Ausbaustufe 1 – SWK5 (Abgabe: 13. 11. 2022, 24:00 Uhr)

Sämtliche Komponenten von *CaaS* sind mit den in der Lehrveranstaltung SWK5 behandelten Technologienzu realisieren. In dieser ersten Ausbaustufe benötigen Sie nur .NET und ein Datenbanksystem Ihrer Wahl.

(A1.1) *Datenbank/Datenmodell:* Alle auf der *CaaS*-Plattform erfassten Daten sind in einer relationalen Datenbank zu speichern. Entwerfen Sie ein strukturiertes (normalisiertes) Datenmodell für *CaaS*. Erstellen Sie eine Testdatenbank, welche für die Problemstellung repräsentative und ausreichendumfangreiche Daten enthält.

Stellen Sie auch sicher, dass sämtliche Operationen effizient ausgeführt werden können und IhreImplementierung auch imstande ist, Daten in einem realistischen Umfang zu speichern. Konkret sollte ihr System die Daten von zumindest zwei Shop-Betreibern speichern. Jeder Shop-Betreiberbietet zumindest 100 Produkte zum Kauf an und hat zumindest 100 Kund*innen. Jede Kund*in hatdurchschnittlich bereits 10 Bestellungen, die im Mittel 3 Produkte umfasst, getätigt. Bei mindestens 10 Kund*innen ist ein Warenkorb offen.

Verwenden Sie zum automatisierten Testen und für den Produktivbetrieb unterschiedliche Instanzen Ihrer Datenbank.

(A1.2) *Datenzugriffsschicht:* Überlegen Sie sich für die Entitäten Ihres Datenmodells (*Produkte, Warenkörbe, Kund*innen-Stammdaten,* etc.) die erforderlichen Zugriffsfunktionen und fassen Sie diese zu Interfaces zusammen. Die Geschäftslogik soll ausschließlich von diesen Interfaces abhängig sein.

Entwickeln Sie ein geeignetes Format für die Repräsentation der Daten. Definieren Sie dazu einfache Klassen (die primär aus Konstruktoren und Properties bestehen), welche zum Transport der Daten dienen. Diese Klassen werden häufig als *Domänenklassen* oder *Datentransferklassen* bezeichnet. Im Wesentlichen werden Sie für jede Entität eine entsprechende Domänenklasse benötigen. Die Domänenklassen werden in den Interface-Methoden der Datenzugriffsschicht als Para-metertypen verwendet und müssen daher auch der Geschäftslogik bekannt gemacht werden.

Implementieren Sie die Datenzugriffsschicht auf Basis von ADO.NET. Der Einsatz eines OR- Mapping-Werkzeugs (NHibernate, Entity Framework etc.) ist nicht gestattet.

(A1.3) *Unit-Tests:* Testen Sie die Datenzugriffsschicht ausführlich, indem Sie eine umfangreiche Unit- Test-Suite erstellen. Achten Sie auf eine möglichst große Testabdeckung. Erstellen Sie eher mehr,dafür aber kompakte Unit-Tests, die nach Möglichkeit nur einen Aspekt der Funktionalität über- prüfen. Die Wahl des Testframeworks steht Ihnen frei.

(A1.4) *Dokumentation:* Erstellen Sie eine übersichtliche, gut strukturierte Dokumentation. Ihre Dokumentation sollte zumindest das Datenbankmodell und die Struktur der Datenzugriffsschicht enthalten. Vermeiden Sie dabei die reine Aufzählung der verfügbaren Klassen, Methoden, etc. Dokumentieren und begründen Sie Abwägungen und Entscheidungen, die Sie im Rahmen der Entwicklung getroffen haben.

Ausbaustufe 2 – SWK5

(Abgabe: 22. 12. 2022, 24:00 Uhr)

(A2.1) *CaaS.Core:* Implementieren Sie auf Basis der in Ausbaustufe 1 entwickelten Datenzugriffsschicht die gesamte Funktionalität der Komponente *CaaS.Core*. Überlegen Sie sich, wie die Klienten auf die (Geschäfts-)Logik zugreifen sollen. Fassen Sie die erforderlichen Funktionen zu logischen Grup- pen zusammen und definieren Sie für jede Gruppe ein Interface. Alle REST-Services sollen (sta- tisch) ausschließlich von diesen Geschäftslogik-Interfaces abhängig sein.

Entwerfen Sie die Klassen der Geschäftslogik mit besonderer Sorgfalt und legen Sie besonderen Wert auf eine flexible Software-Architektur. Sorgen Sie insbesondere dafür, dass Komponenten des Rabatt- und Bezahlsystems einfach ausgetauscht werden können. Implementieren Sie alle Funktionen, die nicht unmittelbar für die Umsetzung der REST-Schnittstelle oder des Web-Clientsbenötigt werden, in der Geschäftslogikkomponente von *CaaS.Core*.

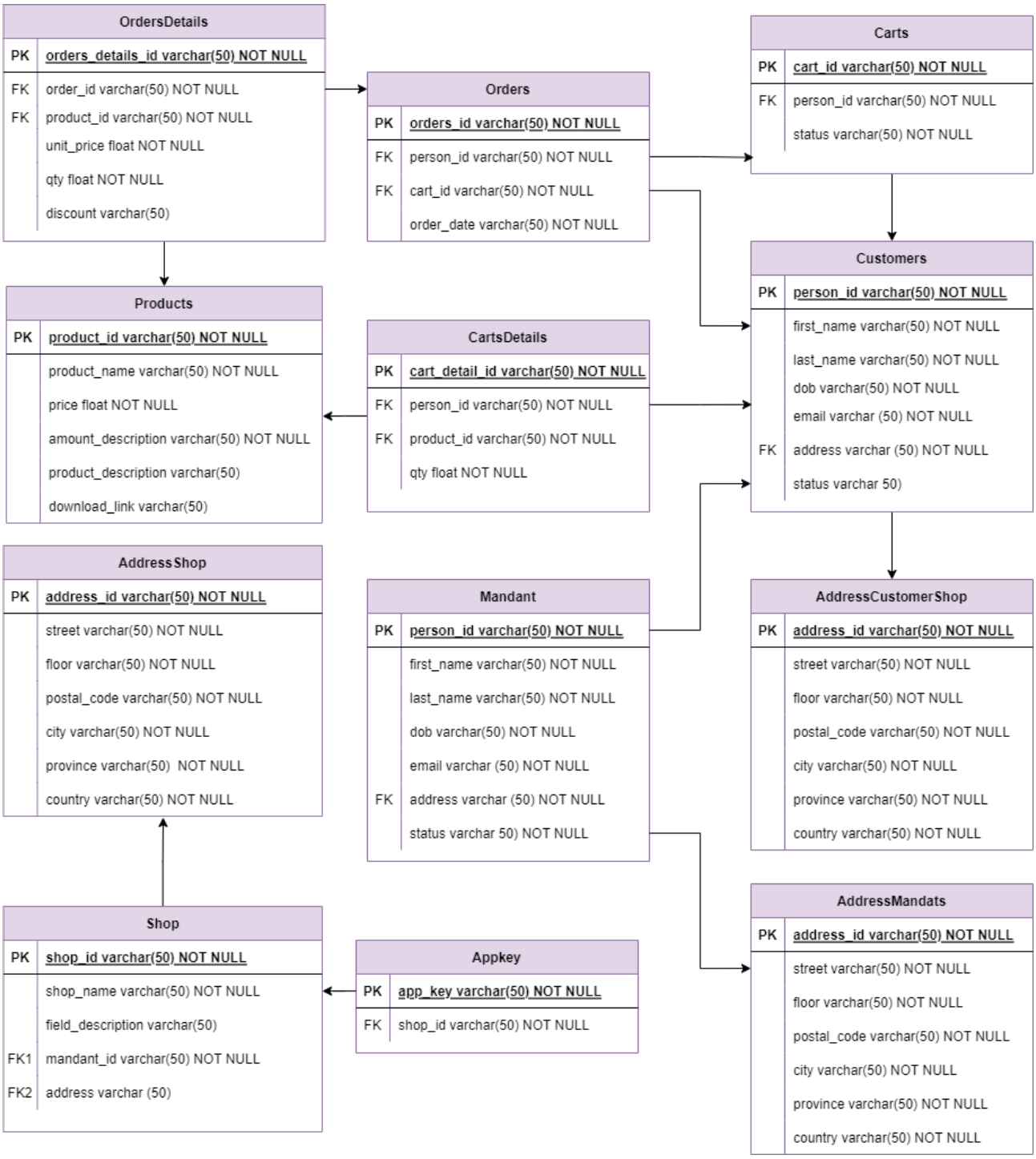
Die zentralen Bestandteile von *CaaS.Core* stellen die Rabattregeln und das Bezahlsystem dar. Überlegen Sie sich, wie Sie die korrekte Funktionsweise dieser Komponenten testen und demonst-rieren können.

(A2.2) *CaaS.Api:* Diese Systemkomponente ist mit ASP.NET als REST-basiertes Web-Service zu realisieren. Die Anforderungen an die zu exportierende Funktionalität des Web-Service ergeben sich aus der Funktionalität von *CaaS.Web*.

(A2.3) *Automatisierte Tests:* Erweitern Sie Ihre Test-Suite um Tests der Geschäftslogik. Die Tests sollen so weit wie möglich die gesamte Funktionalität der Anwendung abdecken.

Isolieren Sie in Ihren Tests die Geschäftslogik von der Persistenzschicht. Ersetzen Sie dazu Ihre DAOs durch Mock-Objekte. Setzen Sie dafür ein Mocking-Framework ein.

(A2.4) *Dokumentation:* Führen Sie die notwendigen Ergänzungen in der Dokumentation durch. Aus Ihrer Dokumentation soll die Architektur des Gesamtsystems hervorgehen, insbesondere die Struktur der Klassen und Interfaces und die Kopplung der Systemkomponenten sollte übersichtlich darge- stellt werden. Hierzu eignen sich besonders gut UML-Klassen- und Paketdiagramme.


OrdersDetails
PK orders_details_id varchar(50) NOT NULL
FK order_id varchar(50) NOT NULL
FK product_id varchar(50) NOT NULL
unit_price float NOT NULL
qty float NOT NULL
discount varchar(50)
Orders
PK orders_id varchar(50) NOT NULL
FK person_id varchar(50) NOT NULL
FK cart_id varchar(50) NOT NULL
order_date varchar(50) NOT NULL
Carts
PK cart_id varchar(50) NOT NULL
FK person_id varchar(50) NOT NULL
status varchar(50) NOT NULL
Customers
PK person_id varchar(50) NOT NULL
first_name varchar(50) NOT NULL
last_name varchar(50) NOT NULL
dob varchar(50) NOT NULL
email varchar (50) NOT NULL
FK address varchar (50) NOT NULL
status varchar 50)
Products
PK product_id varchar(50) NOT NULL
product_name varchar(50) NOT NULL
price float NOT NULL
amount_description varchar(50) NOT NULL
product_description varchar(50)
download_link varchar(50)
CartsDetails
PK cart_detail_id varchar(50) NOT NULL
FK person_id varchar(50) NOT NULL
FK product_id varchar(50) NOT NULL
qty float NOT NULL
AddressShop
PK address_id varchar(50) NOT NULL
street varchar(50) NOT NULL
floor varchar(50) NOT NULL
postal_code varchar(50) NOT NULL
city varchar(50) NOT NULL
province varchar(50) NOT NULL
country varchar(50) NOT NULL
Mandant
PK person_id varchar(50) NOT NULL
first_name varchar(50) NOT NULL
last_name varchar(50) NOT NULL
dob varchar(50) NOT NULL
email varchar (50) NOT NULL
FK address varchar (50) NOT NULL
status varchar 50) NOT NULL
AddressCustomerShop
PK address_id varchar(50) NOT NULL
street varchar(50) NOT NULL
floor varchar(50) NOT NULL
postal_code varchar(50) NOT NULL
city varchar(50) NOT NULL
province varchar(50) NOT NULL
country varchar(50) NOT NULL
Shop
PK shop_id varchar(50) NOT NULL
shop_name varchar(50) NOT NULL
field_description varchar(50)
FK1 mandant_id varchar(50) NOT NULL
FK2 address varchar (50)
Appkey
PK app_key varchar(50) NOT NULL
FK shop_id varchar(50) NOT NULL
AddressMandats
PK address_id varchar(50) NOT NULL
street varchar(50) NOT NULL
floor varchar(50) NOT NULL
postal_code varchar(50) NOT NULL
city varchar(50) NOT NULL
province varchar(50) NOT NULL
country varchar(50) NOT NULL

CaaS.Api

Der Aufbau von meiner Übung lehnt sich stark an die Übung auf. Program.cs beinhalten die Konfiguration fürs StartUp, Datenbanken, Routing, Controller.

Controllers

ProductController.cs
CartsOrderController.cs
StatusInfo.cs
WebApiConventions.cs

Beinhalter die Kontrollstruktur zum Aufruf von Transferfunktionen. Die Transferfunktionen sind für den Transport von Daten DTO zuständig.

Profiles

CustomerProfile.cs
OrderProfile.cs
CartDetailsProfile.cs
OrderDetailsprofile.cs
Cartprofile.cs
Productprofile.cs

Profiles stellt das Mapping zwichen Domänenklassen und DTO Object

CaaS.Dtos

AddressDTO.cs
AppKeyDTO.cs
CartDetailsDTO.cs
CartDTO.cs
OrderDetailsDTO.cs
OrderDTO.cs
PersonDTO.cs
ProductDTO.cs
ShopDTO.cs

DTO spiegelt die Domänenklassen wieder und dient als Transportobjekt zwecks Isolierung

CaaS.Logic

IManagementLogic.cs
IOrderManagementLogic .cs
ManagementLogic.cs

OrderManagementLogic.cs

Beherbergt die Interfaces und Implementierungen der Logik, die für Controller zuständig sind.

CaaS.Features

Das Rabattsystem wurde mit abstract record implementiert und über mit generischem Delegate Func aufgerufen.
Die Statistik in der StatsController implementiert und die Logik dahinter im Order CaaS.Feature.
Ebenfalls die Shopverwaltung erfolgte über die Verwaltungsdatei in CaaS.Features.

Ordnername/Projekname CaaS.Domain

Address.cs

```
namespace CaaS.Domain;
public class Address
{
    public Address(string id, string street, string floor, double postalCode, string city,
string province,string country)
    {
        Id = id;
        Street = street;
        Floor = floor;
        PostalCode = postalCode;
        City = city;
        Province = province;
        Country = country;
    }

    public string Id { get; set; }
    public string Street { get; private set; }
    public string Floor { get; private set; }
    public double PostalCode { get; private set; }
    public string City { get; private set; }
    public string Province { get; private set; }
    public string Country { get; private set; }
    public override string ToString() =>
      $"Address(id:{Id}, Street:{Street}, Floor{Floor}, PostalCode:{PostalCode},
City:{City})";
}
```

AppKey.cs

```
namespace CaaS.Domain;
public class AppKey
{
    public AppKey(string id, string shopId)
    {
        Id = id;
        ShopId = shopId;

    }
    public string Id { get; set; }
    public string ShopId { get; private set; }

    public override string ToString() =>
      $"AppKey(id:{Id}, CustId:{ShopId})";
}
```

Cart.cs

```
namespace CaaS.Domain;
public class Cart
{
    public Cart(string id, string custId, string status)
    {
        Id = id;
        CustId = custId;
        Status = status;
    }
    public string Id { get; set; }
    public string CustId { get; private set; }
    public string Status { get; private set; }
    public override string ToString() =>
      $"Cart(id:{Id}, CustId:{CustId}, Status:{Status})";
}
```

CartDetails.cs

```
namespace CaaS.Domain;
public class CartDetails
{
    public CartDetails(string id, string cartId, string productId, double quantity)
    {
        Id = id;
        CartId = cartId;
        ProductId = productId;
        Quantity = quantity;
    }
    public string Id { get; set; }
    public string CartId { get; private set; }
    public string ProductId { get; private set; }
    public double Quantity { get; private set; }
    public override string ToString() =>
      $"Cart(id:{Id}, CartId:{CartId}, Status:{ProductId})";
}
```

Order.cs

```
namespace CaaS.Domain;
public class Order
{
    public Order(string id, string custId, string cartId, DateTime orderDate)
    {
        Id = id;
        CustId = custId;
        CartId = cartId;
        OrderDate = orderDate
    }
    public string Id { get; set; }
    public string CustId { get; private set; }
    public string CartId { get; private set; }
```

```
    public DateTime OrderDate { get; private set; }
    public override string ToString() =>
      $"Order(id:{Id}, CustId:{CustId}, CartId{CartId}, OrderDate:{OrderDate:yyyy-MM-dd})";
}
```

OrderDetails.cs

```
namespace CaaS.Domain;
public class OrderDetails

{
    public OrderDetails(string id, string orderId, string productId, double unitPrice,
double quantity, double discount)
    {
        Id = id;
        OrderId = orderId;
        ProductId = productId;
        UnitPrice = unitPrice;
        Quantity = quantity;
        Discount = discount;
    }
    public string Id { get; set; }
    public string OrderId { get; private set; }
    public string ProductId { get; private set; }
    public double UnitPrice { get; private set; }
    public double Quantity { get; private set; }
    public double Discount { get; private set; }
    public override string ToString() =>
      $"OrdersDetails(id:{Id}, OrderId:{OrderId}, ProductId{ProductId},
UnitPrice:{UnitPrice}, Quantity:{Quantity})";
}
```

Person.cs

```
namespace CaaS.Domain;
public class Person
{
    public Person(string id, string firstName, string lastName, DateTime dateOfBirth
,string email, string addressId, string status)
    {
        Id = id;
        FirstName = firstName;
        LastName = lastName;
        DateOfBirth = dateOfBirth;
        Email = email;
        AddressId = addressId;
        Status = status;
    }
    public string Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public DateTime DateOfBirth { get; set; }
```

```
    public string Email { get; }
    public string AddressId { get; }
    public string Status { get; }
    public override string ToString() =>
      $"Person(id:{Id}, FirstName:{FirstName}, LastName:{LastName},
DateOfBirth:{DateOfBirth:yyyy-MM-dd})";
}
```

Product.cs

```
namespace CaaS.Domain;
public class Product
{
    public Product(string id, string name, double price, string amountDesc, string
productDesc, string downloadLink)
    {
        if (string.IsNullOrEmpty(id))
        {
            throw new ArgumentException($"'{nameof(id)}' cannot be null or empty.",
nameof(id));
        }
        if (string.IsNullOrEmpty(name))
        {
            throw new ArgumentException($"'{nameof(name)}' cannot be null or empty.",
nameof(name));
        }
        if (string.IsNullOrEmpty(amountDesc))
        {
            throw new ArgumentException($"'{nameof(amountDesc)}' cannot be null or empty.",
nameof(amountDesc));
        }
        Id = id;
        this.Name = name;
        this.Price = price;
        this.AmountDesc = amountDesc;
        this.ProductDesc = productDesc;
        this.DownloadLink = downloadLink;
    }
    public string Id { get; set; }
    public string Name { get; set; }
    public double Price{ get; set; }
    public string AmountDesc { get; set; }
    public string? ProductDesc { get; set; }
    public string? DownloadLink { get; set; }
    public override string ToString() =>
      $"Product(id:{Id}, name:{Name}, price:{Price}, amountDesc:{AmountDesc},
productDesc:{ProductDesc})";
}
```

Shop.cs

```
namespace CaaS.Domain;
public class Shop
{
    public Shop(string id, string name,string fieldDesc, string mandantId, string address)
    {
        Id = id;
        this.Name = name;
        this.FieldDesc = fieldDesc;
        this.MandantId = mandantId;
        this.Address = address;
    }
    public string Id { get; set; }
    public string Name { get; set; }
    public string FieldDesc{ get; set; }
    public string MandantId { get; set; }
    public string Address { get;  set; }
    public override string ToString() =>
      $"Shop(id:{Id}, name:{Name}, mandant:{MandantId})";
}
```

Ordnername /Projekname : CaaS.Dal.Interfaces

IAddressDao.cs

```
using CaaS.Domain;
namespace CaaS.Dal.Interface
{
    public interface IAddressDao
    {
        Task<IEnumerable<Address>> FindAllAsync(string table);
        Task<Address?> FindByIdAsync(string id, string table);
        Task<bool> UpdateAsync(Address address, string table);
        Task<bool> DeleteByIdAsync(string id, string table);
        Task<bool> StoreAsync(Address address, string table);
    }
}
```

IAppKeyDao.cs

```
using CaaS.Domain;
namespace CaaS.Dal.Interface
{
    public interface IAppKeyDao
    {
        Task<IEnumerable<AppKey>> FindAllAsync(string table);
        Task<AppKey?> FindByIdAsync(string id, string table);

        Task<bool> UpdateAsync(AppKey appkey, string table);
        Task<bool> DeleteByIdAsync(string id, string table);
        Task<bool> StoreAsync(AppKey appkey, string table);
    }
}
```

ICartDao.cs

```
using CaaS.Domain;
namespace CaaS.Dal.Interface
{
    public interface ICartDao
    {
        Task<IEnumerable<Cart>> FindAllAsync(string table);
        Task<Cart?> FindByIdAsync(string id, string table);

        Task<bool> UpdateAsync(Cart cart, string table);
        Task<bool> DeleteByIdAsync(string id, string table);
        Task<bool> StoreAsync(Cart cart, string table);
    }
}
```

ICartDetailsDao.cs

```
using CaaS.Domain;
namespace CaaS.Dal.Interface
{
    public interface IOrderDetailsDao
    {
        Task<IEnumerable<OrderDetails>> FindAllAsync(string table);
        Task<OrderDetails?> FindByIdAsync(string id, string table);
        Task<bool> UpdateAsync(OrderDetails Orderdetails, string table);
        Task<bool> DeleteByIdAsync(string id, string table);
        Task<bool> StoreAsync(OrderDetails Orderdetails, string table);
    }
}
```

IOrderDetailsDao.cs

```
using CaaS.Domain;
namespace CaaS.Dal.Interface
{
    public interface ICartDetailsDao
    {
        Task<IEnumerable<CartDetails>> FindAllAsync(string table);
        Task<CartDetails?> FindByIdAsync(string id, string table);
        Task<bool> UpdateAsync(CartDetails cartdetails, string table);
        Task<bool> DeleteByIdAsync(string id, string table);
        Task<bool> StoreAsync(CartDetails cartdetails, string table);
    }
}
```

IPersonDao.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using CaaS.Domain;
nameSpace CaaS.Dal.Interface;
public interface IPersonDao
{
    Task<IEnumerable<Person>> FindAllAsync(string table);
    Task<Person?> FindByIdAsync(string id,string table);

    Task<bool> UpdateAsync(Person person, string table);
    Task<bool> DeleteByIdAsync(string id, string table);
    Task<bool> StoreAsync(Person person, string table);
}
```

IProductDao.cs

```
using CaaS.Domain;
namespace CaaS.Dal.Interface;
public interface IProductDao
{
    Task<IEnumerable<Product>> FindAllAsync(string table);
    Task<Product?> FindByIdAsync(string id, string table);
    Task<bool> UpdateAsync(Product product, string table);
    Task<bool> DeleteByIdAsync(string id, string table);
    Task<bool> StoreAsync(Product product, string table);
}
```

IShopDao.cs

```
using CaaS.Domain;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace CaaS.Dal.Interface;
public interface IShopDao
{
    Task<IEnumerable<Shop>> FindAllAsync(string table);
    Task<Shop?> FindByIdAsync(string id, string table);

    Task<bool> UpdateAsync(Shop shop, string table);
    Task<bool> DeleteByIdAsync(string id, string table);
    Task<bool> StoreAsync(Shop shop, string table);
}
```

Ordnername /Projekname : CaaS.Dal.Ado

AdoAddressDao.cs

```
  using Dal.Common;
//using Microsoft.Data.SqlClient;
using CaaS.Domain;
using CaaS.Dal.Interface;
using System.Data;
namespace CaaS.Dal.Ado;
public class AdoAddressDao : IAddressDao
{
    private readonly AdoTemplate template;
    private Address MapRowToAddress(IDataRecord row) =>
        new(
            id: (string)row["address_id"],
            street: (string)row["street"],
            floor: (string)row["floor"],
            postalCode: (double)row["postal_code"],
            city: (string)row["city"],
            province: (string)row["province"],
            country: (string)row["country"]
        );
    private Address? FindByIdSync(string id, string table)
    {
        return template.QuerySingleSync($"select * from {table} where address_id=@id",
            MapRowToAddress,
            new QueryParameter("@id", id));
    }
    private bool IsAddressByIdSync(string id, string table)
    {
        return FindByIdSync(id, table) is not null;
    }
    public AdoAddressDao(IConnectionFactory connectionFactory)
    {
        this.template = new AdoTemplate(connectionFactory);
    }
    public async Task<IEnumerable<Address>> FindAllAsync(string table)
    {
        return await template.QueryAsync($"select * from {table}", MapRowToAddress);
    }
    public async Task<Address?> FindByIdAsync(string id, string table)
    {
        return await template.QuerySingleAsync($"select * from {table} where
address_id=@id",
            MapRowToAddress,
```

```
            new QueryParameter("@id", id));
    }

    public async Task<bool> UpdateAsync(Address address, string table)
    {
        if (!IsAddressByIdSync(address.Id, table))
        {
            string sqlcmd = $"update {table} set street=@street, floor=@floor,
postal_code=@postal_code where address_id=@id";
            // array für die Parameter erstellen
            return await template.ExecuteAsync(@sqlcmd,
                new QueryParameter("@id", address.Id),
                new QueryParameter("@street", address.Street),
                new QueryParameter("@floor", address.Floor),
                new QueryParameter("@postal_code", address.PostalCode),
                new QueryParameter("@city", address.City),
                new QueryParameter("@province", address.Province),
                new QueryParameter("@country", address.Country)) == 1;
        }
        return false;
    }
    public async Task<bool> DeleteByIdAsync(string id, string table)
    {
        if (!IsAddressByIdSync(id, table))
        {
            string sqlcmd = $"delete from {table} where address_id=@id";
            return await template.ExecuteAsync(@sqlcmd,
                   new QueryParameter("@id", id)) == 1;
        }
        return false;
    }

    public async Task<bool> StoreAsync(Address address, string table)
    {
        if (!IsAddressByIdSync(address.Id, table))
        {
            string sqlcmd = $"insert into {table} ( address_id,street, floor, postal_code,
city, province, country ) " +
            "values (@id,@street, @floor,@postal_code,@city,@province,@country) ";
            return await template.ExecuteAsync(@sqlcmd,
                new QueryParameter("@id", address.Id),
                new QueryParameter("@street", address.Street),
                new QueryParameter("@floor", address.Floor),
                new QueryParameter("@postal_code", address.PostalCode),
                new QueryParameter("@city", address.City),
                new QueryParameter("@province", address.Province),
                new QueryParameter("@country",address.Country)) == 1;
        }
        return false;
```

```
    }
}
```

AdoAppKeyDap.cs

```
 using Dal.Common;
//using Microsoft.Data.SqlClient;
using CaaS.Dal.Interface;
using CaaS.Domain;
using System.Data;
namespace CaaS.Dal.Ado;
public class AdoAppKeyDao : IAppKeyDao
{
    private readonly AdoTemplate template;
    private AppKey MapRowToAppKey(IDataRecord row) =>
        new(
            id: (string)row["app_key"],
            shopId: (string)row["shop_id"]
        );
    private AppKey? FindByIdSync(string id, string table)
    {
        return template.QuerySingleSync($"select * from {table} where app_key=@id",
            MapRowToAppKey,
            new QueryParameter("@id", id));
    }
    private bool IsAppKeyByIdSync(string id, string table)
    {
        return FindByIdSync(id, table) is not null;
    }
    public AdoAppKeyDao(IConnectionFactory connectionFactory)
    {
        this.template = new AdoTemplate(connectionFactory);
    }
    public async Task<IEnumerable<AppKey>> FindAllAsync(string table)
    {
        return await template.QueryAsync($"select * from {table}", MapRowToAppKey);
    }

    public async Task<AppKey?> FindByIdAsync(string id, string table)
    {
        return await template.QuerySingleAsync($"select * from {table} where app_key=@id",
            MapRowToAppKey,
            new QueryParameter("@id", id));
    }
    public async Task<bool> UpdateAsync(AppKey appKey, string table)
    {
        if (!IsAppKeyByIdSync(appKey.Id, table))
        {
            string sqlcmd = $"update {table} set shop_id=@shopId where app_key=@id";
```

```
            // array für die Parameter erstellen
            return await template.ExecuteAsync(@sqlcmd,
                new QueryParameter("@id", appKey.Id),
                new QueryParameter("@shopId", appKey.ShopId)
                ) == 1;
        }
        return false;
    }
    public async Task<bool> DeleteByIdAsync(string id, string table)
    {
        if (!IsAppKeyByIdSync(id, table))
        {
            string sqlcmd = $"delete from {table} where app_key=@id";
            return await template.ExecuteAsync(@sqlcmd,
                   new QueryParameter("@id", id)) == 1;
        }
        return false;
    }

    public async Task<bool> StoreAsync(AppKey appKey, string table)
    {
        if (!IsAppKeyByIdSync(appKey.Id, table))
        {
            string sqlcmd = $"insert into {table} ( app_key,shop_id ) " +
            "values (@id,@shopId) ";

            return await template.ExecuteAsync(@sqlcmd,
                new QueryParameter("@id", appKey.Id),
                new QueryParameter("@shopId", appKey.ShopId)
                ) == 1;
        }
        return false;
    }
}
```

AdoCartDao.cs

```
  using Dal.Common;
//using Microsoft.Data.SqlClient;
using CaaS.Dal.Interface;
using CaaS.Domain;
using System.Data;
namespace CaaS.Dal.Ado;
public class AdoCartDao : ICartDao
{
    private readonly AdoTemplate template;
    private Cart MapRowToCart(IDataRecord row) =>
        new (
            id: (string)row["cart_id"],
            custId: (string)row["cust_id"],
            status: (string)row["status"]
        );
    private Cart? FindByIdSync(string id, string table)
    {
        return template.QuerySingleSync($"select * from {table} where cart_id=@id",
            MapRowToCart,
            new QueryParameter("@id", id));
    }
    private bool IsCartByIdSync(string id, string table)
    {
        return FindByIdSync(id, table) is not null;
    }
    public AdoCartDao(IConnectionFactory connectionFactory)
    {
        this.template = new AdoTemplate(connectionFactory);
    }

    public async Task<IEnumerable<Cart>> FindAllAsync(string table)
    {
        return await template.QueryAsync($"select * from {table}", MapRowToCart);
    }

    public async Task<Cart?> FindByIdAsync(string id,string table)
    {
        return await template.QuerySingleAsync($"select * from {table} where cart_id=@id",
            MapRowToCart,
            new QueryParameter("@id", id));
    }
    public async Task<bool> UpdateAsync(Cart cart,string table)
    {
        if (!IsCartByIdSync(cart.Id, table))
        {
```

```
            string sqlcmd = $"update {table} set cust_id=@custId, status=@status where
cart_id=@id";
            // array für die Parameter erstellen
            return await template.ExecuteAsync(@sqlcmd,
                new QueryParameter("@id", cart.Id),
                new QueryParameter("@custId", cart.CustId),
                new QueryParameter("@status", cart.Status)
                ) == 1;
        }
        return false;
    }
    public async Task<bool> DeleteByIdAsync(string id, string table)
    {
        if (!IsCartByIdSync(id, table))
        {
            string sqlcmd = $"delete from {table} where cart_id=@id";
            return await template.ExecuteAsync(@sqlcmd,
                   new QueryParameter("@id", id)) == 1;
        }
        return false;
    }
    public async Task<bool> StoreAsync(Cart cart, string table)
    {
        if (!IsCartByIdSync(cart.Id, table))
        {
            string sqlcmd = $"insert into {table} (cart_id,cust_id, status) " +
           "values (@id,@custId, @status) ";

            return await template.ExecuteAsync(@sqlcmd,
                new QueryParameter("@id", cart.Id),
                new QueryParameter("@custId", cart.CustId),
                new QueryParameter("@status", cart.Status)
                ) == 1;
        }
        return false;
    }
}
```

AdoCartsDetailsDao.cs

```
  using Dal.Common;
//using Microsoft.Data.SqlClient;
using CaaS.Dal.Interface;
using CaaS.Domain;
using System.Data;
namespace CaaS.Dal.Ado;
public class AdoCartDetailsDao : ICartDetailsDao
{
    private readonly AdoTemplate template;

    private CartDetails MapRowToCartDetails(IDataRecord row) =>
        new (
            id: (string)row["cart_details_id"],
            cartId: (string)row["cart_id"],
            productId: (string)row["product_id"],
            quantity: (double)row["qty"]
        );
    private CartDetails? FindByIdSync(string id, string table)
    {
        return template.QuerySingleSync($"select * from {table} where cart_details_id=@id",
            MapRowToCartDetails,
            new QueryParameter("@id", id));
    }
    private bool IsCartDetailsByIdSync(string id, string table)
    {
        return FindByIdSync(id, table) is not null;
    }
    public AdoCartDetailsDao(IConnectionFactory connectionFactory)
    {
        this.template = new AdoTemplate(connectionFactory);
    }
    public async Task<IEnumerable<CartDetails>> FindAllAsync(string table)
    {
        return await template.QueryAsync($"select * from {table}", MapRowToCartDetails);
    }
    public async Task<CartDetails?> FindByIdAsync(string id,string table)
    {
        return await template.QuerySingleAsync($"select * from {table} where
cart_details_id=@id",
            MapRowToCartDetails,
            new QueryParameter("@id", id));
    }
    public async Task<bool> UpdateAsync(CartDetails cartDetails,string table)
    {
        if (!IsCartDetailsByIdSync(cartDetails.Id, table))
        {
```

```
            string sqlcmd = $"update {table} set cart_id=@cartId, product_id=@productId,
qty=@quantity where cart_details_id=@id";
            // array für die Parameter erstellen
            return await template.ExecuteAsync(@sqlcmd,
                new QueryParameter("@id", cartDetails.Id),
                new QueryParameter("@cartId", cartDetails.CartId),
                new QueryParameter("@productId", cartDetails.ProductId),
                new QueryParameter("@quantity", cartDetails.Quantity)) == 1;
        }
        return false;
    }
    public async Task<bool> DeleteByIdAsync(string id, string table)
    {
        if (!IsCartDetailsByIdSync(id, table))
        {
            string sqlcmd = $"delete from {table} where cart_details_id=@id";
            return await template.ExecuteAsync(@sqlcmd,
                   new QueryParameter("@id", id)) == 1;
        }
        return false;
    }
    public async Task<bool>StoreAsync(CartDetails cartDetails, string table)
    {
        if (!IsCartDetailsByIdSync(cartDetails.Id, table))
        {
            string sqlcmd = $"insert into {table} ( cart_details_id,cart_id, product_id,
qty ) " +
            "values (@id,@cartId, @productId,@quantity)";

            return await template.ExecuteAsync(@sqlcmd,
                new QueryParameter("@id", cartDetails.Id),
                new QueryParameter("@cartId", cartDetails.CartId),
                new QueryParameter("@productId", cartDetails.ProductId),
                new QueryParameter("@quantity", cartDetails.Quantity)
                ) == 1;
        }
        return false;
    }
}
```

AdoOrderDao.cs

```
  using Dal.Common;
//using Microsoft.Data.SqlClient;
using CaaS.Dal.Interface;
using CaaS.Domain;
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Common;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data.SqlTypes;
namespace CaaS.Dal.Ado;
public class AdoOrderDao : IOrderDao
{
    private readonly AdoTemplate template;
    private Order MapRowToOrder(IDataRecord row) =>
        new (
            id: (string)row["order_id"],
            custId: (string)row["cust_id"],
            cartId: (string)row["cart_id"],
            orderDate: (DateTime)row["order_date"]
        );
    private Order? FindByIdSync(string id, string table)
    {
        return template.QuerySingleSync($"select * from {table} where order_id=@id",
            MapRowToOrder,
            new QueryParameter("@id", id));
    }
    private bool IsOrderByIdSync(string id, string table)
    {
        return FindByIdSync(id, table) is not null;
    }
    public AdoOrderDao(IConnectionFactory connectionFactory)
    {
        this.template = new AdoTemplate(connectionFactory);
    }
    public async Task<IEnumerable<Order>> FindAllAsync(string table)
    {
        return await template.QueryAsync($"select * from {table}", MapRowToOrder);
    }
    public async Task<Order?> FindByIdAsync(string id,string table)
    {
        return await template.QuerySingleAsync($"select * from {table} where order_id=@id",
            MapRowToOrder,
            new QueryParameter("@id", id));
    }
```

```
    public async Task<bool> UpdateAsync(Order Order,string table)
    {
        if (!IsOrderByIdSync(Order.Id, table))
        {
            string sqlcmd = $"update {table} set cust_id=@custId, cart_id=@cartId,
order_date=@order_date where order_id=@id";
            // array für die Parameter erstellen
            return await template.ExecuteAsync(@sqlcmd,
                new QueryParameter("@id", Order.Id),
                new QueryParameter("@custId", Order.CustId),
                new QueryParameter("@cartId", Order.CartId),
                new QueryParameter("@order_date", Order.OrderDate)) == 1;
        }
        return false;
    }
    public async Task<bool> DeleteByIdAsync(string id, string table)
    {
        if (!IsOrderByIdSync(id, table))
        {
            string sqlcmd = $"delete from {table} where order_id=@id";
            return await template.ExecuteAsync(@sqlcmd,
                   new QueryParameter("@id", id)) == 1;
        }
        return false;
    }
    public async Task<bool>StoreAsync(Order Order, string table)
    {
        if (!IsOrderByIdSync(Order.Id, table))
        {
            string sqlcmd = $"insert into {table} ( order_id,cust_id, cart_id, order_date)
" +
            "values (@id,@custId, @cartId,@order_date) ";

            return await template.ExecuteAsync(@sqlcmd,
                new QueryParameter("@id", Order.Id),
                new QueryParameter("@custId", Order.CustId),
                new QueryParameter("@cartId", Order.CartId),
                new QueryParameter("@order_date", Order.OrderDate)
                ) == 1;
        }
        return false;
    }
}
```

AdoProductDao.cs

```
  using Dal.Common;
//using Microsoft.Data.SqlClient;
using CaaS.Dal.Interface;
using CaaS.Domain;
using System.Data;
namespace CaaS.Dal.Ado;
public class AdoProductDao : IProductDao
{
    private readonly AdoTemplate template;
    private Product MapRowToProduct(IDataRecord row) =>
        new (
            id: (string)row["product_id"],
            name: (string)row["product_name"],
            price: (double) row ["price"],
            amountDesc: (string)row["amount_desc"],
            productDesc: (string)row["product_desc"],
            downloadLink: (string)row["download_link"]
        );
    private Product? FindByIdSync(string id, string table)
    {
        return template.QuerySingleSync($"select * from {table} where product_id=@id",
            MapRowToProduct,
            new QueryParameter("@id", id));
    }
    private bool IsProductByIdSync(string id, string table)
    {
        return FindByIdSync(id, table) is not null;
    }
    public AdoProductDao(IConnectionFactory connectionFactory)
    {
        this.template = new AdoTemplate(connectionFactory);
    }
    public async Task<IEnumerable<Product>> FindAllAsync(string table)
    {
        return await template.QueryAsync($"select * from {table}", MapRowToProduct);
    }
    public async Task<Product?> FindByIdAsync(string id,string table)
    {
        return await template.QuerySingleAsync($"select * from {table} where
product_id=@id",
            MapRowToProduct,
            new QueryParameter("@id", id));
    }
    public async Task<bool> UpdateAsync(Product product,string table)
    {
        if (!IsProductByIdSync(product.Id, table))
```

```
        {
            string sqlcmd = $"update {table} set product_name=@name, price = @price," +
            $"amount_desc=@amountDesc,
product_desc=@productDesc,download_link=@downloadLink  where product_id=@id";
            // array für die Parameter erstellen
            return await template.ExecuteAsync(@sqlcmd,
                new QueryParameter("@id", product.Id),
                new QueryParameter("@name", product.Name),
                new QueryParameter("@price", product.Price),
                new QueryParameter("@amountDesc", product.AmountDesc),
                new QueryParameter("@productDesc", product.ProductDesc),
                new QueryParameter("@downloadLink", product.DownloadLink)
                ) == 1;
        }
        return false;
    }
    public async Task<bool> DeleteByIdAsync(string id, string table)
    {
        if (!IsProductByIdSync(id, table))
        {
            string sqlcmd = $"delete from {table} where product_id=@id";
            return await template.ExecuteAsync(@sqlcmd,
                   new QueryParameter("@id", id)) == 1;
        }
        return false;
    }
    public async Task<bool> StoreAsync(Product product, string table)
    {
        if (!IsProductByIdSync(product.Id, table))
        {
            string sqlcmd = $"insert into {table} (product_id,product_name, price,
amount_desc, product_desc, download_link ) " +
           "values (@id,@name, @price,@amountDesc,@productDesc,@downloadLink) ";
            return await template.ExecuteAsync(@sqlcmd,
                new QueryParameter("@id", product.Id),
                new QueryParameter("@name", product.Name),
                new QueryParameter("@price", product.Price),
                new QueryParameter("@amountDesc", product.AmountDesc),
                new QueryParameter("@productDesc", product.ProductDesc),
                new QueryParameter("@downloadLink", product.DownloadLink)
                ) == 1;
        }
        return false;
    }
}
```

AdoShopDao.cs

```
  using Dal.Common;
//using Microsoft.Data.SqlClient;
using CaaS.Dal.Interface;
using CaaS.Domain;
using System.Data;
namespace CaaS.Dal.Ado;
public class AdoShopDao : IShopDao
{
    private readonly AdoTemplate template;
    private readonly IShopDao shopDao;
    private object cw;
    private Shop MapRowToShop(IDataRecord row) =>
        new (
            id: (string)row["shop_id"],
            name: (string)row["shop_name"],
            fieldDesc: (string)row["field_descriptions"],
            mandantId: (string)row["mandant_id"],
            address: (string)row["address"]
        );
    private Shop? FindByIdSync(string id, string table)
    {
        return template.QuerySingleSync($"select * from {table} where shop_id=@id",
            MapRowToShop,
            new QueryParameter("@id", id));
    }
    private bool IsShopByIdSync(string id, string table)
    {
        return FindByIdSync(id, table) is not null;
    }
    public AdoShopDao(IConnectionFactory connectionFactory)
    {
        this.template = new AdoTemplate(connectionFactory);
    }
    public async Task<IEnumerable<Shop>> FindAllAsync(string table)
    {
        return await template.QueryAsync($"select * from {table}", MapRowToShop);
    }
    public async Task<Shop?> FindByIdAsync(string id,string table)
    {
        return await template.QuerySingleAsync($"select * from {table} where shop_id=@id",
            MapRowToShop,
            new QueryParameter("@id", id));
    }
    public async Task<bool> UpdateAsync(Shop shop,string table)
    {
        if (!IsShopByIdSync(shop.Id, table))
        {
```

```
            string sqlcmd = $"update {table} set shop_name=@name,
field_descriptions=@fieldDesc, mandant_id=@mandantId, address=@address where shop_id=@id";
            // array für die Parameter erstellen
            return await template.ExecuteAsync(@sqlcmd,
                new QueryParameter("@id", shop.Id),
                new QueryParameter("@name", shop.Name),
                new QueryParameter("@fieldDesc", shop.FieldDesc),
                new QueryParameter("@mandantId", shop.MandantId),
                new QueryParameter("@address", shop.Address)
                ) == 1;
        }
        return false;
    }
    public async Task<bool> DeleteByIdAsync(string id, string table)
    {
        if (!IsShopByIdSync(id, table))
        {
            string sqlcmd = $"delete from {table} where shop_id=@id";
            return await template.ExecuteAsync(@sqlcmd,
                   new QueryParameter("@id", id)) == 1;
        }
        return false;
    }
    public async Task<bool> StoreAsync(Shop shop, string table)
    {
        if(!IsShopByIdSync(shop.Id,table))
        {
        string sqlcmd = $"insert into {table} ( shop_id,shop_name, field_descriptions,
mandant_id, address) " +
            "values (@id,@name, @fieldDesc,@mandantId,@address) ";

        return await template.ExecuteAsync(@sqlcmd,
            new QueryParameter("@id", shop.Id),
            new QueryParameter("@name", shop.Name),
            new QueryParameter("@fieldDesc", shop.FieldDesc),
            new QueryParameter("@mandantId", shop.MandantId),
            new QueryParameter("@address", shop.Address)
            ) == 1;
        }
        return false;
    }
}
```

Ordnername /Projekname Dal.Common

Ado.Template.cs

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Common;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Dal.Common;
// Delegates in eigenes File geben
public delegate T RowMapper<T>(IDataRecord row);
public class AdoTemplate
{
    private readonly IConnectionFactory connectionFactory;

    public AdoTemplate(IConnectionFactory connectionFactory)
    {
        this.connectionFactory = connectionFactory;
    }
    private void AddParameters(DbCommand command, QueryParameter[] parameters)
    {
        foreach (var p in parameters)
        {
            DbParameter dbParam = command.CreateParameter();
            dbParam.ParameterName = p.Name;
            dbParam.Value = p.Value;
            command.Parameters.Add(dbParam);
        }
    }
    public async Task<IEnumerable<T>> QueryAsync<T>(string sql, RowMapper<T> rowMapper,
        params QueryParameter[] parameters)
    {
        await using DbConnection connection = await
connectionFactory.CreateConnectionAsync();
        await using DbCommand command = connection.CreateCommand();
        command.CommandText = sql;
        AddParameters(command, parameters);

        var items = new List<T>();
        await using (DbDataReader reader = await command.ExecuteReaderAsync())
        {
            while (await reader.ReadAsync())
            {
```

```
                items.Add(rowMapper(reader));
            }
        }
        return items;
    }
    public IEnumerable<T> QuerySync<T>(string sql, RowMapper<T> rowMapper,
        params QueryParameter[] parameters)
    {
        using DbConnection connection = connectionFactory.CreateConnectionSync();
        using DbCommand command = connection.CreateCommand();
        command.CommandText = sql;
        AddParameters(command, parameters);
        var items = new List<T>();
        using (DbDataReader reader = command.ExecuteReader())
        {
            while (reader.Read())
            {
                items.Add(rowMapper(reader));
            }
        }
        return items;
    }
    public async Task<T?> QuerySingleAsync<T>(string sql, RowMapper<T> rowMapper,
        params QueryParameter[] parameters)
    {
        return (await QueryAsync(sql, rowMapper, parameters)).SingleOrDefault();
    }


    public T? QuerySingleSync<T>(string sql, RowMapper<T> rowMapper,
        params QueryParameter[] parameters)
    {
        return QuerySync(sql, rowMapper, parameters).SingleOrDefault();
    }

    public async Task<int> ExecuteAsync(string sql, params QueryParameter[] parameters)
    {
        await using DbConnection connection = await
connectionFactory.CreateConnectionAsync();
        await using DbCommand command = connection.CreateCommand();
        command.CommandText = sql;
        AddParameters(command, parameters);

        return await command.ExecuteNonQueryAsync();
    }
}
```

ConfigurationUtil.cs

```
namespace Dal.Common;

using Microsoft.Extensions.Configuration;

public static class ConfigurationUtil
{
  private static IConfiguration? configuration = null;

  public static IConfiguration GetConfiguration() =>
    configuration ??= new ConfigurationBuilder()
      .AddJsonFile("appsettings.json", optional: false)
      .Build();

  public static (string connectionString, string providerName)
GetConnectionParameters(string configName)
  {
    var connectionConfig =
GetConfiguration().GetSection("ConnectionStrings").GetSection(configName);
    return (connectionConfig["ConnectionString"], connectionConfig["ProviderName"]);
  }
}
```

DbUtil.cs

```
namespace Dal.Common;
using System.Data.Common;
public static class DbUtil
{
  public static void RegisterAdoProviders()
  {
    // Use new Implementation of MS SQL Provider
    DbProviderFactories.RegisterFactory("Microsoft.Data.SqlClient",
Microsoft.Data.SqlClient.SqlClientFactory.Instance);
    DbProviderFactories.RegisterFactory("MySql.Data.MySqlClient",
MySql.Data.MySqlClient.MySqlClientFactory.Instance);
  }
}
```

DefaultConnectionFactory.cs

```
using System.Data.SqlClient;
namespace Dal.Common;
using System.Data.Common;
using System.Threading.Tasks;
using Dal.Common;
using Microsoft.Extensions.Configuration;
public class DefaultConnectionFactory : IConnectionFactory
{
    private readonly DbProviderFactory dbProviderFactory;
    public static IConnectionFactory FromConfiguration(IConfiguration config, string
connectionStringConfigName)
    {
        var connectionConfig =
config.GetSection("ConnectionStrings").GetSection(connectionStringConfigName);
        string connectionString = connectionConfig["ConnectionString"];
        string providerName = connectionConfig["ProviderName"];
        return new DefaultConnectionFactory(connectionString, providerName);
    }
    public DefaultConnectionFactory(string connectionString, string providerName)
    {
        this.ConnectionString = connectionString;
        this.ProviderName = providerName;
        DbUtil.RegisterAdoProviders();
        this.dbProviderFactory = DbProviderFactories.GetFactory(providerName);
    }
    public string ConnectionString { get; }
    public string ProviderName { get; }
    public async Task<DbConnection> CreateConnectionAsync()
    {
        var connection = dbProviderFactory.CreateConnection();
        if (connection is null)
        {
            throw new InvalidOperationException("DbProviderFactory.CreateConnection()
returned null");
        }
        connection.ConnectionString = this.ConnectionString;
        await connection.OpenAsync();
        return connection;
    }
    public DbConnection? CreateConnectionSync()
    {
        var connection = dbProviderFactory.CreateConnection();
        if (connection is null)
        {
            throw new InvalidOperationException("DbProviderFactory.CreateConnection()
returned null");
        }
```

```
        connection.ConnectionString = this.ConnectionString;
        connection.Open();
        return connection;
    }
}
```

IConnectionFactory.cs

```
using System.Data.Common;
namespace Dal.Common;
public interface IConnectionFactory
{
    string ConnectionString { get; }
    string ProviderName { get; }
    Task<DbConnection> CreateConnectionAsync();
    DbConnection? CreateConnectionSync();
}
```

QueryParameter.cs

```
namespace Dal.Common;
public class QueryParameter
{
  public QueryParameter(string name, object? value)
  {
    this.Name = name;
    this.Value = value;
  }
  public string  Name { get; }
  public object? Value { get; }
}
```

Ordnername /Projekname : CaaS.Client

appsettings.json

```
{
  "ConnectionStrings": {
    "PersonDbConnection": {
        "ConnectionString": "Data
Source=(LocalDB)\\MSSQLLocalDB;AttachDbFilename=C:\\Users\\Administrator\\Documents\\Semest
er5\\SWK\\CaaS\\Db\\person_db_2019.mdf;Integrated Security=True;Connect Timeout=30",
        "ProviderName": "Microsoft.Data.SqlClient"
    }
  }
}
```

DalPersonTester.cs

```
using CaaS.Dal.Interface;
using CaaS.Domain;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Transactions;
namespace CaaS.Client;
internal class DalPersonTester
{
    // null prüfen
    private readonly IPersonDao personDao;
    private readonly string table ="Mandants";
    public DalPersonTester(IPersonDao personDao)
    {
        this.personDao = personDao;
    }
    public async Task TestFindAllAsync()
    {
        (await personDao.FindAllAsync(table))
            .ToList()
            .ForEach(p => Console.WriteLine($"{p.Id,5} | {p.FirstName,-10} | {p.LastName,-
15} | {p.DateOfBirth,10:yyyy-MM-dd}"));
    }
    public async Task TestFindByIdAsync()
    {
        Person? person1 = await personDao.FindByIdAsync("mandant-1",table);
        Console.WriteLine($"FindById(1) -> {person1?.ToString() ?? "<null>"}");
```

```
        Person? person2 = await personDao.FindByIdAsync("mandant-2",table);
        Console.WriteLine($"FindById(99) -> {person2?.ToString() ?? "<null>"}");
    }
    public async Task TestUpdateAsync()
    {
        Person? person = await personDao.FindByIdAsync("mandant-1",table);
        Console.WriteLine($"before update: person -> {person?.ToString() ?? "<null>"}");
        if (person is null)
        {
            return;
        }
        person.DateOfBirth = person.DateOfBirth.AddYears(-1);
        await personDao.UpdateAsync(person,table);
        person = await personDao.FindByIdAsync("mandant-1",table);
        Console.WriteLine($"after update:  person -> {person?.ToString() ?? "<null>"}");
    }
    public async Task TestDeleteByIdAsync()
    {
        Person? person = await personDao.FindByIdAsync("mandant-2", table);
        Console.WriteLine($"before update: person -> {person?.ToString() ?? "<null>"}");
        if (person is null)
        {
            return;
        }
        await personDao.DeleteByIdAsync(person.Id, table);
        person = await personDao.FindByIdAsync("mandant-2", table);
        Console.WriteLine($"after update:  person -> {person?.ToString() ?? "<null>"}");
    }
    public async Task TestStoreAsync()
    {
        Person? person = await personDao.FindByIdAsync("mandant-2", table);
        Console.WriteLine($"before update: person -> {person?.ToString() ?? "<null>"}");

        person = new Person("mandant-2", "ryo", "kimura", new DateTime(1971,12,7),
"ryokimura@example.com", "addr-m2", "mandant");
        await personDao.StoreAsync(person, table);
        person = await personDao.FindByIdAsync("mandant-2", table);
        Console.WriteLine($"after update:  person -> {person?.ToString() ?? "<null>"}");
    }
    public async Task TestTransactionsAsync()
    {
        Person? person1 = await personDao.FindByIdAsync("mandant-1",table);
        Person? person2 = await personDao.FindByIdAsync("mandant-2",table);
        if (person1 is null || person2 is null)
        {
            Console.WriteLine("Cannot perfom test because persons with id 1 and 2 are
required");
            return;
        }
```

```
        DateTime oldDate1 = person1.DateOfBirth;
        DateTime oldDate2 = person2.DateOfBirth;
        DateTime newDate1 = DateTime.MinValue;
        DateTime newDate2 = DateTime.MinValue;
        try
        {
            using (var scope = new
TransactionScope(TransactionScopeAsyncFlowOption.Enabled))
            {
                person1.DateOfBirth = newDate1 = oldDate1.AddDays(1);
                person2.DateOfBirth = newDate2 = oldDate2.AddDays(1);
                await personDao.UpdateAsync(person1,"Mandants");
                //throw new ArgumentException(); // comment this out to rollback
transaction
                await personDao.UpdateAsync(person2,"Mandants");
                scope.Complete();
            }
        }
        catch
        {
        }
        person1 = await personDao.FindByIdAsync("mandant-1",table);
        person2 = await personDao.FindByIdAsync("mandant-2",table);
        if (person1 is null || person2 is null)
        {
            Console.WriteLine("Cannot perfom test because persons with id 1 and 2 are
required");
            return;
        }
        if (oldDate1 == person1.DateOfBirth && oldDate2 == person2.DateOfBirth)
            Console.WriteLine("Transaction was ROLLED BACK.");
        else if (newDate1 == person1.DateOfBirth && newDate2 == person2.DateOfBirth)
            Console.WriteLine("Transaction was COMMITTED.");
        else
            Console.WriteLine("No Transaction.");
```

DalProductTester.cs

```
using CaaS.Dal.Interface;
using CaaS.Domain;
using Microsoft.Data.SqlClient;
using Microsoft.Practices.EnterpriseLibrary.ExceptionHandling;
using Org.BouncyCastle.Crypto.Encodings;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Transactions;
namespace CaaS.Client;
internal class DalProductTester
{
  // null prüfen
    private readonly IProductDao ProductDao;
    private readonly string table ="ProductShop1";
    private readonly string id1="arz-1";
    private readonly string id2="arz-2";
    public DalProductTester(IProductDao ProductDao)
    {
        this.ProductDao = ProductDao;
    }
    public async Task TestFindAllAsync()
    {
        (await ProductDao.FindAllAsync(table))
            .ToList()
            .ForEach(p => Console.WriteLine($"{p.Id,9} | {p.Name,80} | {p.Price,5} |"));
    }
    public async Task TestFindByIdAsync()
    {
        Product? product1 = await ProductDao.FindByIdAsync(id1,table);
        Console.WriteLine($"FindById{id1} -> {product1?.ToString() ?? "<null>"}");

        Product? product2 = await ProductDao.FindByIdAsync(id2,table);
        Console.WriteLine($"FindById{id2} -> {product2?.ToString() ?? "<null>"}");
    }
    public async Task TestUpdateAsync()
    {
        Product? product = await ProductDao.FindByIdAsync(id1,table);
        Console.WriteLine($"before updating: Product -> {product?.ToString() ??
"<null>"}");
        if (product is null)
```

```
        {
            return;
        }

        product.Price = 4.5;
        await ProductDao.UpdateAsync(product,table);
        product = await ProductDao.FindByIdAsync(id1,table);
        Console.WriteLine($"after updating:  Product -> {product?.ToString() ??
"<null>"}");
    }
    public async Task TestDeleteByIdAsync()
    {
        Product? product = await ProductDao.FindByIdAsync(id2, table);
        Console.WriteLine($"before deleting: Product -> {product?.ToString() ??
"<null>"}");
        await ProductDao.DeleteByIdAsync(product.Id, table);
        product = await ProductDao.FindByIdAsync(id2, table);
        Console.WriteLine($"after deleting:  Product -> {product?.ToString() ??
"<null>"}");
    }
    public async Task TestStoreAsync()
    {
        Product? product = await ProductDao.FindByIdAsync(id2, table);
        Console.WriteLine($"before storing: Product -> {product?.ToString() ?? "<null>"}");
        if (product == null)
        {
            product = new Product(id2, "(BAXTER) GLUCOSE 10% w/v ***", 11, "250 ml x 30
Viaflo Bags", "not yet", "download-link");
            await ProductDao.StoreAsync(product, table);
        }
        product = await ProductDao.FindByIdAsync(id2, table);
        Console.WriteLine($"after storing:  Product -> {product?.ToString() ?? "<null>"}");
    }
    public async Task TestTransactionsAsync()
    {
        Product? product1 = await ProductDao.FindByIdAsync(id1,table);
        Product? product2 = await ProductDao.FindByIdAsync(id2,table);
        if (product1 is null || product2 is null)
        {
            Console.WriteLine("Cannot perfom test because Products with id 1 and 2 are
required");
            return;
        }
        double oldPrice1 = product1.Price*1.5;
        double oldPrice2 = product2.Price*1.8;
        double newPrice1 = product1.Price/2;
        double newPrice2 = product2.Price/2;
        try
        {
```

```
            using (var scope = new
TransactionScope(TransactionScopeAsyncFlowOption.Enabled))
            {
                product1.Price=((double)(newPrice1 = (double)(oldPrice1)));
                product2.Price=((double)(newPrice2 = (double)(oldPrice2)));
                await ProductDao.UpdateAsync(product1,table);
                //throw new ArgumentException(); // comment this out to rollback
transaction
                await ProductDao.UpdateAsync(product2,table);
                scope.Complete();
            }
        }
        catch
        {
        }
        product1 = await ProductDao.FindByIdAsync(id1,table);
        product2 = await ProductDao.FindByIdAsync(id2,table);
        if (product1 is null || product2 is null)
        {
            Console.WriteLine("Cannot perfom test because Products with id 1 and 2 are
required");
            return;
        }

        if (oldPrice1 == product1.Price && oldPrice2 == product2.Price)
            Console.WriteLine("Transaction was ROLLED BACK.");
        else if (newPrice1 == product1.Price && newPrice2 == product2.Price)
            Console.WriteLine("Transaction was COMMITTED.");
        else
            Console.WriteLine("No Transaction.");
    }
}
```

DalShopTester.cs

```
using CaaS.Dal.Interface;
using CaaS.Domain;
using Org.BouncyCastle.Crypto.Encodings;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Transactions;
using static System.Reflection.Metadata.BlobBuilder;
namespace CaaS.Client;
internal class DalShopTester
{
    // null prüfen
    private readonly IShopDao shopDao;
    private readonly string table ="Shops";
    private readonly string id1="shop-1";
    private readonly string id2="shop-2"
    public DalShopTester(IShopDao shopDao)
    {
        this.shopDao = shopDao;
    }
    public async Task TestFindAllAsync()
    {
        (await shopDao.FindAllAsync(table))
            .ToList()
            .ForEach(p => Console.WriteLine($"{p.Id,9} | {p.Name,20} | {p.Address,5} |"));
    }
    public async Task TestFindByIdAsync()
    {
        Shop? shop1 = await shopDao.FindByIdAsync(id1,table);
        Console.WriteLine($"FindById {id1} -> {shop1?.ToString() ?? "<null>"}");

        Shop? shop2 = await shopDao.FindByIdAsync(id2,table);
        Console.WriteLine($"FindById {id2} -> {shop2?.ToString() ?? "<null>"}");
    }
    public async Task TestUpdateAsync()
    {
        Shop? shop = await shopDao.FindByIdAsync(id1,table);
        Console.WriteLine($"before update: Shop -> {shop?.ToString() ?? "<null>"}");
        if (shop is null)
        {
            return;
        }

        shop.Address= "addr-sh5";
        await shopDao.UpdateAsync(shop,table);
```

```
        Console.WriteLine($"after update: Shop -> {shop?.ToString() ?? "<null>"}");
        (await shopDao.FindAllAsync(table))
            .ToList()
            .ForEach(p => Console.WriteLine($"{p.Id,9} | {p.Name,20} | {p.Address,5} |"));
    }
    public async Task TestDeleteByIdAsync()
    {
        Shop? shop = await shopDao.FindByIdAsync(id2, table);
        Console.WriteLine($"before delete: Shop -> {shop?.ToString() ?? "<null>"}");
        if (shop is null)
        {
            return;
        }
        Console.WriteLine($"Deleting: ");
        await shopDao.DeleteByIdAsync(id1, table);
        await shopDao.DeleteByIdAsync(shop.Id, table);
        Console.WriteLine($"after delete: ");
        (await shopDao.FindAllAsync(table))
            .ToList()
            .ForEach(p => Console.WriteLine($"{p.Id,9} | {p.Name,20} | {p.Address,5} |"));

    public async Task TestStoreAsync()
    {
        Shop? shop = new Shop("shop-1", "LoveRead", "books", "mandant-1", "addr-sh1");
        await shopDao.StoreAsync(shop, table);
        Console.WriteLine($"before update: Shop -> {shop?.ToString() ?? "<null>"}");
        shop = new Shop("shop-2", "DrugLinz", "Medicaments", "mandant-2", "addr-sh2");
        await shopDao.StoreAsync(shop, table);
        shop = new Shop("shop-3","ScienceBooks", "Books mostly about science", "addr-sh3",
"download-link");
        await shopDao.StoreAsync(shop, table);
        Console.WriteLine($"after update: ");
        (await shopDao.FindAllAsync(table))
           .ToList()
           .ForEach(p => Console.WriteLine($"{p.Id,9} | {p.Name,20} | {p.Address,5} |"));
    }
    public async Task TestTransactionsAsync()
    {
        Shop? shop1 = await shopDao.FindByIdAsync(id1,table);
        Shop? shop2 = await shopDao.FindByIdAsync(id2,table);
        if (shop1 is null || shop2 is null)
        {
            Console.WriteLine("Cannot perfom test because Shops with id 1 and 2 are
required");
            return;
        }
        string oldAddress1 = shop1.Address;
        string oldAddress2 = shop2.Address;
        string newAddress1 = "addr-sh3";
```

```
        string newAddress2 = "addr-sh4";
        try
        {
            using (var scope = new
TransactionScope(TransactionScopeAsyncFlowOption.Enabled))
            {
                shop1.Address=((newAddress1 = (oldAddress1 )));
                shop2.Address=((newAddress2 = (oldAddress2 )));
                await shopDao.UpdateAsync(shop1,table);
                //throw new ArgumentException(); // comment this out to rollback
transaction
                await shopDao.UpdateAsync(shop2,table);
                scope.Complete();
            }
        }
        catch
        {
        }
        shop1 = await shopDao.FindByIdAsync(id1,table);
        shop2 = await shopDao.FindByIdAsync(id2,table);
        if (shop1 is null || shop2 is null)
        {
            Console.WriteLine("Cannot perfom test because Shops with id 1 and 2 are
required");
            return;
        }

        if (oldAddress1 == shop1.Address && oldAddress2 == shop2.Address)
            Console.WriteLine("Transaction was ROLLED BACK.");
        else if (newAddress1 == shop1.Address && newAddress2 == shop2.Address)
            Console.WriteLine("Transaction was COMMITTED.");
        else
            Console.WriteLine("No Transaction.");
    }
}
```

Program.cs

```
using Dal.Common;
using Microsoft.Extensions.Configuration;
using CaaS.Client;
using CaaS.Dal.Ado;
//using PersonAdmin.Dal.Simple;

static void PrintTitle(string text = "", int length = 60, char ch = '-')
{
  int preLen = (length - (text.Length + 2)) / 2;
  int postLen = length - (preLen + text.Length + 2);
  Console.WriteLine($"{new String(ch, preLen)} {text} {new String(ch, postLen)}");
}
IConfiguration configuration = ConfigurationUtil.GetConfiguration();
IConnectionFactory connectionFactory =
    DefaultConnectionFactory.FromConfiguration(configuration, "CaaSConnection");
var tester2 = new DalPersonTester(new AdoPersonDao(connectionFactory));
PrintTitle("AdoPersonDao.FindAll", 50);
await tester2.TestFindAllAsync();
PrintTitle("AdoPersonDao.FindById", 50);
await tester2.TestFindByIdAsync();
PrintTitle("AdoPersonDao.Update", 50);
await tester2.TestUpdateAsync();
PrintTitle("AdoPersonDao.Delete", 50);
await tester2.TestDeleteByIdAsync();
PrintTitle("AdoPersonDao.Store", 50);
await tester2.TestStoreAsync();
PrintTitle("Transactions", 50);
await tester2.TestFindAllAsync();
await tester2.TestTransactionsAsync();
await tester2.TestFindAllAsync();
/================================================================================
var tester3 = new DalProductTester(new AdoProductDao(connectionFactory));
PrintTitle("AdoProductDao.FindAll", 50);
await tester3.TestFindAllAsync();
PrintTitle("AdoProductDao.FindById", 50);
await tester3.TestFindByIdAsync();
PrintTitle("AdoProductDao.Update", 50);
await tester3.TestUpdateAsync();
PrintTitle("AdoProductDao.Delete", 50);
await tester3.TestDeleteByIdAsync();
PrintTitle("AdoProductDao.Store", 50);
await tester3.TestStoreAsync();
PrintTitle("Transactions", 50);
await tester3.TestFindAllAsync();
await tester3.TestTransactionsAsync();
await tester3.TestFindAllAsync();
```

```
//=============================================================================================
var tester4 = new DalShopTester(new AdoShopDao(connectionFactory));
PrintTitle("AdoShopDao.FindAll", 50);
await tester4.TestFindAllAsync();
PrintTitle("AdoShopDao.FindById", 50);
await tester4.TestFindByIdAsync();
PrintTitle("AdoShopDao.Update", 50);
await tester4.TestUpdateAsync();
PrintTitle("AdoShopDao.Delete", 50);
await tester4.TestDeleteByIdAsync();
PrintTitle("AdoShopDao.Store", 50);
await tester4.TestStoreAsync();
PrintTitle("Transactions", 50);
await tester4.TestFindAllAsync();
await tester4.TestTransactionsAsync();
await tester4.TestFindAllAsync();
```