|  |
| --- |
| Zentrum für berufliche Weiterbildung |
| Projektarbeit - Auftragsverwaltung |
| DBA & PGA II |

|  |
| --- |
| Leonie Buser & Benjamin Peter  20.09.2022 |

Inhaltsverzeichnis

[Ausgangslage 2](#_Toc114661494)

[ERM 2](#_Toc114661495)

[Use-Case 2](#_Toc114661496)

[CRUD-Funktionen 2](#_Toc114661497)

[Verwaltung der Daten 3](#_Toc114661498)

[Artikelgruppe 3](#_Toc114661499)

[Jahresvergleich 3](#_Toc114661500)

[Rechnung 3](#_Toc114661501)

[Repository 4](#_Toc114661502)

[Serialisierung 4](#_Toc114661503)

[Regex 4](#_Toc114661504)

[Inbetriebnahme 4](#_Toc114661505)

[Arbeitsjournal 5](#_Toc114661506)

[Benjamin & Leonie 5](#_Toc114661507)

[Benjamin 5](#_Toc114661508)

[Leonie 5](#_Toc114661509)

[Verzeichnisse 6](#_Toc114661510)

[Abbildungen 6](#_Toc114661511)

[Tabellen 6](#_Toc114661512)

# Ausgangslage

Es soll für die Muster AG eine Datenbankapplikation entwickelt werden, in der folgende Daten verwaltet werden können:

* Kunden und Adressen
* Artikel und Artikelgruppen
* Aufträge und Auftragspositionen

Für die Applikation soll folgender Technologie-Stack verwendet werden:

* Windows Desktop Applikation mit C#
* Präsentation-Layer wahlweise mit Windows Forms oder WPF
* Data-Layer mit Entity Framework und CodeFirst
* MS SQL-Server

# ERM

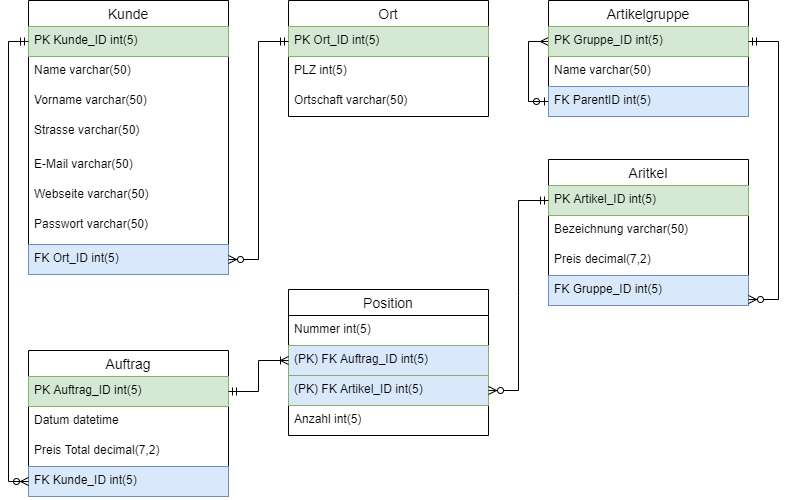


Abb. 1: ERM

# Use-Case

## CRUD-Funktionen

Die Applikation ermöglicht das Erstellen, Lesen, Bearbeiten und Löschen von folgenden Positionen:

* Kunden
* Artikelgruppen
* Artikel
* Aufträge

## Verwaltung der Daten

Nachstehende Attribute können verwaltet werden:

* Kunden & Adressen:  
  Kundennummer, Name, Strasse, PLZ, Ort, E-Mail, Webseite, Passwort
* Artikelgruppe:  
  Namen, übergeordnete Artikelgruppe
* Artikel:  
  Artikelnummer, Bezeichnung, Preis, Artikelgruppe
* Aufträge & Positionen  
  Auftragsnummer, Datum, Kunde; Positionen mit Nummer, Artikel Anzahl

### Artikelgruppe

Für die hierarchische Anzeige von Artikelgruppen wurde eine rekursive CTE erstellt. Diese steht in der Applikation im Hauptfenster im Register «Gruppen» in Form einer «TreeView» zur Verfügung.

Momentan können Gruppen auf 2 Ebenen erstellt werden.

## Jahresvergleich

Mittels Window-Functions können diverse Statistikdaten der letzten 3 Jahre ausgewertet werden. Folgende Informationen sind ersichtlich:

* Anzahl Aufträge
* Anzahl verwaltete Artikel
* Durchschnittliche Anzahl Artikel pro Auftrag
* Gesamtumsatz
* Umsatz pro Kunde

## Rechnung

Für die Rechnungen haben wir uns dagegen entschieden, eine separate Tabelle zu erstellen. Als Rechnungsnummer & -datum haben wir direkt die Daten aus der Order-Tabelle verwendet. Der Brutto-Preis wird direkt bei der Abfrage berechnet.

Der Hauptgrund, warum wir uns für dieses Vorgehen entschieden haben ist, Redundante Daten zu vermeiden.

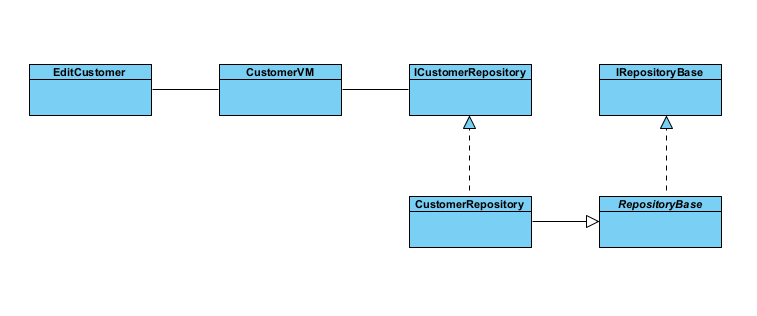
Mittels eines Filters kann der Datumsbereich eingegrenzt werden.

Die Rechnungen werden mit den zum Zeitpunkt der Bestellung gültigen Daten angezeigt (Kunden- & Artikeldaten).

## Repository

Im UML sieht man am Beispiel des Kunden (Customer) wie die Klassen zusammenhängen.

Die View (EditCustomer) erstellt mithilfe eines IoC Containers das ViewModel und injiziert dabei die nötigen Repository-Interfaces. Die Repositories selbst erben vom RepositoryBase, welches wiederum das IRepositoryBase Interface implementiert. In diesem Interface sind die Allgemeinen Repository Methoden aufgeführt.



## Serialisierung

Für die Serialisierung haben wir eine Separate DTO-Klasse erstellt, um die Datenbank-Tabellen für die Verarbeitung in einer Klasse zusammenzufassen.

Eine schönere Variante wäre eigentlich solche Domain-Klassen und Aggregate für alle Daten, mit denen das ViewModel arbeitet zu erstellen und zu verwenden. Somit wären EntityFramework und Domain Klassen klar getrennt und man könnte für den XML und JSON Import/Export auch die ICustomerRepository Schnittstellen verwenden.

## Regex

Die Regex-Überprüfung wurde in der Customer Klasse realisiert und wird vom ViewModel aufgerufen.

Weggelassen haben wir die Überprüfen der Kunden-Nummer mit «/^CU\d+$/» da wir automatisch generierte Integer Primärschlüssel als Kunden-Nummer verwenden.

# Inbetriebnahme

Das Programm kann von GIT heruntergeladen oder geklont werden. Der Link dazu ist:  
<https://github.com/narrog/DbAdvPrgAdv_Auftragsverwaltung.git>

Die MS SQL-Datenbank inkl. Testdaten wird beim Starten des Programmes über Visual Studio automatisch generiert. Der Name der DB lautet: DbAdvPrgAdv\_Auftragsverwaltung.

Für die produktive Verwendung müsste die Applikation so verändert werden, dass die Migrationen manuell ausgeführt werden. So sind keine Testdaten in der Datenbank.

# Arbeitsjournal

## Benjamin & Leonie

Tabelle : Arbeitsjournal Benjamin & Leonie

|  |  |
| --- | --- |
| ERM | 1.00 h |
| Entity Frameworks | 1.00 h |
| Windows erstellen (WPF) | 1.00 h |
| Buttons konfigurieren | 2.00 h |
| Bestellungen CRUD | 4.00 h |
| Jahresvergleich | 2.00 h |
| Suche | 2.00 h |
| Repository Pattern implementiert | 4.00 h |
| IoC Container | 2.00 h |
|  |  |
| **Total** | **19.00 h** |

## Benjamin

Tabelle : Arbeitsjournal Benjamin

|  |  |
| --- | --- |
| Entity Frameworks | 1.00 h |
| Kunden CRUD | 2.00 h |
| Temporale Datenbank erstellt | 0.50h |
| Hilfe bei CTE | 0.50 h |
| Bestellungen fertig gestellt | 1.00 h |
| Jahresvergleich | 5.00 h |
| Repository Interface und Base | 2.00 h |
| Serialisierung | 3.00 h |
|  |  |
| **Total** | **15.00 h** |

## Leonie

Tabelle : Arbeitsjournal Leonie

|  |  |
| --- | --- |
| CTE für Gruppen | 1.00 h |
| Gruppen CRUD | 1.00 h |
| Artikel CRUD | 1.00 h |
| Rechnungsübersicht | 3.00 h |
| Fix Artikel / Gruppen / Kunden | 3.00 h |
| Dokumentation | 1.50 h |
| Regex | 1.00 h |
| Regex Tests | 0.50 h |
|  |  |
| **Total** | **12.00 h** |

# Verzeichnisse

## Abbildungen

[Abb. 1: ERM 2](#_Toc89173631)

## Tabellen

[Tabelle 1: Arbeitsjournal Benjamin & Leonie 4](#_Toc97321097)

[Tabelle 2: Arbeitsjournal Benjamin 4](#_Toc97321098)

[Tabelle 3: Arbeitsjournal Leonie 4](#_Toc97321099)