

Praxisarbeit Modul 322

CRM App

PROJEKT-BERICHT 25.06.2024

ALAIN BOCHERENS

Dokumentinformationen

Auftraggeber	Lukas Müller (IBZ Basel)
Projektleiter	Alain Bocherens
Autor	Alain Bocherens
Ausgabedatum	25 Juni 2024
Status	Abgeschlossen
Version	0.1

1 Inhalt

Dokumentinformationen	1
Teil 1: Ablauf und Umfeld	4
1. Einleitung	4
2. Aufgabenstellung	5
2.1 Ausgangslage.....	5
2.2 Detaillierte Aufgabenstellung.....	5
2.3 Mittel und Methoden	7
3. Projektmethodik.....	8
3.1 Vorgehensmodell	8
3.1.1 IPERKA.....	8
Teil 2: Projektdokumentation	9
4. Informieren	9
5. Planen	10
5.1 Aufwandschätzung	10
5.2 ERM.....	11
5.3 UML Use Case	12
5.4 Mockups	13
5.4.1 MainWindow.....	13
5.4.2 CompanyWindow.....	15
5.4.3 ContactWindow.....	16
6. Entscheiden	17
7. Realisieren.....	18
7.1 Datenbank / Models.....	18
7.1.1 Models	18
7.1.2 DbContext	21

7.1.3	Services / CRUD.....	22
7.2	MainWindow / MainViewModel	25
7.2.1	MainWindow.....	25
7.2.2	MainViewModel.....	27
7.2.3	Resultat	29
7.3	CompanyWindow / ComanyViewModel	30
7.3.1	CompanyWindow.....	30
7.3.2	CompanyViewModel	31
7.3.3	Resultat	31
7.4	ContactWindow / ContactViewModel.....	32
7.4.1	ContactWindow.....	32
7.4.2	ContactViewModel.....	33
7.4.3	Resultat	33
8.	<i>Kontrollieren</i>	34
8.1	Testkonzept.....	34
8.2	Testprotokoll	34
9.	<i>Auswerten</i>	35
9.1	Reflexion.....	35

Teil 1: Ablauf und Umfeld

1. Einleitung

Die Praxisarbeit wird im Rahmen des Moduls 322 durchgeführt. Die Durchführung des Projekts beginnt am 18.06.2024 und wird am 25.06.2024 abgeschlossen.

Die Arbeitsbereiche des Projekts sind die folgenden:

- Applikationsentwicklung OO
- MS Windows
- C#

Der Titel des Projekts lautet «CRM App»

2. Aufgabenstellung

2.1 Ausgangslage

Kundenbeziehungen sind eine gute Sache. Für die Verwaltung von Kunden und Ansprechpersonen wird oft eine CRM (Customer-Relationship-Management) Anwendung eingesetzt. Eine bereits existierende Softwarelösung soll nun mit einem kleinen CRM-Modul ausgebaut werden. In Vorabgesprächen wurde festgelegt, dass die Benutzeroberfläche strikte nach den Regeln der harmonischen Dialoggestaltung, intuitiv und benutzerfreundlicher aufgebaut werden muss.

2.2 Detaillierte Aufgabenstellung

Sie erhalten den Auftrag für dieses CRM-Modul eine Benutzeroberfläche zu entwerfen, welche die Grundprinzipien zur Dialoggestaltung nach ISO 9241-110 berücksichtigt, sodass eine optimierte anwenderfreundliche Bedienung gewährleistet ist.

In dieser CRM-Anwendung sind folgende Elemente vorzusehen:

- Firma mit Namen, Strasse, PLZ, Ort und Tags (z.B. .NET, Java etc.)
- Kontaktperson mit Anrede, Vor-/Nachname, Rolle, E-Mail, Telefon, Handy, Status (Aktiv, Inaktiv).
- Listenanzeige der Firmen mit den Kontaktpersonen und Anzeige der Detailinformationen.
- Schaltflächen, um neue Firma oder Ansprechpartner hinzufügen zu können.
- Beim Hinzufügen einer Firma oder Ansprechpartners muss eine Speichern- und Abbrechen-Schaltfläche vorhanden sein.
- Ansprechpartner dürfen aufgrund einer langfristigen Datenhaltung nicht gelöscht, sondern ausschliesslich mit Status Inaktiv markiert werden.

Aufgabe Design

Skizzieren Sie für dieses CRM-Modul eine vollständige Benutzeroberfläche mit allen erforderlichen Elementen aus der Auftragsbeschreibung. Erstellen Sie die Skizze mit einem Mockup Tool (Sketch-Flow etc.), welches Sie im Unterricht verwendet haben. Achten Sie, dass die Steuerelemente gut und eindeutig erkennbar sind und der Dialog nach den ergonomischen Richtlinien aufgebaut ist. Prüfen Sie auch, dass die Grundsätze der Dialoggestaltung aus der Norm EN ISO 9241-110 strikte eingehalten sind.

Abzugebende Lösungselemente: A21-Design.png (Designentwurf als JPG oder PNG-Datei.)

Aufgabe Realisierung

Erstellen Sie eine WPF-Anwendung und implementieren Sie Ihr Design aus der Entwurfsphase (Aufgabe Design). Die Benutzeroberfläche muss vollständig inkl. sämtlichen Funktionen realisiert werden. Zudem soll die Oberfläche auch eine spätere Spracherweiterung ermöglichen und aus wartungstechnischen Gründen keine hardcodierte Ressourcen enthalten. Um die Darstellung von der Logik der Benutzerschnittstelle zu trennen ist das MVVM-Entwurfsmuster zu verwenden.

2.3 Mittel und Methoden

Entwicklungsumgebung: Microsoft Visual Studio 2022

Programmiersprachen: C#, XAML

Frameworks: .NET 8, WPF

Datenbank: MySQL

Quellcodeverwaltung: GitHub

3. Projektmethodik

3.1 Vorgehensmodell

Das Vorgehensmodell beschreibt die Prozesse und Abfolge von Aktivitäten, die zur Durchführung eines Projektes notwendig sind. Dabei wird das Projekt in verschiedene, strukturierte Abschnitte unterteilt.

3.1.1 IPERKA

Das Vorgehensmodell IPERKA besteht aus den sechs Phasen: Infomieren, Planen, Entscheiden, Realisieren, Kontrollieren und Auswerten. Ich habe mich für dieses Vorgehensmodell entschieden, da es sich sehr gut für kleine und individuelle Projekte eignet. Es ist universell einsetzbar und kann leicht an die spezifischen Anforderungen eines Projektes angepasst werden. Dass ich das IPERKA-Modell bereits aus meiner Ausbildung kenne und bereits mehrere kleinere Projekte damit absolviert habe, hat bei dieser Entscheidung auch einen Einfluss gehabt.

Teil 2: Projektdokumentation

4. Informieren

Die erste Phase von IPERKA ist: Informieren. Diese Phase dient dazu sich über den genauen Auftrag zu informieren und sich einen Überblick darüber zu verschaffen, um festzustellen, ob man alles Nötige hat, um den Auftrag zu vollenden.

Ich habe mir dafür nochmals die genaue Aufgabenstellung der Praxisarbeit angeschaut und bin dann noch die Bewertungskriterien durchgegangen. Danach habe ich eine Checkliste mit den einzelnen Arbeitsschritten gemäss Aufgabenstellung erstellt.

5. Planen

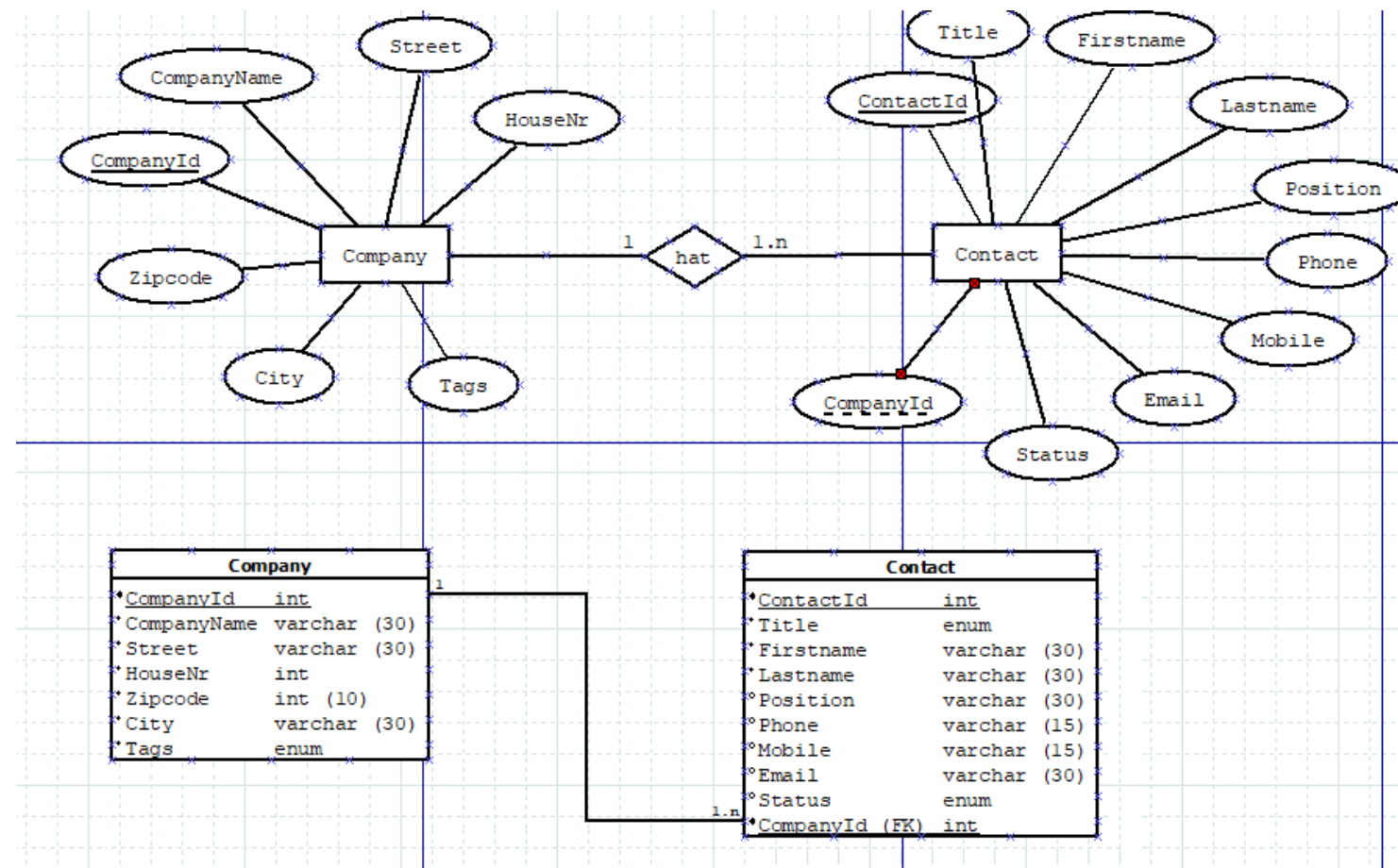
Die zweite Phase von IPERKA ist: Planen. In dieser Phase erstellt man Zeitpläne, Modelle und Diagramme.

5.1 Aufwandschätzung

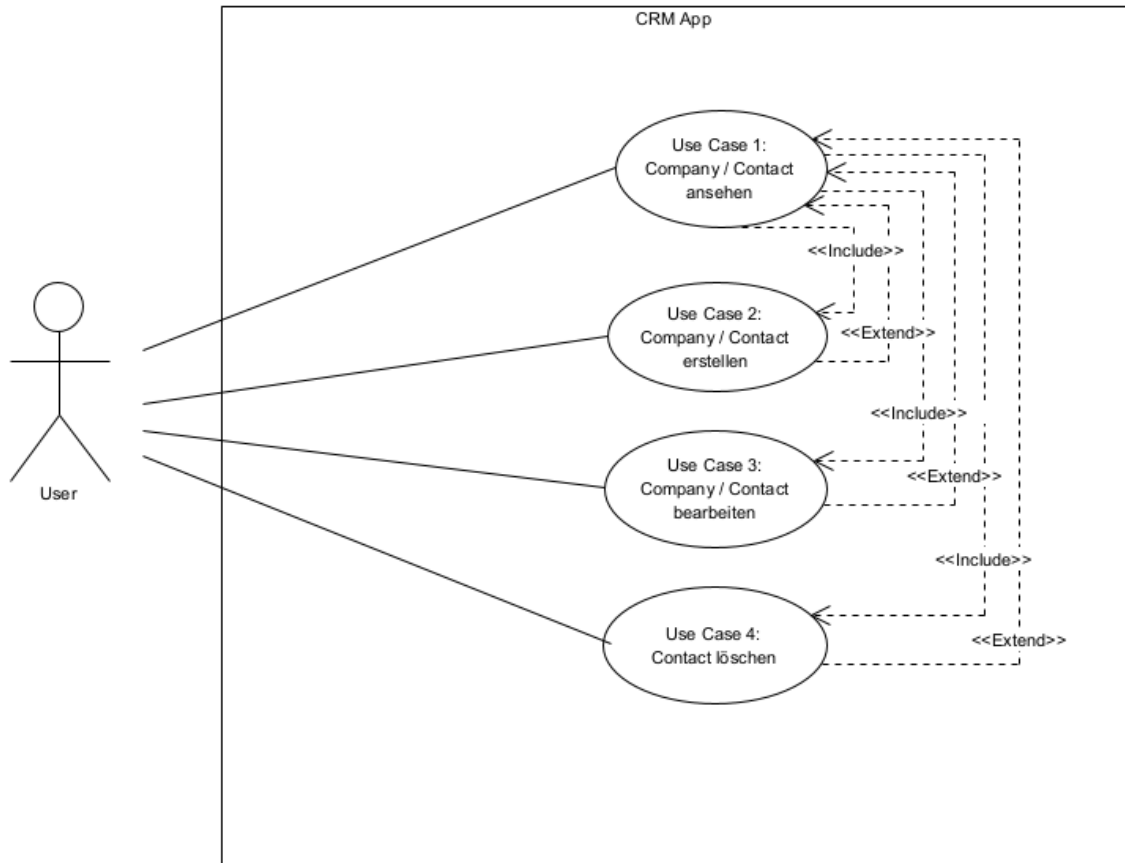
Phase	Aufgaben	Soll	Ist
Informieren	Projektinformationen durchgehen	1	1
	Zwischentotal	1	1
Planen	ERM erstellen	1	1
	UML UseCases erstellen	1	1
	Mockups erstellen	1	1
	Zwischentotal	3	3
Entscheiden	Mockup auswählen	0.5	0.5
	Zwischentotal	0.5	0.5
Realisieren	Datenbank / Models erstellen	4	2
	MainWindow / MainViewModel erstellen	6	15
	CompanyWindow / CompanyViewModel erstellen	4	8
	ContactWindow / ContactViewModel erstellen	4	2
	Zwischentotal	18	27
Testen	Testfälle definieren	1	0
	Testfälle durchführen	1	0
	Zwischentotal	2	0
Auswerten	Ergebnis festhalten	1	1
	Zwischentotal	1	1
Zwischentotal		22.5	32.5
Sonstiges	Dokumentation	4	4
Total		26,5	36.5

5.2 ERM

Mit dem ERM (Entity-Relationship-Modell) werden Entitäten, Attribute und Beziehungen zwischen den Entitäten einer Datenbank grafisch dargestellt. Das ERM wurde mit der Software «Dia» erstellt.



5.3 UML Use Case



Use Case Nummer	Beschreibung
Use Case 1	Der Benutzer kann alle Firmen (Company) und Ansprechpersonen (Contact) ansehen.
Use Case 2	Der Benutzer kann neue Firmen und Ansprechpersonen erstellen. Für Use Case 2 muss man Use Case 1 bereits erledigt haben («include»). Der Use Case 2 kann man optional nach dem Use Case 1 erledigen («extend»).
Use Case 3	Der Benutzer Firmen und Ansprechpersonen bearbeiten. Für Use Case 3 muss man Use Case 1 bereits erledigt haben («include»). Der Use Case 3 kann man optional nach dem Use Case 1 erledigen («extend»).
Use Case 4	Der Benutzer kann eine Ansprechperson löschen. Für Use Case 4 muss man Use Case 1 bereits erledigt haben («include»). Der Use Case 4 kann man optional nach dem Use Case 1 erledigen («extend»).

5.4 Mockups

Die Mockups dienen mir als Vorlage für die Darstellung der WPF-Anwendung. Ich habe sie mit der Software «Balsamiq Wireframes» erstellt.

5.4.1 MainWindow

Für das MainWindow habe ich zwei Varianten erstellt.

Variante 1:

CRM-Anwendung

+

Nr ▼	Firma	Strasse	HausNr	PLZ	Ort	Tag	Control
1	Muster AG	Musterstrasse	12	4054	Basel	Java	+
1	Test AG	Musterstrasse	34	8008	Zürich	.NET	+
1	Soundso AG	Musterstrasse	9	4054	Basel	C#	+
1	WieWoWas GmbH	Musterstrasse	90	4054	Basel	PHP	+
1	Alanis AG	Musterstrasse	55	4054	Basel	Python	+

Nr ▼	Anrede	Vorname	Nachname	Position	Telefon	Handy	Email	Control
1	Herr	Max	Muster	Projektmanager	0612345678	0798765432	info@test.ch	🗑

In der ersten Variante gibt es über der Tabelle einen Button mit dem neuen Firmen erstellt werden. In der Tabelle der Firmen befindet sich rechts eine Spalte mit dem Button zum Erstellen der Ansprechpersonen. Die Tabelle Contact wird als «nested grid» zu jedem Firmeneintrag implementiert. Rechts in der Tabelle Contact befindet sich eine Spalte mit dem Button zum Löschen von Ansprechpersonen. Die Bearbeitung wird in der Zeile stattfinden.

Variante 2:

CRM-Anwendung							
Hinzufügen Bearbeiten Löschen							
Nr ▼	Firma	Strasse	HausNr	PLZ	Ort	Tag	
1	Muster AG	Musterstrasse	12	4054	Basel	Java	
1	Test AG	Musterstrasse	34	8008	Zürich	.NET	
1	Soundso AG	Musterstrasse	9	4054	Basel	C#	
1	WieWoWas GmbH	Musterstrasse	90	4054	Basel	PHP	
1	Alanis AG	Musterstrasse	55	4054	Basel	Python	

Nr ▼	Anrede	Vorname	Nachname	Position	Telefon	Handy	Email
1	Herr	Max	Muster	Projektmanager	0612345678	0798765432	info@test.ch

Bei der zweiten Variante gibt es keine Spalte mit Buttons, sondern diese befinden sich in einer Toolbar über der Tabelle. Die Bearbeitung wird in einem separaten Fenster stattfinden.

5.4.2 CompanyWindow

The screenshot shows a window titled "Company Form". Inside the window, there are six input fields arranged in two columns. The left column contains "Company:", "Street:", and "Zipcode:". The right column contains "Tag:", "HouseNr:", and "City:". The "Tag:" field is a ComboBox with "ComboBox" selected. At the bottom right of the form area, there are two buttons: "Save" and "Cancel".

Das CompanyWindow beinhaltet ein einfaches Formular mit den Buttons Save und Cancel.

5.4.3 ContactWindow

The screenshot shows a window titled "Contact Form". Inside the window, there are several input fields and two buttons. The fields are arranged in two columns. The first column contains "Title:" with a dropdown menu showing "ComboBox", "Firstname:" with a text box, "Phone:" with a text box, and "Email" with a text box. The second column contains "Lastname:" with a text box, "Mobile" with a text box, and "Position:" with a text box. At the bottom right of the form area, there are two buttons: "Save" and "Cancel".

Das ContactWindow beinhaltet ein einfaches Formular mit den Buttons Save und Cancel.

6. Entscheiden

Ich habe mich beim MainWindow schlussendlich für die Variante 1 entschieden, da mir das Design mit den Buttons neben den Einträgen besser gefällt und ich somit auch ein XAML-Window weniger brauchen werde.

CRM-Anwendung

+

Nr	Firma	Strasse	HausNr	PLZ	Ort	Tag	Control
1	Muster AG	Musterstrasse	12	4054	Basel	Java	+
1	Test AG	Musterstrasse	34	8008	Zürich	.NET	+
1	Soundso AG	Musterstrasse	9	4054	Basel	C#	+
1	WieWoWas GmbH	Musterstrasse	90	4054	Basel	PHP	+
1	Alanis AG	Musterstrasse	55	4054	Basel	Python	+

Nr	Anrede	Vorname	Nachname	Position	Telefon	Handy	Email	Control
1	Herr	Max	Muster	Projektmanager	0612345678	0798765432	info@test.ch	🗑️

7. Realisieren

Die vierte Phase der IPERKA ist: Realisieren. In dieser Phase wird das Projekt umgesetzt.

7.1 Datenbank / Models

Die Datenbank wird als «Code First» mit C# und OR Mapper implementiert.

Anmerkung: Der DbContext und die Services wurden zwar implementiert, aber Aufgrund von Problemen und dem Zeitmangel wurden in den ViewModels keine CRUD-Methoden für die Datenbank implementiert.

7.1.1 Models

Als erstes habe ich die Modelle für die Datenbank erstellt. Diese dienen als Vorlage wie jede Tabelle in der Datenbank erstellt werden soll.

Ich habe die abstrakte Klasse «BaseModel» als Basismodell erstellt, welches an alle anderen Modelle die Eigenschaft «Id» als «Primary Key» übergibt.

```
public abstract class BaseModel
{
    [Key, DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    7 Verweise
    public int Id { get; set; }
}
```

Als nächstes habe ich die Modellklasse «CompanyModel» erstellt, die als Vorlage für die Datenbanktabelle «Company» dient.

```
[Table("Company")]

public class CompanyModel : BaseModel
{
    [Required, StringLength(30)]
    2 references
    public string Company { get; set; }

    [Required, StringLength(30)]
    2 references
    public string Street { get; set; }

    [Required]
    2 references
    public string HouseNr { get; set; }

    [Required]
    2 references
    public int Zipcode { get; set; }

    [Required, StringLength(30)]
    2 references
    public string City { get; set; }

    [Required]
    2 references
    public Tags Tag { get; set; }

    3 references
    public List<ContactModel> Contacts { get; set; } = [];
}
```

Für das Feld Tag habe ich das Enum «Tags» erstellt:

```
// Enum für Firmenspezialisierung
7 references
public enum Tags
{
    PHP = 1,

    Java = 2,

    Python = 3
}
```

Danach habe ich die Modelklasse «ContactModel» erstellt, die als Vorlage für die Datenbanktabelle «Contact» dient.

```
[Table("Contact")]
26 references
public class ContactModel : BaseModel
{
    [Required]
    4 references
    public Title Title { get; set; }

    [Required, StringLength(30)]
    4 references
    public string Firstname { get; set; }

    [Required, StringLength(30)]
    4 references
    public string Lastname { get; set; }

    [StringLength(15)]
    4 references
    public string? Phone { get; set; }

    [StringLength(15)]
    4 references
    public string? Mobile { get; set; }

    [Required, StringLength(30), EmailAddress]
    4 references
    public string Email { get; set; }

    [Required]
    5 references
    public Status Status { get; set; } = Status.Active;

    [Required, ForeignKey(nameof(CompanyId))]
    1 reference
    public CompanyModel Company { get; set; }

    1 reference
    public int CompanyId { get; set; }
}
```

Für die Felder Title (Anrede) und Status habe ich jeweils ein Enum erstellt:

<pre>// Anrede 5 references public enum Title { Herr = 1, Frau = 2 }</pre>	<pre>public enum Status { // Aktive Ansprechpersonen Active = 1, // Inaktive Ansprechpersonen Inactive = 2 }</pre>
---	---

7.1.2 DbContext

Der «DbContext» oder auch Datenbankkontext ist die Verbindung zwischen der Datenbank und der Anwendung. Es ist verantwortlich für das Mapping der Datenbanktabellen und erleichtert den Zugriff auf die Datenbank. Ausserdem wird es benötigt, um Daten in der Datenbank abzufragen, hinzuzufügen, zu ändern und zu löschen.

Ich habe meinen «DbContext» AppDbContext genannt.

```
public class AppDbContext : DbContext
```

Mit den «DbSet»-Properties ermögele ich den Zugriff auf die Datenbanktabelle der entsprechenden Entitätsklasse, um Abfragen und Änderungen vorzunehmen.

```
public DbSet<CompanyModel> Companies { get; set; }
public DbSet<ContactModel> Contacts { get; set; }
```

Mit der Methode «OnModelCreating» wird die Datenbank mit den Modellen als Vorlage erstellt.

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<CompanyModel>();
    modelBuilder.Entity<ContactModel>().Navigation(nameof(ContactModel.Company)).AutoInclude();

    modelBuilder.ApplyConfigurationsFromAssembly(typeof(AppDbContext).Assembly);
}
```

Für den Verbindungsaufbau mit dem MySQL-Server habe ich die Methode «OnConfiguring» erstellt.

```
protected override void OnConfiguring(
    DbContextOptionsBuilder optionsBuilder)
{
    optionsBuilder.UseMySQL("Data Source=crm.db", MySQLServerVersion.LatestSupportedServerVersion);
    optionsBuilder.EnableSensitiveDataLogging();
}
```

7.1.3 Services / CRUD

CRUD steht für die Operationen «Create» (Erstellen), «Read» (Lesen), «Update» (Ändern) und «Delete» (Löschen). Mit Create werden Datenbankeinträge erstellt. Mit Read werden Daten aus der Datenbank ausgelesen. Mit Update werden Daten in der Datenbank angepasst oder geändert. Und mit Delete werden Datenbankeinträge gelöscht.

Für jedes Modell wird eine Service-Klasse erstellt, um die CRUD-Methoden zu implementieren. Ich habe die abstrakte Klasse «BaseService» als Basisklasse für die anderen Services erstellt.

```
public abstract class BaseService<T>(AppDbContext dbContext) where T: BaseModel
{
    24 Verweise
    public AppDbContext DbContext { get; } = dbContext;

    3 Verweise
    public abstract Task<List<T>> GetAllAsync();
    3 Verweise
    public abstract Task<T?> CreateAsync(T t); // C
    3 Verweise
    public abstract Task<T?> GetSingleAsync(int id); // R
    3 Verweise
    public abstract Task<bool> UpdateAsync(T t); // U
    3 Verweise
    public abstract Task<bool> DeleteAsync(T t); // D
}
```

CompanyService:

```
public class ComapnyService(AppDbContext dbContext) : BaseService<CompanyModel>(dbContext)
{
    1 reference
    public override async Task<CompanyModel?> CreateAsync(CompanyModel t)
    {
        DbContext.Companies.Add(t);

        return await DbContext.SaveChangesAsync() > 0 ? t : null;
    }

    1 reference
    public override async Task<bool> DeleteAsync(CompanyModel t)
    {
        DbContext.Companies.Remove(t);

        return (await DbContext.SaveChangesAsync()) > 0;
    }

    1 reference
    public override async Task<List<CompanyModel>> GetAllAsync()
    {
        return await DbContext.Companies.ToListAsync();
    }

    1 reference
    public override async Task<CompanyModel?> GetSingleAsync(int id)
    {
        return await DbContext.Companies.FindAsync(id);
    }

    1 reference
    public override async Task<bool> UpdateAsync(CompanyModel t)
    {
        DbContext.Companies.Update(t);

        return (await DbContext.SaveChangesAsync()) > 0;
    }
}
```


ContactService:

```
public class ContactService(AppDbContext dbContext) : BaseService<ContactModel>(dbContext)
{
    1 reference
    public override async Task<bool> DeleteAsync(ContactModel t)
    {
        dbContext.Contacts.Remove(t);
        return (await dbContext.SaveChangesAsync()) > 0;
    }

    1 reference
    public override async Task<List<ContactModel>> GetAllAsync()
    {
        return await dbContext.Contacts.ToListAsync();
    }

    1 reference
    public override async Task<ContactModel?> GetSingleAsync(int id)
    {
        return await dbContext.Contacts.FindAsync(id);
    }

    1 reference
    public override async Task<ContactModel?> CreateAsync(ContactModel t)
    {
        dbContext.Contacts.Add(t);

        return await dbContext.SaveChangesAsync() > 0 ? t : null;
    }

    1 reference
    public override async Task<bool> UpdateAsync(ContactModel t)
    {
        dbContext.Contacts.Update(t);

        return (await dbContext.SaveChangesAsync()) > 0;
    }
}
```

7.2 MainWindow / MainViewModel

7.2.1 MainWindow

Für die Tabellen im MainWindow habe ich die GridControl-Komponente von DevExpress verwendet.

Grid «Company»:

```
<dxg:GridControl Name="gridControl"
    ItemsSource="{Binding Companies}"
    AutoGenerateColumns="None" Cursor="AppStarting">
    <dxg:GridControl.Columns>
        <dxg:GridColumn FieldName="Company" Header="Company Name" AllowEditing="True"/>
        <dxg:GridColumn FieldName="Street" Header="Street" AllowEditing="True"/>
        <dxg:GridColumn FieldName="HouseNr" Header="House Number" AllowEditing="True"/>
        <dxg:GridColumn FieldName="Zipcode" Header="Zipcode" AllowEditing="True"/>
        <dxg:GridColumn FieldName="City" Header="City" AllowEditing="True"/>
        <dxg:GridColumn FieldName="Tag" Header="Tag" AllowEditing="True"/>
        <dxg:GridColumn Header="Actions">
            <dxg:GridColumn.CellTemplate>
                <DataTemplate>
                    <Button Content="New Contact"
                        Command="{Binding DataContext.OpenNewContactCommand, RelativeSource={RelativeSource Self}}"
                        CommandParameter="{Binding}" />
                </DataTemplate>
            </dxg:GridColumn.CellTemplate>
        </dxg:GridColumn>
    </dxg:GridControl.Columns>
</dxg:GridControl>
```

Nested Grid «Contact»:

```

<dxg:GridControl.DetailDescriptor>
  <dxg:DataControlDetailDescriptor ItemsSourcePath="Contacts">
    <dxg:GridControl AutoGenerateColumns="None">
      <dxg:GridControl.Columns>
        <dxg:GridColumn FieldName="Title" Header="Title" AllowEditing="True"/>
        <dxg:GridColumn FieldName="Firstname" Header="First Name" AllowEditing="True"/>
        <dxg:GridColumn FieldName="Lastname" Header="Last Name" AllowEditing="True"/>
        <dxg:GridColumn FieldName="Phone" Header="Phone" AllowEditing="True"/>
        <dxg:GridColumn FieldName="Mobile" Header="Mobile" AllowEditing="True"/>
        <dxg:GridColumn FieldName="Email" Header="Email" AllowEditing="True"/>
        <dxg:GridColumn FieldName="Status" Header="Status" AllowEditing="True"/>
        <dxg:GridColumn Header="Actions">
          <dxg:GridColumn.CellTemplate>
            <DataTemplate>
              <Button Content="Delete"
                Command="{Binding DataContext.DeleteContactCommand, RelativeSource={RelativeSource Self}}"
                CommandParameter="{Binding}" />
            </DataTemplate>
          </dxg:GridColumn.CellTemplate>
        </dxg:GridColumn>
      </dxg:GridControl.Columns>
    </dxg:GridControl>
  </dxg:DataControlDetailDescriptor>
</dxg:GridControl.DetailDescriptor>

```

7.2.2 MainViewModel

Im MainViewModel habe ich dann im Konstruktor Beispiel-Daten erstellt.

```
public MainViewModel()
{
    // Initialize the Companies collection with some sample data
    Companies = new ObservableCollection<CompanyModel>
    {
        new CompanyModel
        {
            Company = "Company A",
            Street = "Main St",
            HouseNr = "123",
            Zipcode = 12345,
            City = "Tech City",
            Tag = Tags.Java,
            Contacts = new List<ContactModel>
            {
                new ContactModel
                {
                    Title = Title.Frau,
                    Firstname = "Alice",
                    Lastname = "Smith",
                    Phone = "123-4567",
                    Mobile = "123-4567",
                    Email = "alice@companya.com",
                    Status = Status.Active
                },
                new ContactModel
                {
                    Title = Title.Herr,
                    Firstname = "Bob",
                    Lastname = "Johnson",
                    Phone = "234-5678",
                    Mobile = "234-5678",
                    Email = "bob@companya.com",
                    Status = Status.Active
                }
            }
        }
    },
}
```

Und ich habe die Methoden für die Buttons implementiert:

```
OpenNewCompanyCommand = new RelayCommand(_ => OpenNewCompany());
OpenNewContactCommand = new RelayCommand(company => OpenNewContact((CompanyModel)company));
DeleteContactCommand = new RelayCommand(contact => DeleteContact((ContactModel)contact));
}

1 reference
public ICommand OpenNewCompanyCommand { get; }
1 reference
public ICommand OpenNewContactCommand { get; }
1 reference
public ICommand DeleteContactCommand { get; }
```

```
// Opens Company Form
1 reference
private void OpenNewCompany()
{
    CompanyWindow companyWindow = new CompanyWindow(AddNewCompany);
    companyWindow.ShowDialog();
}

// Adds a new Company Entry
1 reference
private void AddNewCompany(CompanyModel newCompany)
{
    if (newCompany != null)
    {
        Companies.Add(newCompany);
    }
}

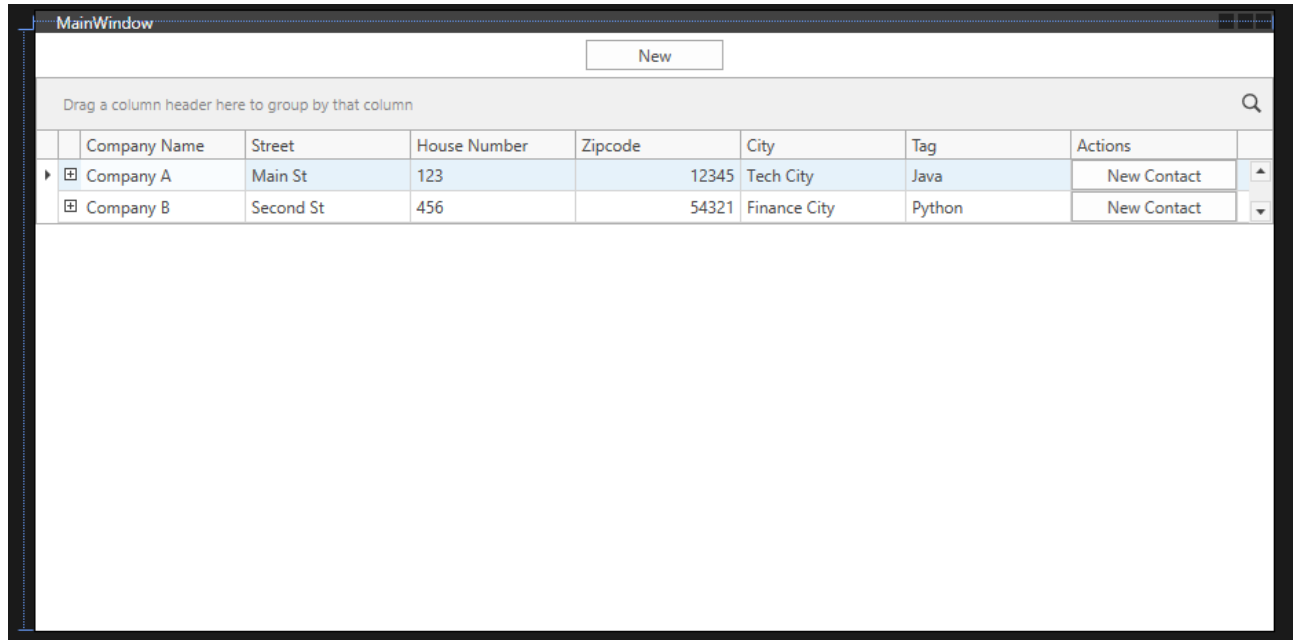
// Opens Contact Form
1 reference
private void OpenNewContact(CompanyModel company)
{
    ContactWindow contactWindow = new ContactWindow(company, AddNewContact);
    contactWindow.ShowDialog();
}

// Adds a new Contact Entry
1 reference
private void AddNewContact(CompanyModel company, ContactModel newContact)
{
    if (company != null && newContact != null)
    {
        company.Contacts.Add(newContact);
    }
}

// Sets the Status of a Contact to inactive
1 reference
private void DeleteContact(ContactModel contact)
{
    if (contact != null)
    {
        contact.Status = Status.Inactive;
    }
}
```

7.2.3 Resultat

Das Result sieht dann so aus:



Drag a column header here to group by that column							Q
	Company Name	Street	House Number	Zipcode	City	Tag	Actions
▶	Company A	Main St	123	12345	Tech City	Java	New Contact
▣	Company B	Second St	456	54321	Finance City	Python	New Contact

PS: Leider konnte ich auf Grund eines Fehlers, den ich nicht ausfindig machen konnte, die Anwendung nicht starten. Deshalb habe ich nur ein Bild der Vorschau.

7.3 CompanyWindow / ComanyViewModel

7.3.1 CompanyWindow

Im CompanyWindow habe ich zuerst ein kleines Grid definiert.

```
<Grid Margin="10">
  <Grid.RowDefinitions>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="*/>
    <RowDefinition Height="Auto"/>
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="Auto"/>
    <ColumnDefinition Width="*/>
    <ColumnDefinition Width="Auto"/>
    <ColumnDefinition Width="*/>
  </Grid.ColumnDefinitions>
```

Danach habe ich die Formular-Elemente und die Buttons implementiert.

```
<TextBlock Text="Company:" Grid.Row="0" Grid.Column="0" VerticalAlignment="Center"/>
<TextBox Text="{Binding Company}" Grid.Row="0" Grid.Column="1" Margin="5"/>

<TextBlock Text="Tag:" Grid.Row="0" Grid.Column="2" VerticalAlignment="Center"/>
<ComboBox ItemsSource="{Binding Tags}" SelectedItem="{Binding Tag}" Grid.Row="0" Grid.Column="3" Margin="5"/>

<TextBlock Text="Street:" Grid.Row="1" Grid.Column="0" VerticalAlignment="Center"/>
<TextBox Text="{Binding Street}" Grid.Row="1" Grid.Column="1" Margin="5"/>

<TextBlock Text="HouseNr:" Grid.Row="1" Grid.Column="2" VerticalAlignment="Center"/>
<TextBox Text="{Binding HouseNr}" Grid.Row="1" Grid.Column="3" Margin="5"/>

<TextBlock Text="Zipcode:" Grid.Row="2" Grid.Column="0" VerticalAlignment="Center"/>
<TextBox Text="{Binding Zipcode}" Grid.Row="2" Grid.Column="1" Margin="5"/>

<TextBlock Text="City:" Grid.Row="2" Grid.Column="2" VerticalAlignment="Center"/>
<TextBox Text="{Binding City}" Grid.Row="2" Grid.Column="3" Margin="5"/>

<StackPanel Grid.Row="4" Grid.ColumnSpan="4" Orientation="Horizontal" HorizontalAlignment="Right" Margin="5">
  <Button Content="Save" Command="{Binding SaveCommand}" Width="75" Height="50" Margin="10"/>
  <Button Content="Cancel" Command="{Binding CancelCommand}" Width="75" Height="50" Margin="5"/>
</StackPanel>
```

7.3.2 CompanyViewModel

Im CompanyViewModel habe ich dann die Methoden für die Buttons Save und Cancel implementiert.

```
_saveCallback = saveCallback;
_closeAction = closeAction;

SaveCommand = new RelayCommand(_ => Save());
CancelCommand = new RelayCommand(_ => Cancel());
}

1 reference
public ICommand SaveCommand { get; }
1 reference
public ICommand CancelCommand { get; }

1 reference
private void Save()
{
    _saveCallback?.Invoke(Company);
    _closeAction?.Invoke();
}

1 reference
private void Cancel()
{
    _closeAction?.Invoke();
}
```

7.3.3 Resultat

Das Result sieht dann so aus:

The screenshot shows a window titled "CompanyWindow" with a grid of input fields. The fields are arranged in two columns. The first column contains "Company:", "Street:", and "Zipcode:". The second column contains "Tag:", "HouseNr", and "City:". At the bottom right of the window, there are two buttons labeled "Save" and "Cancel". The window has a standard Windows-style title bar and a blue border.

PS: Leider konnte ich auf Grund eines Fehlers, den ich nicht ausfindig machen konnte, die Anwendung nicht starten. Deshalb habe ich nur ein Bild der Vorschau.

7.4 ContactWindow / ContactViewModel

7.4.1 ContactWindow

Im CompanyWindow habe ich zuerst ein kleines Grid definiert genau wie im CompanyWindow.

```
<Grid Margin="10">
  <Grid.RowDefinitions>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="Auto"/>
    <ColumnDefinition Width="*/>
    <ColumnDefinition Width="Auto"/>
    <ColumnDefinition Width="*/>
  </Grid.ColumnDefinitions>
```

Danach habe ich die Formular-Elemente und die Buttons implementiert.

```
<TextBlock Text="Title:" Grid.Row="0" Grid.Column="0" VerticalAlignment="Center"/>
<ComboBox ItemsSource="{Binding Titles}" SelectedItem="{Binding Contact.Title}" Grid.Row="0" Grid.Column="1" Margin="5"/>

<TextBlock Text="Firstname:" Grid.Row="1" Grid.Column="0" VerticalAlignment="Center"/>
<TextBox Text="{Binding Contact.Firstname}" Grid.Row="1" Grid.Column="1" Margin="5"/>

<TextBlock Text="Lastname:" Grid.Row="1" Grid.Column="2" VerticalAlignment="Center"/>
<TextBox Text="{Binding Contact.Lastname}" Grid.Row="1" Grid.Column="3" Margin="5"/>

<TextBlock Text="Phone:" Grid.Row="2" Grid.Column="0" VerticalAlignment="Center"/>
<TextBox Text="{Binding Contact.Phone}" Grid.Row="2" Grid.Column="1" Margin="5"/>

<TextBlock Text="Mobile:" Grid.Row="2" Grid.Column="2" VerticalAlignment="Center"/>
<TextBox Text="{Binding Contact.Mobile}" Grid.Row="2" Grid.Column="3" Margin="5"/>

<TextBlock Text="Email:" Grid.Row="3" Grid.Column="0" VerticalAlignment="Center"/>
<TextBox Text="{Binding Contact.Email}" Grid.Row="3" Grid.Column="1" Margin="5"/>

<TextBlock Text="Position:" Grid.Row="3" Grid.Column="2" VerticalAlignment="Center"/>
<TextBox Text="{Binding Contact.Position}" Grid.Row="3" Grid.Column="3" Margin="5"/>

<StackPanel Grid.Row="5" Grid.ColumnSpan="4" Orientation="Horizontal" HorizontalAlignment="Right" Margin="5">
  <Button Content="Save" Command="{Binding SaveCommand}" Width="75" Margin="5"/>
  <Button Content="Cancel" Command="{Binding CancelCommand}" Width="75" Margin="5"/>
</StackPanel>
```

7.4.2 ContactViewModel

Im ContactViewModel habe ich dann die Methoden für die Buttons Save und Cancel implementiert.

```
_saveCallback = saveCallback;
_closeAction = closeAction;

SaveCommand = new RelayCommand(_ => Save());
CancelCommand = new RelayCommand(_ => Cancel());
}

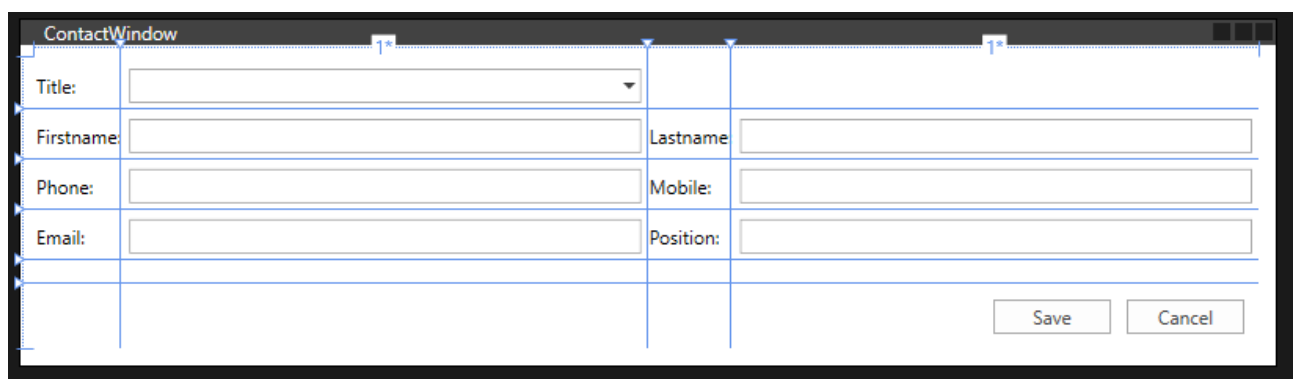
1 reference
public ICommand SaveCommand { get; }
1 reference
public ICommand CancelCommand { get; }

1 reference
private void Save()
{
    _saveCallback?.Invoke(_company, Contact);
    _closeAction?.Invoke();
}

1 reference
private void Cancel()
{
    _closeAction?.Invoke();
}
```

7.4.3 Resultat

Das Resultat sieht dann so aus:



The screenshot shows a window titled "ContactWindow" containing a form. The form has the following fields:

- Title: A dropdown menu.
- Firstname: A text input field.
- Lastname: A text input field.
- Phone: A text input field.
- Mobile: A text input field.
- Email: A text input field.
- Position: A text input field.

At the bottom right of the form, there are two buttons: "Save" and "Cancel".

PS: Leider konnte ich auf Grund eines Fehlers, den ich nicht ausfindig machen konnte, die Anwendung nicht starten. Deshalb habe ich nur ein Bild der Vorschau.

8. Kontrollieren

Die fünfte Phase von IPERKA ist: Kontrollieren. In dieser Phase findet das Testing statt und es wird überprüft, ob das Projekt den Anforderungen entspricht.

8.1 Testkonzept

Bei der Test-Methode habe ich mich für die «Black-Box-Test»-Methode entschieden. In dieser Methode werden die Testfälle anhand des UseCase-Diagramms durchgeführt und dementsprechend nummeriert.

Die Testfälle werden auf der Umgebung, auf der die Anwendung programmiert wurde, getestet.

Die Anwendung wird nur auf der Basis der Benutzbarkeit getestet. Performance-Test werden keine durchgeführt.

Anmerkung: Aufgrund eines Problems, das ich nicht lösen konnte, lässt sich die Anwendung nicht starten. Deshalb ist es auch nicht möglich irgendwelche Tests durchzuführen. Ich habe deswegen und auch auf Grund von Zeitmangel kein Testkonzept erstellt.

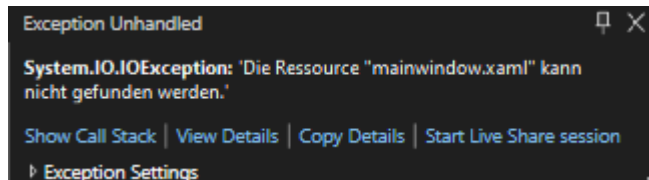
8.2 Testprotokoll

Anmerkung: Aufgrund eines Problems, das ich nicht lösen konnte, lässt sich die Anwendung nicht starten. Weil deshalb nicht getestet werden kann, wurde kein Testkonzept erstellt und keine Tests durchgeführt.

9. Auswerten

9.1 Reflexion

Es wurde alles gemäss Aufgabenstellung und Planung implementiert. Leider konnte ich die Anwendung nicht starten da folgendes Problem aufgetreten ist.



Ich konnte deshalb nicht testen, ob alles funktioniert wie gewünscht. Ich habe auch auf Grund von Zeitmangel schlussendlich kein Exception-Handling mehr machen können und auch für die Speicherung in der Datenbank hat es nicht mehr gereicht. Ich wollte eig. auch noch das Design ausbessern. Zur Implementierung der Sprachauswahl bin ich leider auch nicht mehr gekommen.

Schlussendlich war es zusammen mit dem Stress von der Arbeit im Praktikumsbetrieb und weiteren Praxisarbeiten, die ich parallel zu dieser Arbeit erledigen musste, einfach zu viel und zu wenig Zeit.

Ich hoffe das dies bei der Bewertung berücksichtigt wird.