

**Dokumentation**

Inhalt

[1 Applikation Inbetriebnahme II](#_Toc62467113)

[1.1 Download II](#_Toc62467114)

[1.2 Datenbank II](#_Toc62467115)

[1.3 Testdaten II](#_Toc62467116)

[2 Unser Vorgehen III](#_Toc62467117)

[2.1 Sourcecode Versionierung III](#_Toc62467118)

[2.2 Kanban III](#_Toc62467119)

[3 Mockups IV](#_Toc62467120)

[4 ERM V](#_Toc62467121)

[5 Klassendiagramm VI](#_Toc62467122)

[5.1 Links Views VI](#_Toc62467123)

[5.1.1 Forms VI](#_Toc62467124)

[5.1.2 Views VI](#_Toc62467125)

[5.2 Mitte Services VII](#_Toc62467126)

[5.3 Rechts DB-Entities VII](#_Toc62467127)

[5.3.1 Ausnahmen VII](#_Toc62467128)

[5.4 Assoziationen VII](#_Toc62467129)

# Applikation Inbetriebnahme

## Download

Der Sourcecode von *order-management* wird mit Git verwaltet. Als Plattform wird [www.github.com](http://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>

## 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.

## 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.

# Unser Vorgehen

## Sourcecode Versionierung

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

## Kanban

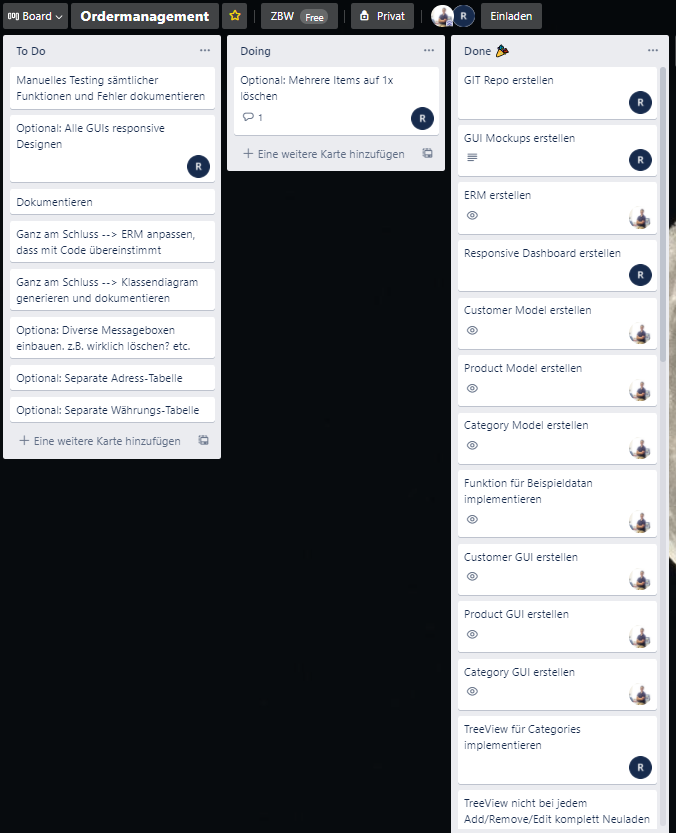
Als zusätzliche Hilfe für die Übersicht der Tasks verwendeten wir ein Kanban-Board auf [www.trello.com](http://www.trello.com).

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.

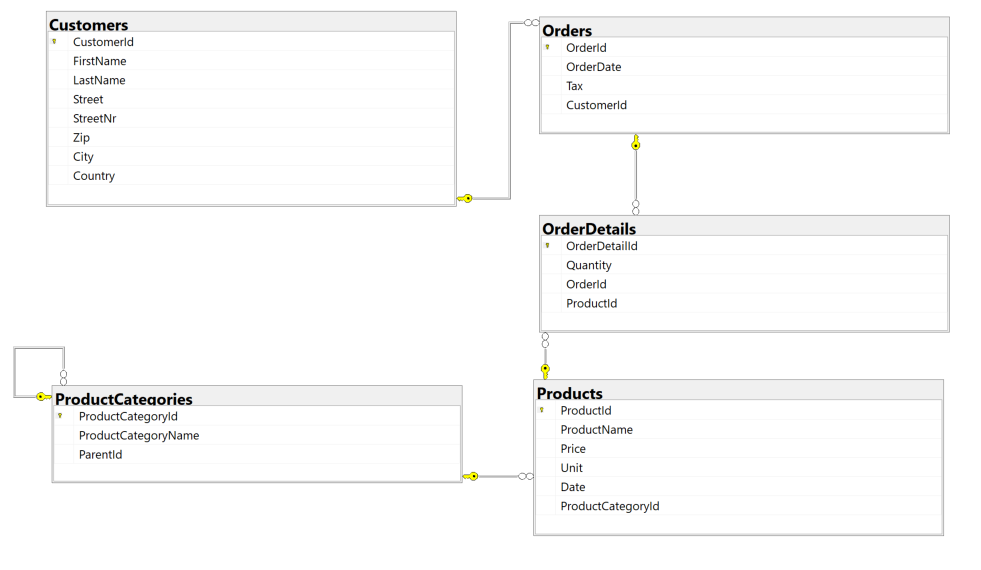
Dies ist nur ein kleiner Ausschnitt von unserem Kanban-Board:



# Mockups

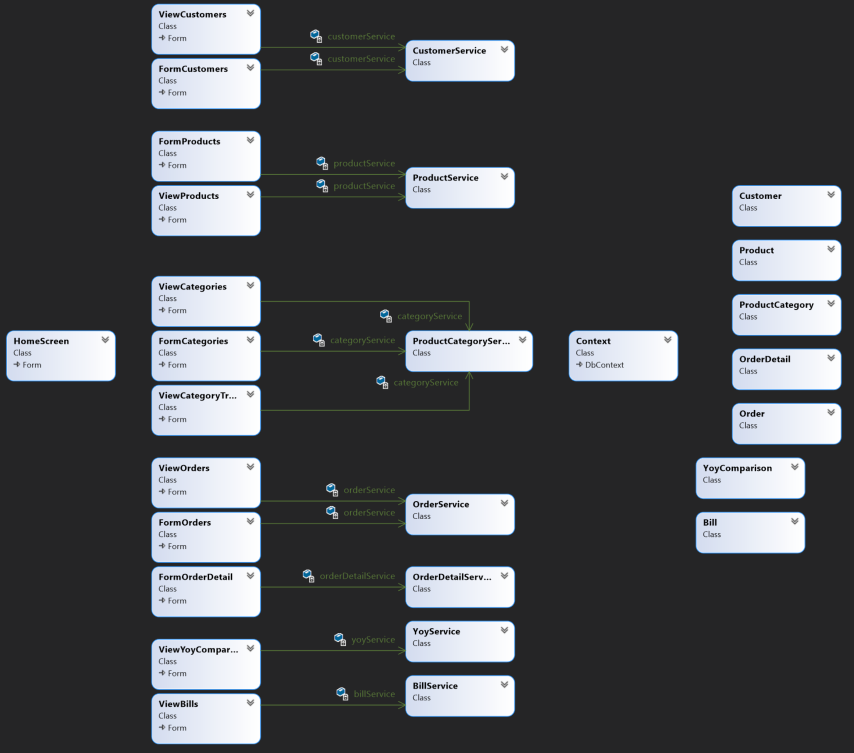
# ERM

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



# Klassendiagramm

Folgendes Diagramm zeigt der visuelle Aufbau der Applikation.



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

## 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).

### Forms

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

### Views

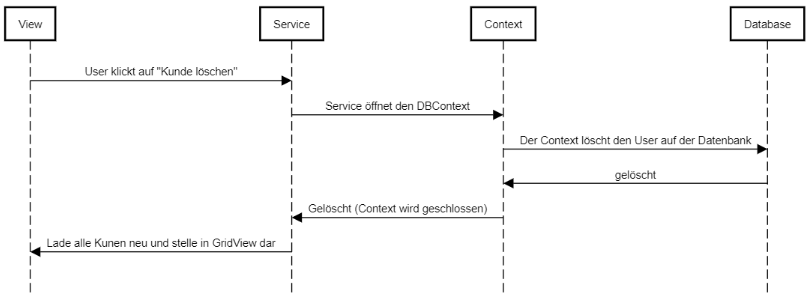
Auf den Views befindet sich jeweils lediglich ein Data-Grid-View welches Daten veranschaulicht.

## 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.:

* GettAll()
* 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.



## Rechts DB-Entities

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

### 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.

## 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.

# TODO

# TODO

# TODO

# Rückblick auf das Projekt

## Raphael

## Ricardo