



Dokumentation



Inhalt

1	Applikation Inbetriebnahme	II
1.1	Download	II
1.2	Datenbank.....	II
1.3	Testdaten	II
2	Unser Vorgehen	III
2.1	Sourcecode Versionierung	III
2.2	Kanban.....	III
3	UI	IV
3.1	Mockups vs Endergebnis	IV
3.2	Modal Forms	V
4	CTE – Categories View	VII
5	ERM.....	VIII
6	Klassendiagramm.....	IX
6.1	Links Views	IX
6.1.1	Forms.....	IX
6.1.2	Views	IX
6.2	Mitte Services.....	X
6.3	Rechts DB-Entities	X
6.3.1	Ausnahmen	X
6.4	Assoziationen.....	XI
7	TODO.....	XI
8	TODO.....	XI
9	TODO.....	XI
10	Rückblick auf das Projekt.....	XI
10.1	Raphael.....	XI
10.2	Ricardo.....	XI

1 Applikation Inbetriebnahme

1.1 Download

Der Sourcecode von *order-management* wird mit Git verwaltet. Als Plattform wird www.github.com verwendet. Um den Sourcecode herunter zu laden kann entweder dieser Command ausgeführt werden:

```
git clone https://github.com/ricardo17coelho/order-management.git
```

Oder er kann von folgender URL manuell als .zip heruntergeladen werden:

- <https://github.com/ricardo17coelho/order-management>

1.2 Datenbank

Order-Management braucht zwingend eine laufende MSSQL-Datenbank, die auf dem gleichen Host läuft, wie die Applikation. Der angemeldete Windows-User muss sich zwingend mit dieser Datenbank verbinden dürfen. (`Trusted_Connection=True`)

Bevor man die Applikation startet, muss man die Datenbank vorbereiten. Dafür sollte das Visual Studio Projekt geöffnet werden und in der Package Manager Console folgender Command ausgeführt werden:

```
Update-Database
```

Dies generiert die nötigen Tabellen mit allen dazugehörigen Spalten und Beziehungen.

1.3 Testdaten

Dieser Command generiert nicht nur sämtliche Tabellen, sondern auch einige Testdaten, damit *order-management* dann auch getestet werden kann. Dies wurde in der Methode `onModelCreating()` im `DbContext` implementiert.

2 Unser Vorgehen

2.1 Sourcecode Versionierung

Uns war von Anfang an klar, dass wir den Sourcecode mit Git auf Github verwalten wollten.

2.2 Kanban

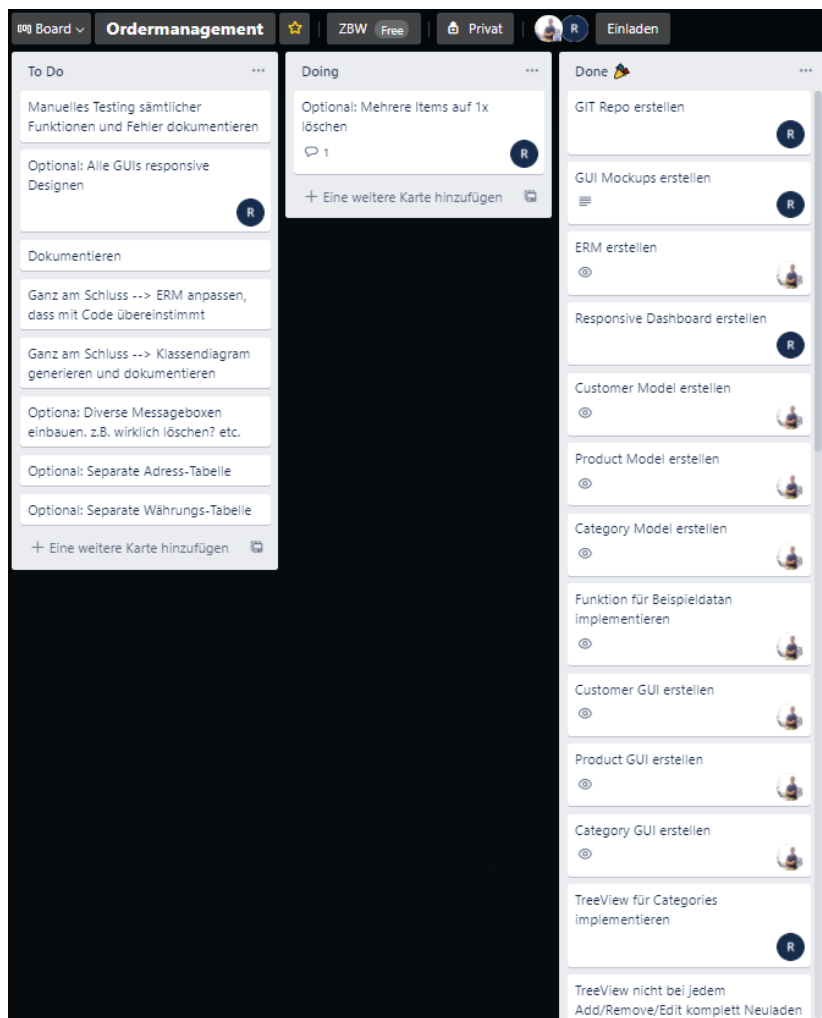
Als zusätzliche Hilfe für die Übersicht der Tasks verwendeten wir ein Kanban-Board auf www.trello.com.

Anfangs definierten wir einige Tasks und teilten sie einander zu. Die ersten Tasks waren:

- Ricardo: GIT Repo erstellen
- Ricardo: GUI Mockups erstellen
- Raphael: ERM erstellen
- Ricardo: Dashboard (Homescreen) erstellen
- Raphael: Sämtliche Entity Models erstellen

Danach erstellten wir immer weitere Tasks, priorisierten sie und teilten sie einander zu.

Dies ist nur ein kleiner Ausschnitt von unserem Kanban-Board:



3 UI

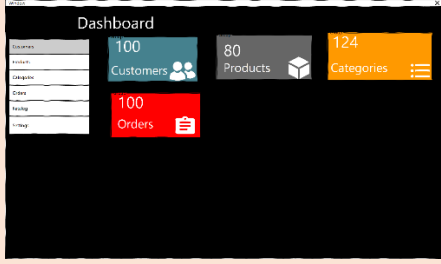
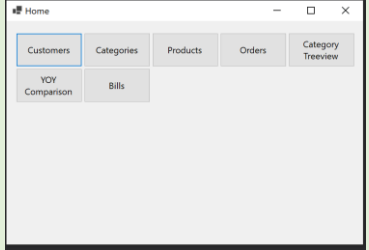
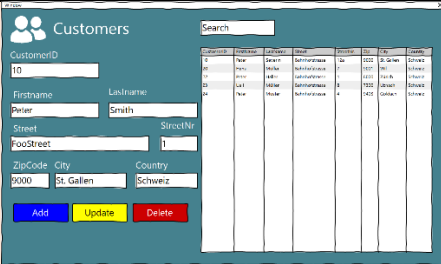
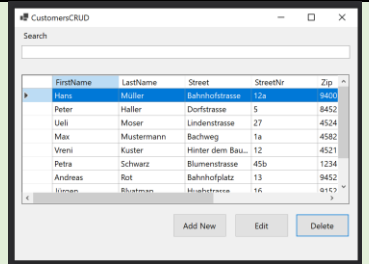
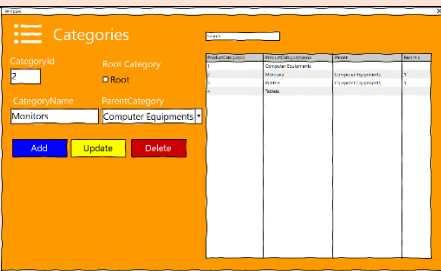
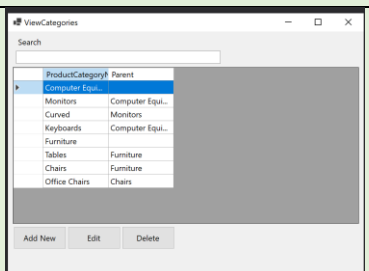
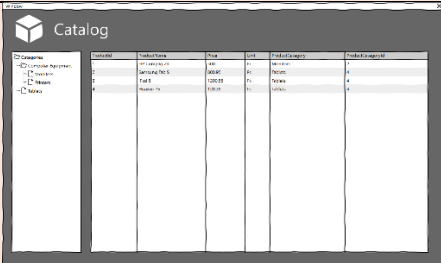
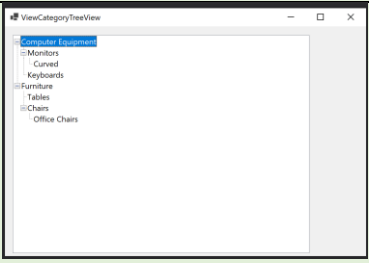
3.1 Mockups vs Endergebnis

Warum abweicht unser Endprodukt von erstellten Mockups?

Am Anfang wollten wir ein einfaches Design für all die Views gestalten und mit Hilfe von verschiedener Farbe, die «views» unterscheiden.

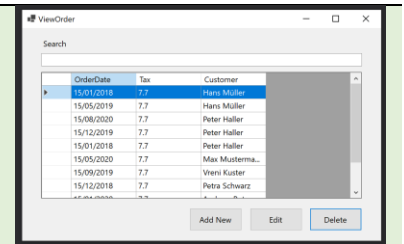
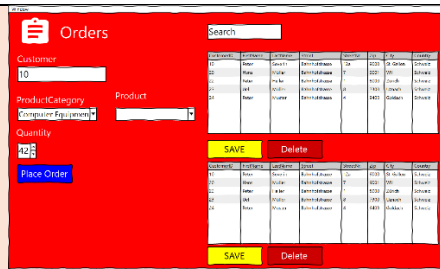
Jedoch haben wir schnell gemerkt, dass WindowsForms sehr limitiert ist.

Sowohl ich als auch Raphael sind daran gewohnt mit guten Frameworks in WEB Bereich zu arbeiten, welche uns viele Möglichkeiten anbieten und deswegen, dachten wir dass wir hier auch umsetzen könnten, aber das war nicht wirklich möglich.

Vor-/Nachteile	Mockup	Ergebnis
<ul style="list-style-type: none"> + Schnellzugriff + Zähler - Responsive - Mehrmals gleiche Aktion 		
<ul style="list-style-type: none"> + Keine Zusätzliche Fenster - Unübersichtlich - Responsive 		
<ul style="list-style-type: none"> + Keine Zusätzliche Fenster - Unübersichtlich - Responsive 		
<ul style="list-style-type: none"> + Zugriff auf Produkte 		

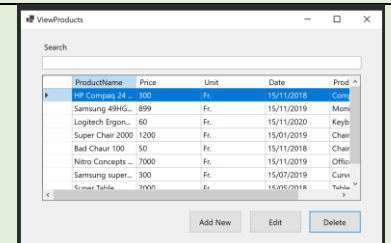
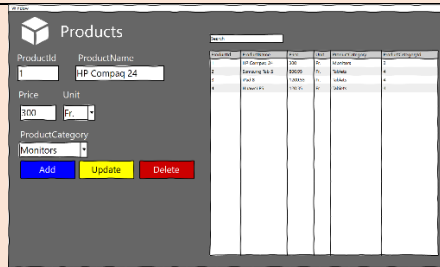
+ Keine Zusätzliche Fenster

- Unübersichtlich
- Responsive



+ Keine Zusätzliche Fenster

- Unübersichtlich
- Responsive



+ Diverse Einstellung auf eine Seite



3.2 Modal Forms

Wir haben uns am Anfang entschieden, alle Grids + Formulare auf ein Panel darzustellen.

Im Verlauf von der Entwicklung haben wir nun gemerkt, dass das nicht so wirklich übersichtlich ist. Deswegen haben wir uns dann entschieden, all die Formulare in einer separaten Modal Fenster anzuzeigen.

Somit erreichen wir ein besseres Design für unsere Grid und die ADD und DELETE Operationen, können sauber über dieses Modal Dialog gelöst werden.

Beim Klicken auf «Add» wird das Formular gelöscht und eine neuen Datensatz kann erfasst werden.

Wenn 1 Datensatz im Grid ausgewählt ist, dann kann man auf «Edit» Button drücken, das Modal wird geöffnet und die Felder werden mit der ausgewählte Datensatz gefüllt.

Beim Speichern wird dann beachtet, ob die Datensatz bereits ein ID hat oder nicht.

Falls ja, werden die Felder in der DB geändert, falls nicht, wird eine neue Datensatz in der DB angelegt.

Add Mode

FormProducts

Product Name:

Price:

Unit:

Introduction Date:

Category:

FormCategories

Category Name:

☐ Root Category

Parent Category:

CustomersForm

First Name:

Last Name:

Street:

StreetNr:

Zip:

City:

Country:

FormOrders

Customer:

Order Date:

Tax:

Positions:

Edit Mode

FormProducts

Product Name:

Price:

Unit:

Introduction Date:

Category:

FormCategories

Category Name:

☒ Root Category

Parent Category:

CustomersForm

First Name:

Last Name:

Street:

StreetNr:

Zip:

City:

Country:

FormOrders

Customer:

Order Date:

Tax:

Positions:

Quantity	Product
2	Samsung 49HG...
4	Bad Chaur 100
4	Nitro Concepts ...
1	Logitech Ergon...
2	Super Chair 2000

4 CTE – Categories View

«No pain, No gain».

So kann ich am besten die CTE beschreiben, nach dem hier viel Zeit investiert wurde.

```
var query = @"WITH RecurseTable " +
    "(ProductCategoryId, ProductCategoryName, ParentId, Level) " +
    "AS (SELECT " +
        "ProductCategoryId," +
        "ProductCategoryName," +
        "ISNULL(ParentId, 0)," +
        "0 AS Level " +
    "FROM ProductCategories " +
    "WHERE ParentId IS NULL " +
    "UNION ALL " +
    "SELECT " +
        "pcat.ProductCategoryId," +
        "pcat.ProductCategoryName," +
        "pcat.ParentId," +
        "Level + 1 " +
    "FROM ProductCategories AS pcat " +
    "INNER JOIN RecurseTable AS rec " +
        "ON rec.ProductCategoryId = pcat.ParentId " +
    ") " +
    "SELECT " +
        "ProductCategoryId," +
        "ProductCategoryName," +
        "ParentId," +
        "Level " +
    "FROM RecurseTable";
```

	ProductCategoryId	ProductCategoryName	ParentId	Level
1	1000	Computer Equipment	0	0
2	1004	Furniture	0	0
3	1005	Tables	1004	1
4	1006	Chairs	1004	1
5	1007	Office Chairs	1006	2
6	1001	Monitors	1000	1
7	1003	Keyboards	1000	1
8	1002	Curved	1001	2

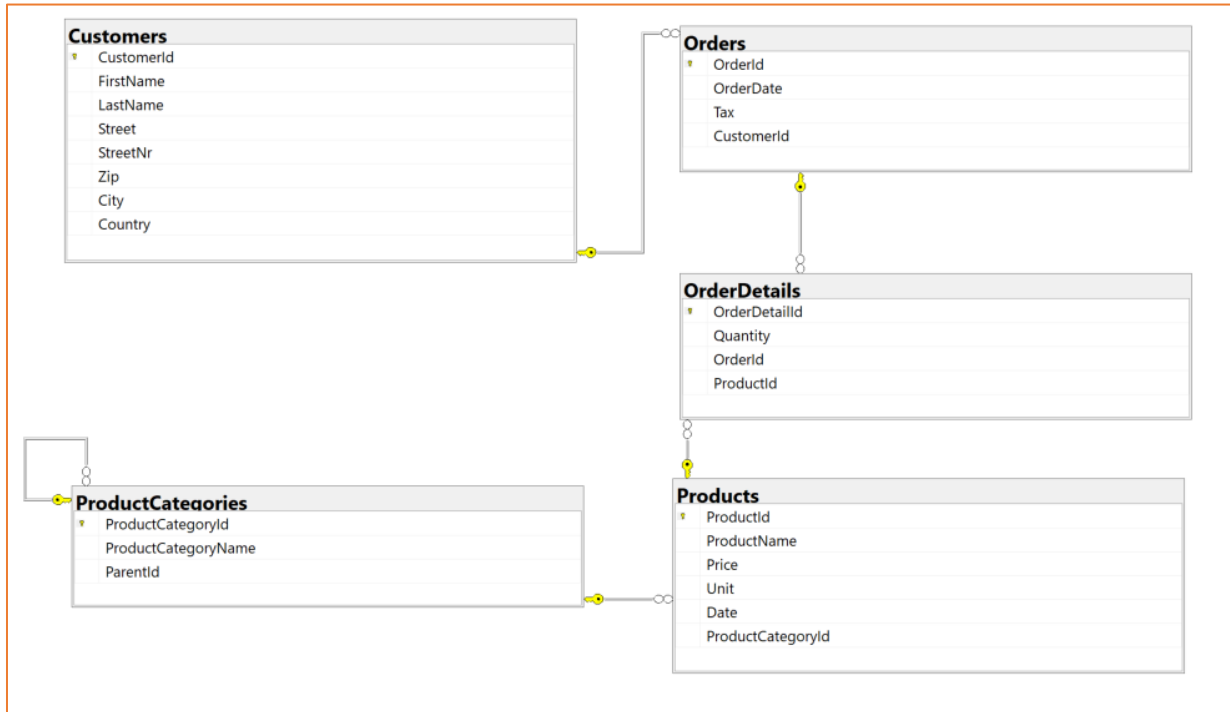
Die Aufruf von C# code erfolgt mit als Query string mit Hilfe von EntityFrameworkCore.

Danach erhalten wir als Ergebnis eine Tabelle mit ID der Kategorie, Beschreibung, ParentID, welche Identifiziert zu welche übergeordnet Kategorie diese Kategorie gehört und als letzten mit «Level» wird angezeigt zu welchem Level sich diese Kategorie gehört.

Danach wird diese Liste iteriert. Als erstes nimmt man ein Node vom Level 0 und dann werden alle items mit dem gleichen ParentID zu diesen Node hinzugefügt bis alle items in der TreeView hinzugefügt wurden.

5 ERM

Aufgrund der Vorgaben, wurde folgendes ERM entworfen und im EF Code-First Ansatz entwickelt.



6 Klassendiagramm

Folgendes Diagramm zeigt der visuelle Aufbau der Applikation.



Das Ganze von links nach rechts in 3 Bereiche aufgeteilt werden.

6.1 Links Views

Auf der linken Seite befinden sich sämtliche Views. Wir unterscheiden immer zwischen Forms und Views (und dem Homescreen, welcher den Eintritt in die Applikation abbildet).

6.1.1 Forms

Forms sind Formulare, welche Textfelder enthalten, wo der User Daten eingeben kann. Diese Daten werden dann schlussendlich auf der Datenbank abgespeichert.

6.1.2 Views

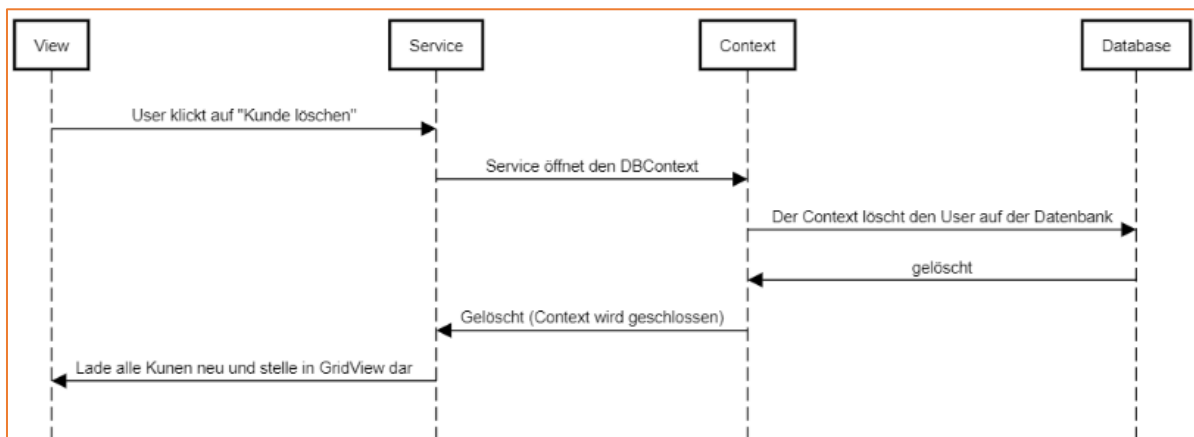
Auf den Views befindet sich jeweils lediglich ein Data-Grid-View welches Daten veranschaulicht.

6.2 Mitte Services

Für jedes Entity gibt es einen dazugehörigen Service. Diese bilden jeweils die Verbindung zur Datenbank ab. Dort drin gibt es jeweils Methoden wie z.B.:

- GetAll()
- GetByName(string name)
- Add(Entity entity)
- Delete(Entity entity)
- Etc.

Jede Methode, die eine Verbindung zur Datenbank benötigt, kümmert sich selbst um das Öffnen und Schliessen vom DbContext, damit dieser immer nur möglichst kurz offen bleibt.



6.3 Rechts DB-Entities

Ganz rechts befinden sich die DB-Entities, welche so auch im ERM wiederzufinden sind.

6.3.1 Ausnahmen

Folgende Klassen sind Ausnahmen und findet man so in der Datenbank nicht.

- YoyComparison
- Bill

In der Methode `OnModelCreating()` im `DbContext` wurde definiert, dass für diese zwei Klassen keine Tabelle erzeugt werden soll:

```
modelBuilder.Entity<YoyComparison>().HasNoKey().ToView(null);
modelBuilder.Entity<Bill>().HasNoKey().ToView(null);
```

Diese zwei Klassen werden lediglich als «Query-Objekte» verwendet. D.h. Diese Objekte werden nicht in der Datenbank gespeichert, aber wir benötigen sie als Objektstruktur für gewisse Queries und für die Darstellung im `ViewYoyComparison` und im `ViewBill`.

6.4 Assoziationen

Der Übersichtshalber wurden absichtlich nicht sämtliche Assoziationen eingezeichnet. Konkret gemeint ist damit z.B. das `FormProduct`.

Gemäss ERM hat jedes Produkt eine dazugehörige Produkt-Kategorie. Das heisst man muss diese beim Erfassen eines Produktes auswählen können. Damit das möglich ist, verwendet das `FormProduct` nicht NUR den `ProductService`, sondern auch den `ProductCategoryService`.

Genau gleich, wie z.B. das `FormOrder` zusätzlich auch den `ProductService` UND den `CustomerService` verwendet etc.

7 TODO

8 TODO

9 TODO

10 Rückblick auf das Projekt

10.1 Raphael

10.2 Ricardo

Für mich war dieses Projekt zu aufwendig. Ich finde sehr schade, dass nicht benotet wird.

Ich könnte in diesem Projekt den gelernten Stoff, welche in dem Unterricht ermittelt wurde, umsetzen und habe viel dabei gelernt.

Am Anfang hatten wir ein grober Plan, wie viel all die Ansichten darstellen wollten, jedoch hat sich im Verlauf vom Projekt viel geändert.

Da ich nicht so viele Erfahrung mit C#/.Net habe war für mich ein bisschen schwierig all diesen Abhängigkeiten am Anfang zu verstehen.

Gemeinsam mit Raphael habe ich es dann langsam verstanden und wir haben eine «interessante» Lösung entwickelt.