

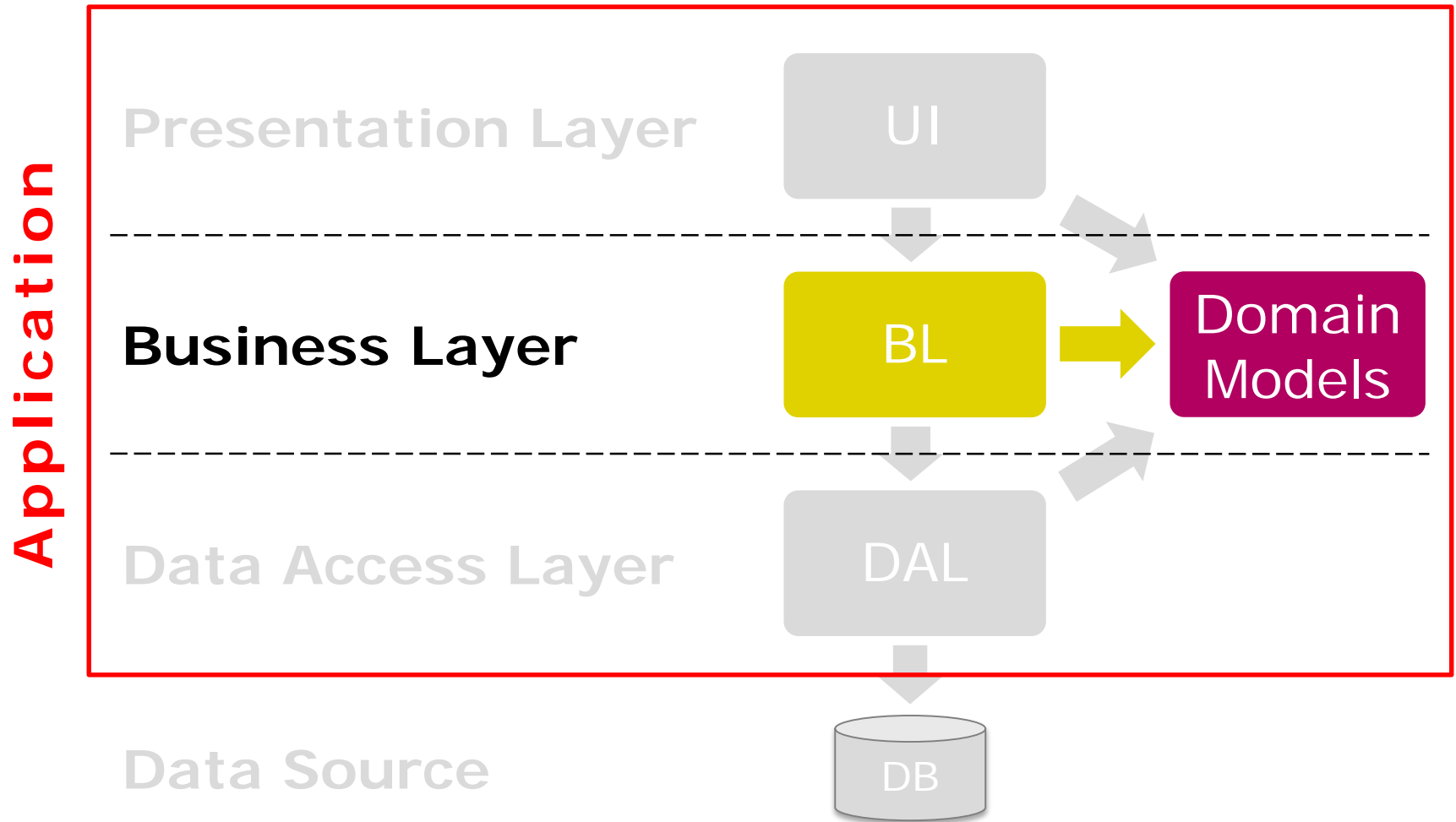
Example 'SupportCenter'

Data Validatie

Info

- Voorzie data validatie op de domeinmodellen van de applicatie
- Verwerk de controle van de validatie in de n-tier architectuur van de applicatie!

N-Tier architectuur



Validatie-definities

pre-defined

Oefening

- Voorzie validatie-definities voor volgende situaties
 - Ticket
 - 'Text' is verplicht
 - 'Text' mag niet meer dan 100 tekens zijn en toon hierbij volgende foutboodschap "Er zijn maximaal 100 tekens toegestaan"
 - HardwareTicket
 - 'DeviceName' moet beginnen met "PC-" gevolgd door een getal
 - TicketResponse
 - 'Text' is verplicht
 - 'Ticket' is verplicht

OPGELET: vergeet geen referentie te leggen naar `System.ComponentModel.DataAnnotations`

'Ticket.cs'

...

```
using System.ComponentModel.DataAnnotations;
```

```
namespace SC.BL.Domain
```

```
{  
    public class Ticket  
    {  
        public int TicketNumber { get; set; }  
        public int AccountId { get; set; }  
        [Required]  
        [MaxLength(100, ErrorMessage="Er zijn maximaal 100 tekens toegestaan")]  
        public string Text { get; set; }  
        public DateTime DateOpened { get; set; }  
        public TicketState State { get; set; }  
  
        public ICollection<TicketResponse> Responses { get; set; }  
    }  
}
```

'HardwareTicket.cs'

```
...  
using System.ComponentModel.DataAnnotations;  
  
namespace SC.BL.Domain  
{  
    public class HardwareTicket : Ticket  
    {  
        [RegularExpression("^(PC-)[0-9]+")]  
        public string DeviceName { get; set; }  
    }  
}
```

'TicketResponse.cs'

```
...  
using System.ComponentModel.DataAnnotations;  
  
namespace SC.BL.Domain  
{  
    public class TicketResponse  
    {  
        public int Id { get; set; }  
        [Required]  
        public string Text { get; set; }  
        public DateTime Date { get; set; }  
        public bool IsClientResponse { get; set; }  
  
        [Required]  
        public Ticket Ticket { get; set; }  
    }  
}
```

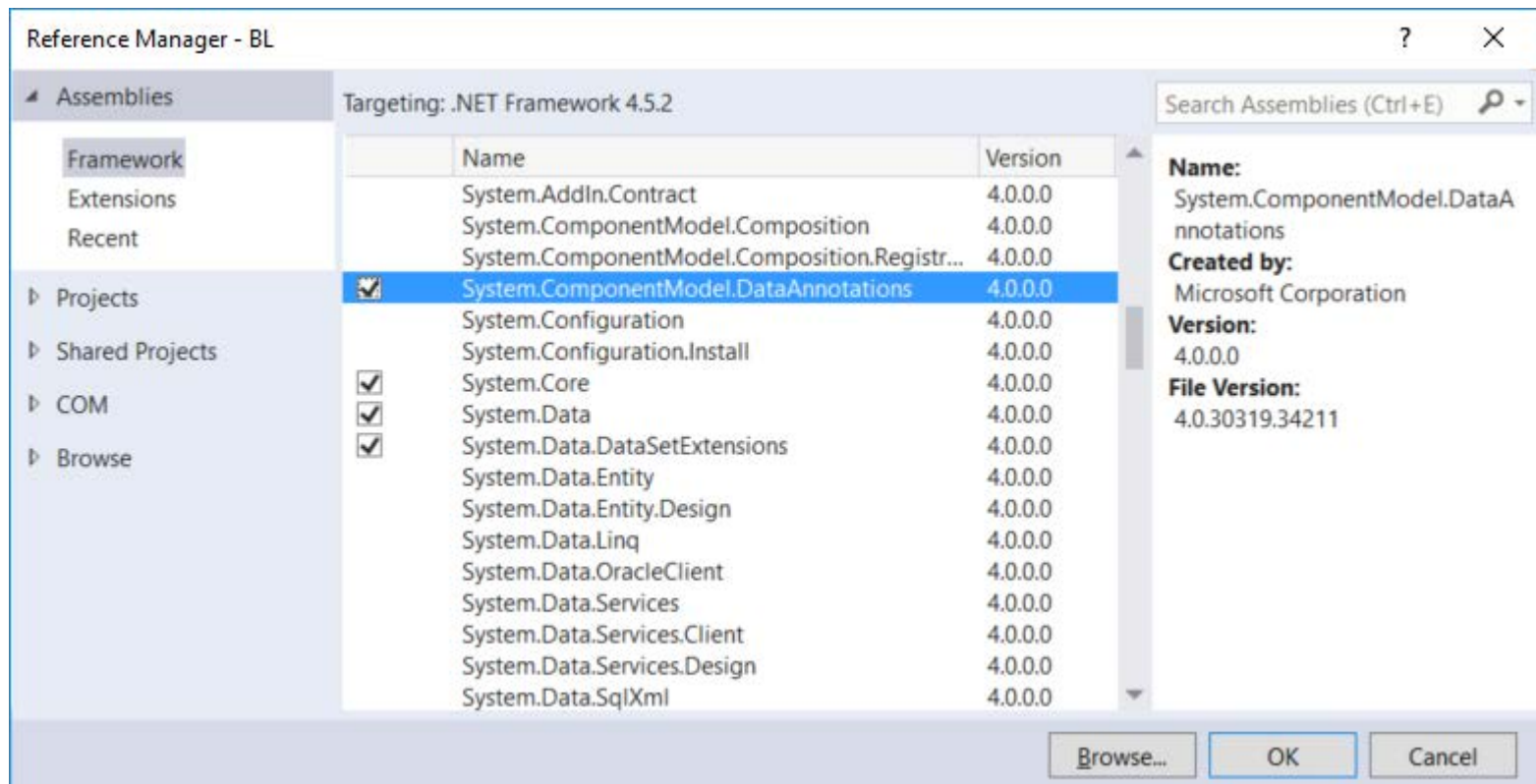

Validatie controle

Oefening

- Zorg dat in BL telkens de validatie-definities van een domein model gecontroleerd worden alvorens de repository (DAL) wordt aangesproken
 - gebruik klasse 'Validator' ('System.ComponentModel.DataAnnotations')
 - gebruik de methode 'ValidateObject' om een 'Ticket' te valideren
 - gebruik de methode 'TryValidateObject' om een 'TicketResponse' te valideren


project 'BL'

- Reference toevoegen:



'TicketManager.cs'

```
...  
using System.ComponentModel.DataAnnotations;  
  
namespace SC.BL  
{  
    public class TicketManager : ITicketManager  
    {  
        ...  
        private Ticket AddTicket(Ticket ticket)  
        {  
            this.Validate(ticket);  
            return repo.CreateTicket(ticket);  
        }  
  
        public void ChangeTicket(Ticket ticket)  
        {  
            this.Validate(ticket);  
            repo.UpdateTicket(ticket);  
        }  
  
        ...  
        public TicketResponse AddTicketResponse(int ticketNumber, string response  
                                                , bool isClientResponse)  
        {  
            ...  
            this.Validate(newTicketResponse);  
            this.Validate(ticketToAddResponseTo);  
  
            repo.CreateTicketResponse(newTicketResponse);  
            repo.UpdateTicket(ticketToAddResponseTo);  
  
            ...  
        }  
    }  
}
```



'TicketManager.cs'



'TryValidateObject' maakt het mogelijk om in debug (via variabele 'errors') alle ongeldige validaties te bekijken!

```
private void Validate(Ticket ticket)
{
    List<ValidationResult> errors = new List<ValidationResult>();

    bool valid = Validator.TryValidateObject(ticket, new ValidationContext(ticket)
                                                , errors, validateAllProperties: true);

    if (!valid)
        throw new ValidationException("Ticket not valid!");
}
```

```
private void Validate(TicketResponse response)
{
    Validator.ValidateObject(response, new ValidationContext(response)
                                , validateAllProperties: true);
}
```

```
}
}
```

'ValidateObject' gooit een 'ValidationException' met daarin een 'ValidationResult' van de eerste niet geldige validatie!

Oefening

- Test de validatie-definities!
 - We voorzien hiervoor in het project 'UI-CA' een apart test programma zodat we alle mogelijke validaties kunnen simuleren
 - Maak een klasse 'ProgramForTesting' met een Main-methode, stel in als startup-object van het project
 - Maak gebruik van 'Validator.TryValidateObject' om de validation-errors te kunnen bekijken

OPGELET: reference nodig naar
System.ComponentModel.DataAnnotations

Oefening

- Voorzie een 'Ticket' met volgende properties en valideer:
 - TicketNumber: 1
 - AccountId: 1
 - Text: ""
 - DateOpened: DateTime.Now
 - State: Open

OPGELET:

Om de individuele validatie-errors te kunnen bekijken plaats je een **breakpoint** op '}' van de Main-methode

'ProgramForTesting.cs'

```
...
using SC.BL;

namespace SC.UI.CA
{
    class ProgramForTesting
    {
        static void Main(string[] args)
        {
            Ticket t1 = new Ticket() {
                TicketNumber = 1, AccountId = 1, Text = "",
                State = TicketState.Open, DateOpened = DateTime.Now
            };
            var errors = new List<ValidationResult>();
            Validator.TryValidateObject(t1, new ValidationContext(t1), errors,
                                      validateAllProperties: true);
        }
    }
}
```

Locals		
Name	Value	
errors	Count = 1	
[0]	{Het veld Text is vereist.}	
ErrorMessage	"Het veld Text is vereist."	Q
MemberNames	{string[1]}	
[0]	"Text"	Q

Oefening

- Voorzie een 'HardwareTicket' met volgende properties en valideer:
 - TicketNumber: 2
 - AccountId: 1
 - DeviceName: "LPT-9876"
 - Text: "text"
 - DateOpened: DateTime.Now
 - State: Open

OPGELET:

Om de individuele validatie-errors te kunnen bekijken plaats je een **breakpoint** op '}' van de Main-methode

'ProgramForTesting.cs'

```
...
using SC.BL;

namespace SC.UI.CA
{
    class ProgramForTesting
    {
        static void Main(string[] args)
        {
            Ticket t2 = new HardwareTicket() {
                TicketNumber = 2, AccountId = 1, DeviceName = "LPT-9876", Text = "text",
                State = TicketState.Open, DateOpened = DateTime.Now
            };
            var errors = new List<ValidationResult>();
            Validator.TryValidateObject(t2, new ValidationContext(t2), errors,
                                     validateAllProperties: true);
        }
    }
}
```

Locals		
Name	Value	
errors	Count = 1	
[0]	{Het veld DeviceName moet overeenkomen met de reguliere expressie ^(PC-)[0-9]+.}	
ErrorMessage	"Het veld DeviceName moet overeenkomen met de reguliere expressie ^(PC-)[0-9]+."	Q
MemberNames	{string[1]}	
[0]	"DeviceName"	Q

User-defined validatie

User-defined

- Enkel de validatie-logica moet m.b.v. de interface 'IValidatableObject' uitgewerkt worden op de domeinmodellen

OPGELET:

Alle projecten die een dependency hebben op een type dat de interface 'IValidatableObject' implementeert, moeten ook een reference hebben naar
'System.ComponentModel.DataAnnotations'

Oefening

- Werk volgende validatie-logica uit voor 'TicketResponse'
 - 'Date' moet groter zijn dan 'DateOpened' van het ticket
 - gebruik interface 'IValidatableObject' (implementeer de interface expliciet)

'TicketResponse.cs'

```
...
using System.ComponentModel.DataAnnotations;

namespace SC.BL.Domain
{
    public class TicketResponse : IValidatableObject
    {
        public int Id { get; set; }
        [Required]
        public string Text { get; set; }
        public DateTime Date { get; set; }
        public bool IsClientResponse { get; set; }

        [Required]
        public Ticket Ticket { get; set; }

        IEnumerable<ValidationResult> IValidatableObject.Validate(ValidationContext validationContext)
        {
            List<ValidationResult> errors = new List<ValidationResult>();

            if (Date <= Ticket.DateOpened)
            {
                errors.Add(new ValidationResult("Can't be before the date the ticket is created!"
                                                , new string[] { "Date", "Ticket.DateOpened" }));
            }

            return errors;
        }
    }
}
```

Oefening

- Test de user-defined validatie!
 - Voorzie een 'TicketResponse' met volgende properties:
 - Id: 1
 - Text: "response"
 - IsClientResponse: true
 - DateOpened: 1/1/2014
 - en een bijhorend 'Ticket' met volgende properties:
 - TicketNumber: 3
 - AccountId: 1
 - Text: "text"
 - DateOpened: 1/1/2015
 - State: Open

'ProgramForTesting.cs'

```
...
using System.ComponentModel.DataAnnotations;
using SC.BL.Domain;

namespace SC.UI.CA
{
    class ProgramForTesting
    {
        static void Main(string[] args)
        {
            TicketResponse tr = new TicketResponse() {
                Id = 1, Text = "response", IsClientResponse = true, Date = new DateTime(2014, 1, 1),
                Ticket = new Ticket() {
                    TicketNumber = 3, AccountId = 1, Text = "text", State = TicketState.Open,
                    DateOpened = new DateTime(2015, 1, 1)
                }
            };

            var errors = new List<ValidationResult>();
            Validator.TryValidateObject(tr, new ValidationContext(tr), errors, true);

            Console.ReadLine();
        }
    }
}
```

Locals		
Name	Value	
errors	Count = 1	
[0]	{ Can't be before the date the ticket is created! }	
ErrorMessage	"Can't be before the date the ticket is created!"	Q
MemberNames	{ string[1] }	
[0]	"Date"	Q

Oefening

- Test nu ook alle validatie-regels via de 'Console Applicatie', door nieuwe tickets en ticketresponses aan te maken die niet voldoen aan de voorwaarden!

OPGELET:

- Stel het startup-object terug in op 'Program'
- Alle projecten die een dependency hebben op een type dat de interface 'IValidatableObject' implementeert moeten een reference hebben op 'System.ComponentModel.DataAnnotations'
- Zorg dat in 'BL.TicketManager' voor de validatie-controle van 'Ticket' en 'TicketResponse' gebruik gemaakt wordt van 'Validator.TryValidateObject' om de validatie-errors te kunnen bekijken m.b.v. een breakpoint