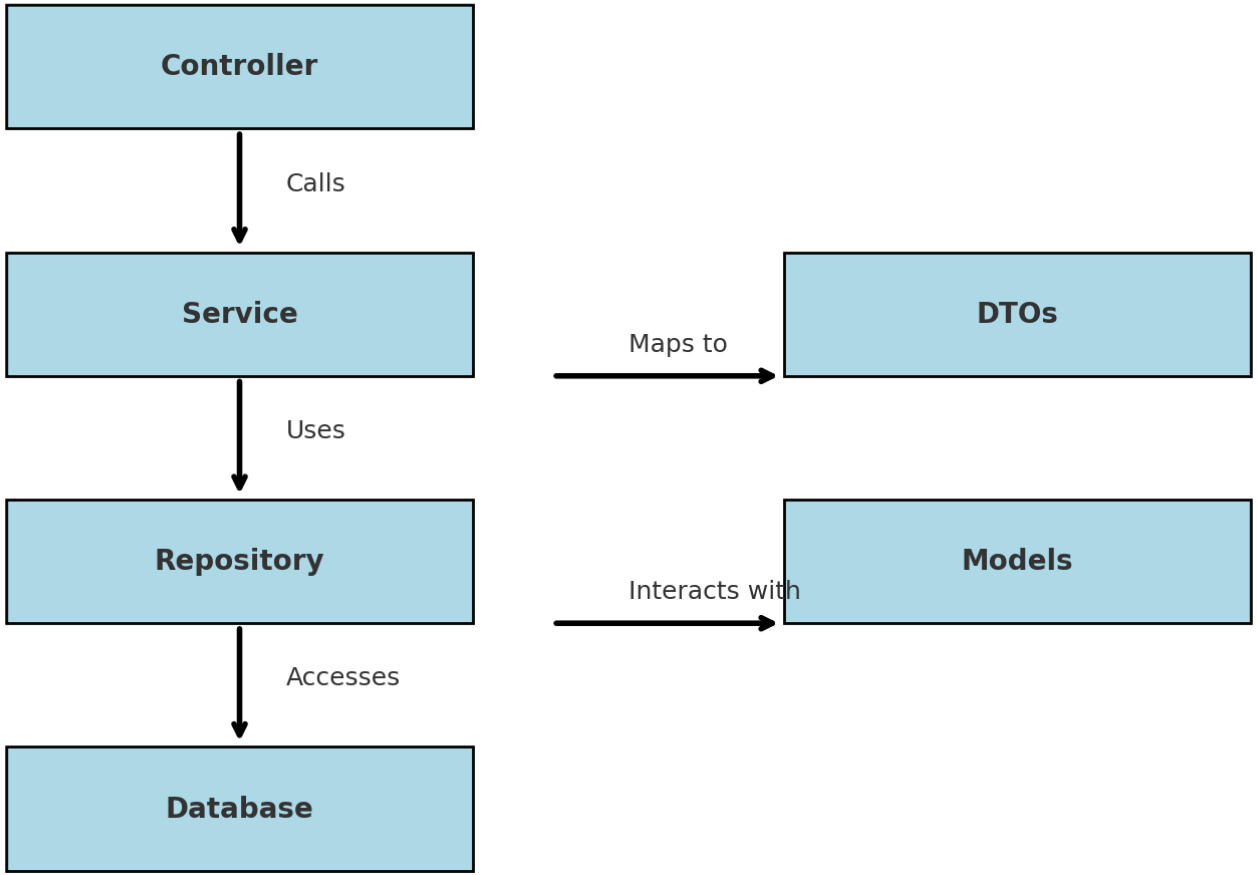
- Response enthält meist mehr Felder
  - Bsp. Datenbank Id
- Response als Erweiterung des Requests
- Spezialfälle wie AuthRequest/AuthResponse
  - Request mit PW, Response mit Id

Mapping
Entity zu DTO und zurück

- Der Controller erhält und sendet DTOs
- Der Service und die Repositories verwenden Entities
- Mapping Entity zu DTO und umgekehrt im Controller oder Service

Service

- **Separation of Concerns:** Hält die Geschäftslogik getrennt von Controllern und Repositories.
- **Reusability:** Die Geschäftslogik kann in verschiedenen Teilen der Anwendung wiederverwendet werden.
- **Testability:** Services können unabhängig von der HTTP-Schicht mittels Unit-Tests getestet werden.
- **Vereinfachte Controller:** Controller koordinieren lediglich den Ablauf zwischen den Services und der HTTP-Schicht.
- **Encapsulation:** Services können Logik aus mehreren Repositories kombinieren oder komplexe Workflows abwickeln, die nicht in Controllern oder Repositories untergebracht werden sollten.


Controller
Calls
Service
Maps to
DTOs
Uses
Repository
Interacts with
Models
Accesses
Database

Fragen?